

30th International Symposium on Algorithms and Computation

ISAAC 2019, December 8–11, 2019, Shanghai University of
Finance and Economics, Shanghai, China

Edited by

Pinyan Lu

Guochuan Zhang



Editors

Pinyan Lu

Shanghai University of Finance and Economics, China
lu.pinyan@mail.shufe.edu.cn

Guochuan Zhang

Zhejiang University, China
zgc@zju.edu.cn

ACM Classification 2012

Theory of computation; Theory of computation → Models of computation; Theory of computation → Computational complexity and cryptography; Theory of computation → Randomness, geometry and discrete structures; Theory of computation → Theory and algorithms for application domains; Theory of computation → Design and analysis of algorithms

ISBN 978-3-95977-130-6

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/978-3-95977-130-6>.

Publication date

December, 2019

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 3.0 Unported license (CC-BY 3.0): <https://creativecommons.org/licenses/by/3.0/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.2019.0

ISBN 978-3-95977-130-6

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*, Gran Sasso Science Institute and Reykjavik University)
- Christel Baier (TU Dresden)
- Mikolaj Bojanczyk (University of Warsaw)
- Roberto Di Cosmo (INRIA and University Paris Diderot)
- Javier Esparza (TU München)
- Meena Mahajan (Institute of Mathematical Sciences)
- Dieter van Melkebeek (University of Wisconsin-Madison)
- Anca Muscholl (University Bordeaux)
- Luke Ong (University of Oxford)
- Catuscia Palamidessi (INRIA)
- Thomas Schwentick (TU Dortmund)
- Raimund Seidel (Saarland University and Schloss Dagstuhl – Leibniz-Zentrum für Informatik)

ISSN 1868-8969

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

■ Contents

Preface	
<i>Pinyan Lu and Guochuan Zhang</i>	0:ix
Symposium Organization	
.....	0:xi–0:xv
Graph Searches and Their End Vertices	
<i>Yixin Cao, Zhifeng Wang, Guozhen Rong, and Jianxin Wang</i>	1:1–1:18
Lower Bound for Non-Adaptive Estimation of the Number of Defective Items	
<i>Nader H. Bshouty</i>	2:1–2:9
A Polynomial-Delay Algorithm for Enumerating Connectors Under Various Connectivity Conditions	
<i>Kazuya Haraguchi and Hiroshi Nagamochi</i>	3:1–3:15
Top Tree Compression of Tries	
<i>Philip Bille, Paweł Gawrychowski, Inge Li Gørtz, Gad M. Landau, and Oren Weimann</i>	4:1–4:18
Two Phase Transitions in Two-Way Bootstrap Percolation	
<i>Ahad N. Zehmakan</i>	5:1–5:21
Sliding Window Property Testing for Regular Languages	
<i>Moses Ganardi, Danny Hucker, Markus Lohrey, and Tatiana Starikovskaya</i>	6:1–6:13
On the Hardness of Set Disjointness and Set Intersection with Bounded Universe	
<i>Isaac Goldstein, Moshe Lewenstein, and Ely Porat</i>	7:1–7:22
Gathering and Election by Mobile Robots in a Continuous Cycle	
<i>Paola Flocchini, Ryan Killick, Evangelos Kranakis, Nicola Santoro, and Masafumi Yamashita</i>	8:1–8:19
Strategy-Proof Approximation Algorithms for the Stable Marriage Problem with Ties and Incomplete Lists	
<i>Koki Hamada, Shuichi Miyazaki, and Hiroki Yanagisawa</i>	9:1–9:14
Online Multidimensional Packing Problems in the Random-Order Model	
<i>David Naori and Danny Raz</i>	10:1–10:15
Approximate Euclidean Shortest Paths in Polygonal Domains	
<i>R. Inkulu and Sanjiv Kapoor</i>	11:1–11:17
Reachability in High Treewidth Graphs	
<i>Rahul Jain and Raghunath Tewari</i>	12:1–12:14
Approximate Pricing in Networks: How to Boost the Betweenness and Revenue of a Node	
<i>Ruben Brokkelkamp, Sven Polak, Guido Schäfer, and Yllka Velaj</i>	13:1–13:15
Slaying Hydrae: Improved Bounds for Generalized k-Server in Uniform Metrics	
<i>Marcin Bienkowski, Łukasz Jeż, and Paweł Schmidt</i>	14:1–14:14



Measure and Conquer for Max Hamming Distance XSAT <i>Gordon Hoi and Frank Stephan</i>	15:1–15:19
Cyclability in Graph Classes <i>Christophe Crespelle, Carl Feghali, and Petr A. Golovach</i>	16:1–16:13
Complexity of Linear Operators <i>Alexander S. Kulikov, Ivan Mikhailin, Andrey Mokhov, and Vladimir Podolskii</i> ...	17:1–17:12
New Results for the k -Secretary Problem <i>Susanne Albers and Leon Ladewig</i>	18:1–18:19
Triangle Estimation Using Tripartite Independent Set Queries <i>Anup Bhattacharya, Arijit Bishnu, Arijit Ghosh, and Gopinath Mishra</i>	19:1–19:17
Step-By-Step Community Detection in Volume-Regular Graphs <i>Luca Becchetti, Emilio Cruciani, Francesco Pasquale, and Sara Rizzo</i>	20:1–20:23
Blocking Dominating Sets for H -Free Graphs via Edge Contractions <i>Esther Galby, Paloma T. Lima, and Bernard Ries</i>	21:1–21:14
Internal Dictionary Matching <i>Panagiotis Charalampopoulos, Tomasz Kociumaka, Manal Mohamed, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń</i>	22:1–22:17
Approximating the Geometric Edit Distance <i>Kyle Fox and Xinyi Li</i>	23:1–23:16
On Adaptivity Gaps of Influence Maximization Under the Independent Cascade Model with Full-Adoption Feedback <i>Wei Chen and Binghui Peng</i>	24:1–24:19
Minimum-Width Double-Strip and Parallelogram Annulus <i>Sang Won Bae</i>	25:1–25:14
Small Candidate Set for Translational Pattern Search <i>Ziyun Huang, Qilong Feng, Jianxin Wang, and Jinhui Xu</i>	26:1–26:17
The Weighted k -Center Problem in Trees for Fixed k <i>Binay Bhattacharya, Sandip Das, and Subhadeep Ranjan Dev</i>	27:1–27:11
Online Knapsack Problems with a Resource Buffer <i>Xin Han, Yasushi Kawase, Kazuhisa Makino, and Haruki Yokomaku</i>	28:1–28:14
Local Cliques in ER-Perturbed Random Geometric Graphs <i>Matthew Kahle, Minghao Tian, and Yusu Wang</i>	29:1–29:22
Local Routing in Sparse and Lightweight Geometric Graphs <i>Vikrant Ashvinkumar, Joachim Gudmundsson, Christos Levcopoulos, Bengt J. Nilsson, and André van Renssen</i>	30:1–30:13
Searching for Cryptogenography Upper Bounds via Sum of Square Programming <i>Dominik Scheder, Shuyang Tang, and Jiaheng Zhang</i>	31:1–31:12
On the Complexity of Lattice Puzzles <i>Yasuaki Kobayashi, Koki Suetsugu, Hideki Tsuiki, and Ryuhei Uehara</i>	32:1–32:12

The I/O Complexity of Hybrid Algorithms for Square Matrix Multiplication <i>Lorenzo De Stefani</i>	33:1–33:16
Accurate MapReduce Algorithms for k -Median and k -Means in General Metric Spaces <i>Alessio Mazzetto, Andrea Pietracaprina, and Geppino Pucci</i>	34:1–34:16
On Optimal Balance in B-Trees: What Does It Cost to Stay in Perfect Shape? <i>Rolf Fagerberg, David Hammer, and Ulrich Meyer</i>	35:1–35:16
How Does Object Fatness Impact the Complexity of Packing in d Dimensions? <i>Sándor Kisfaludi-Bak, Dániel Marx, and Tom C. van der Zanden</i>	36:1–36:18
On One-Round Discrete Voronoi Games <i>Mark de Berg, Sándor Kisfaludi-Bak, and Mehran Mehr</i>	37:1–37:17
On Explicit Branching Programs for the Rectangular Determinant and Permanent Polynomials <i>V. Arvind, Abhranil Chatterjee, Rajit Datta, and Partha Mukhopadhyay</i>	38:1–38:13
A Competitive Algorithm for Random-Order Stochastic Virtual Circuit Routing <i>Kim Thăng Nguyễn</i>	39:1–39:12
An Improved Data Structure for Left-Right Maximal Generic Words Problem <i>Yuta Fujishige, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda</i>	40:1–40:12
Parameterized Complexity Classification of Deletion to List Matrix-Partition for Low-Order Matrices <i>Akanksha Agrawal, Sudeshna Kolay, Jayakrishnan Madathil, and Saket Saurabh</i> ..	41:1–41:14
The Generalized Microscopic Image Reconstruction Problem <i>Amotz Bar-Noy, Toni Böhnlein, Zvi Lotker, David Peleg, and Dror Rawitz</i>	42:1–42:15
Stabilization Time in Minority Processes <i>Pál András Papp and Roger Wattenhofer</i>	43:1–43:19
Parameterized Complexity of Stable Roommates with Ties and Incomplete Lists Through the Lens of Graph Parameters <i>Robert Bredereck, Klaus Heeger, Dušan Knop, and Rolf Niedermeier</i>	44:1–44:14
Path and Ancestor Queries over Trees with Multidimensional Weight Vectors <i>Meng He and Serikzhan Kazi</i>	45:1–45:17
A $\frac{21}{16}$ -Approximation for the Minimum 3-Path Partition Problem <i>Yong Chen, Randy Goebel, Bing Su, Weitian Tong, Yao Xu, and An Zhang</i>	46:1–46:20
Efficiently Realizing Interval Sequences <i>Amotz Bar-Noy, Keerti Choudhary, David Peleg, and Dror Rawitz</i>	47:1–47:15
Efficient Interactive Proofs for Linear Algebra <i>Graham Cormode and Chris Hickey</i>	48:1–48:19
When Maximum Stable Set Can Be Solved in FPT Time <i>Édouard Bonnet, Nicolas Bousquet, Stéphan Thomassé, and Rémi Watrigant</i>	49:1–49:22

The k -Fréchet Distance: How to Walk Your Dog While Teleporting <i>Hugo Alves Akitaya, Maike Buchin, Leonie Ryjkin, and Jérôme Urhausen</i>	50:1–50:15
New Applications of Nearest-Neighbor Chains: Euclidean TSP and Motorcycle Graphs <i>Nil Mamano, Alon Efrat, David Eppstein, Daniel Frishberg, Michael T. Goodrich, Stephen Kobourov, Pedro Matias, and Valentin Polishchuk</i>	51:1–51:21
Efficient Circuit Simulation in MapReduce <i>Fabian Frei and Koichi Wada</i>	52:1–52:21
Concurrent Distributed Serving with Mobile Servers <i>Abdolhamid Ghodselahi, Fabian Kuhn, and Volker Turau</i>	53:1–53:18
Tracking Paths in Planar Graphs <i>David Eppstein, Michael T. Goodrich, James A. Liu, and Pedro Matias</i>	54:1–54:17
Distance Measures for Embedded Graphs <i>Hugo A. Akitaya, Maike Buchin, Bernhard Kilgus, Stef Sijben, and Carola Wenk</i> .	55:1–55:15
Online Algorithms for Warehouse Management <i>Philip Dasler and David M. Mount</i>	56:1–56:21
On Approximate Range Mode and Range Selection <i>Hicham El-Zein, Meng He, J. Ian Munro, Yakov Nekrich, and Bryce Sandlund</i> ...	57:1–57:14
External Memory Planar Point Location with Fast Updates <i>John Iacono, Ben Karsin, and Grigorios Koumoutsos</i>	58:1–58:18
Minimizing and Computing the Inverse Geodesic Length on Trees <i>Serge Gaspers and Joshua Lau</i>	59:1–59:19
Result-Sensitive Binary Search with Noisy Information <i>Narthana S. Epa, Junhao Gan, and Anthony Wirth</i>	60:1–60:15
Improved Algorithms for Clustering with Outliers <i>Qilong Feng, Zhen Zhang, Ziyun Huang, Jinhui Xu, and Jianxin Wang</i>	61:1–61:12
Unbounded Regions of High-Order Voronoi Diagrams of Lines and Segments in Higher Dimensions <i>Gill Barequet, Evanthia Papadopoulou, and Martin Suderland</i>	62:1–62:15
Neighborhood Inclusions for Minimal Dominating Sets Enumeration: Linear and Polynomial Delay Algorithms in P_7 -Free and P_8 -Free Chordal Graphs <i>Oscar Defrain and Lhouari Nourine</i>	63:1–63:16
Dual-Mode Greedy Algorithms Can Save Energy <i>Barbara Geissmann, Stefano Leucci, Chih-Hung Liu, Paolo Penna, and Guido Proietti</i>	64:1–64:18

■ Preface

This volume contains the papers presented at ISAAC 2019: The 30th International Symposium on Algorithms and Computation, held during December 8-11, 2019 at Shanghai University of Finance and Economics, in Shanghai, P. R. China. The symposium provides a forum for researchers working in algorithms and theory of computation, and brings together experts at the research frontiers in these areas to exchange ideas and to present significant new results.

The program committee, consisting of 38 professional researchers from the field, reviewed 177 submissions and decided to accept 64 among many good papers. Each paper had 3 reviews, with additional reviews solicited as needed. The review process was conducted entirely electronically via EasyChair. We are grateful to EasyChair for allowing us to handle the submissions and the review process and to the program committee for their insightful reviews and discussions, which made our work more smoothly.

The committee selected the following two papers as the recipients of the ISAAC 2019 Best Paper Award and Best Student Paper respectively:

- **Best Paper.** Mark de Berg, Sándor Kisfaludi-Bak and Mehran Mehr: On One-Round Discrete Voronoi Games
- **Best Student Paper.** Ahad N. Zehmakan: Two Phase Transitions in Two-way Bootstrap Percolation

Besides the regular talks, the program also included three invited talks by Wei Chen (Microsoft Research Asia, China), Xi Chen (Columbia University, USA), and Leslie Ann Goldberg (University of Oxford, UK).

We are very grateful to all the people who made this meeting possible: The authors for submitting their papers, the program committee members and external reviewers for their excellent work, and the three invited speakers. In particular, we would like to thank the Institute for Theoretical Computer science (ITCS) at Shanghai University of Finance and Economics for hosting the conference and providing organizational supports.

Oct 10, 2019
Shanghai

Pinyan Lu and Guochuan Zhang



■ Symposium Organization

Program Chair

Pinyan Lu	Shanghai University of Finance and Economics
Guochuan Zhang	Zhejiang University

Program Committee

Georgios Barmpalias	Chinese Academy of Sciences
Xiaohui Bei	Nanyang Technological University
Andrej Bogdanov	The Chinese University of Hong Kong
Ho-Lin Chen	National Taiwan University
Lin Chen	Texas Tech University
Yijia Chen	Fudan University
Marek Chrobak	University of California, Riverside
Ran Duan	Tsinghua University
Leah Epstein	University of Haifa
Thomas Erlebach	University of Leicester
Zhiyi Huang	The University of Hong Kong
Klaus Jansen	University of Kiel
Bundit Laekhanukit	Shanghai University of Finance and Economics
Ron Lavi	Technion – Israel Institute of Technology
Shi Li	University at Buffalo
Guohui Lin	University of Alberta
Jiamou Liu	The University of Auckland
Kuldeep S. Meel	National University of Singapore
Nicole Megow	Universität Bremen
Benjamin Moseley	Carnegie Mellon University
Periklis Papakonstantinou	Rutgers University
Georgios Piliouras	Singapore University of Technology and Design
Venkatesh Raman	The Institute of Mathematical Sciences, Chennai
David Richerby	University of Oxford
Kunihiko Sadakane	The University of Tokyo
Frits Spijksma	Eindhoven University of Technology
Piyush Srivastava	TIFR
Xiaoming Sun	Institute of Computing Technology, Chinese Academy of Sciences
Zhihao Gavin Tang	Shanghai University of Finance and Economics
Marc Uetz	University of Twente
Mingji Xia	Institute of Software, Chinese Academy of Sciences
Yuichi Yoshida	National Institute of Informatics
Chihao Zhang	Shanghai Jiao Tong University
Jialin Zhang	Institute of Computing Technology, Chinese Academy of Sciences
Qin Zhang	Indiana University Bloomington
Stanislav Živný	University of Oxford

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang



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

Advisory Committee

Siu Wing Cheng	The Hong Kong University of Science and Technology
Ding-Zhu Du	The University of Dallas
Seok-Hee Hong	The University of Sydney
Tsan-Sheng Hsu	Academia Sinica
Chung-Shou Liao	National Tsinghua University
Kazuhisa Makino	Kyoto University
Yoshio Okamoto	The University of Electro-Communications
Kunsoo Park	Seoul National University
Takeshi Tokuyama	Tohoku University
Guochuan Zhang	Zhejiang University

Organizing Committee

Huili Liang	Shanghai University of Finance and Economics
Pinyan Lu	Shanghai University of Finance and Economics
Zhihao Gavin Tang	Shanghai University of Finance and Economics
Zihe Wang	Shanghai University of Finance and Economics

Additional Reviewers

A

Ahmed, Shareef
Akshay, S.
Alman, Josh
Antoniadis, Antonios
Asif, Hafiz
Aurenhammer, Franz

B

Bailey, James P.
Balazs, Peter
Banik, Aritra
Bansal, Suguman
Barrus, Michael
Baste, Julien
Basu, Riddhipratim
Becker, Ruben
Belmonte, Rémy
Berndt, Sebastian
Bhargava, Vishwas
Bodwin, Greg
Bringmann, Karl
Broersma, Hajo
Buchheim, Christoph
Buchin, Kevin

C

Cabello, Sergio
Cai, Zhipeng

Capelli, Florent

Chakraborty, Sankardeep

Chalopin, Jérémie

Chang, Yi-Jun

Chau, Vincent

Chen, Hsueh-Ping

Chen, Jiecao

Chen, Shiteng

Chen, Yong

Chen, Yu

Cheng, Siu-Wing

Choudhari, Jayesh

Coester, Christian

Cohen, Alon

Conte, Alessio

Courcelle, Bruno

Cseh, Ágnes

D

Das, Syamantak

Dawar, Anuj

Deppert, Max

Dinneen, Michael

Disser, Yann

Diwan, Ajit

Doerr, Benjamin

Dulio, Paolo

E

Eberle, Franziska
 Elbassioni, Khaled
 Eppstein, David

F

Fan, Chenglin
 Fernau, Henning
 Fleming, Noah
 Fomin, Fedor
 Fox, Kyle
 Francis, Mathew
 Fulla, Peter

G

Galanis, Andreas
 Gijswijt, Dion
 Goaoc, Xavier
 Golovach, Petr
 Grage, Kilian
 Guo, Xiangyu
 Gupta, Siddharth

H

Hatano, Daisuke
 Haviv, Ishay
 He, Kun
 He, Meng
 Hellmuth, Marc
 Hill, Darryl
 Hobbs, Nathaniel
 Huang, Jia
 Huang, Zengfeng
 Husfeldt, Thore
 Hübschle-Schneider, Lorenz

I

Igarashi, Ayumi
 Ikeda, Masahiro
 Iwama, Kazuo
 Iwamasa, Yuni
 Iwamoto, Chuzo

J

Jacob, Ashwin
 Jež, Artur
 Jiang, Shaofeng
 Jin, Kai
 Jin, Yaonan
 Johnson, Matthew

K

Kakimura, Naonori
 Karpov, Nikolai

Karthik C. S.,
 Kawase, Yasushi
 Kern, Walter
 Kiefer, Sandra
 Kim, Eunjung
 Klein, Kim-Manuel
 Kleist, Linda
 Kolay, Sudeshna
 Kowalczyk, Daniel
 Kratsch, Stefan
 Kumar, Gunjan
 Kuszmaul, William
 Köhler, Ekki
 Künnemann, Marvin

L

Lam, Chi-Kit
 Lapinskas, John
 Lassota, Alexandra
 Le Gall, Francois
 Lecroq, Thierry
 Leucci, Stefano
 Levi, Amit
 Lewis-Pye, Andrew
 Li, Qian
 Li, Shuai
 Li, Yi
 Liao, Chao
 Lin, Bingkai
 Lin, Guohui
 Lin, Jiabao
 Liu, Quanquan
 Liu, Shengxin
 Lokshtanov, Daniel
 Lu, Xinhang
 Luo, Kelin
 Luo, Wenchang

M

M. Sridharan, Ramanujan
 Maack, Marten
 Machado, Raphael
 Maehara, Takanori
 Majumdar, Diptapriyo
 Mande, Nikhil
 Manoussakis, George
 Manthey, Bodo
 Manurangsi, Pasin
 Mao, Yuchen
 Marcinkowski, Jan

Meeks, Kitty
Mei, Lili
Melissaris, Nikolas
Miltzow, Till
Misra, Neeldhara
Mizuki, Takaaki
Mnich, Matthias
Moses, William
Muller, Haiko
Myrisiotis, Dimitrios
N
Nakanishi, Masaki
Nakano, Shin-Ichi
Narayanaswamy, N.S.
Naves, Guylain
Nederlof, Jesper
Nehama, Ilan
Nekrich, Yakov
Neogi, Rian
Neuen, Daniel
Nicholson, Patrick K.
Nissim, Roy
Nutov, Zeev
O
Ordyniak, Sebastian
P
Pandey, Arti
Panolan, Fahad
Pavan, A
Pferschy, Ulrich
Q
Qiao, Youming
Qiu, Guoliang
R
Raghvendra, Sharath
Rau, Malin
Rawitz, Dror
Rohwedder, Lars
Romero Orth, Miguel
S
S, Krishna
Sahlot, Vibha
Salavatipour, Mohammad
Satti, Srinivasa Rao
Saurabh, Saket
Scarlett, Jonathan
Schewior, Kevin
Schlöter, Miriam

Schmid, Andreas
Schwiegelshohn, Chris
Scquizzato, Michele
Shan, Xiaohan
Sharma, Vikram
Sherif, Suhail
Shi, Yangguang
Shih, Cheng-Yu
Shimizu, Nobutaka
Shrotri, Aditya A.
Simon, Bertrand
Sinha, Makrand
Sintos, Stavros
Skopalik, Alexander
Solis-Oba, Roberto
Souza, Uéverton
Stephan, Frank
Strozecki, Yann
Suksompong, Warut
Sun, Yuan
T
Takaoka, Asahi
Tan, Zihan
Tao, Biaoshuai
Tao, Chao
Telikepalli, Kavitha
Thaler, Justin
Tian, Guojing
Tokuyama, Takeshi
U
Uniyal, Sumedha
V
van Stee, Rob
Variyam, Vinod
Vaz, Daniel
Verdugo, Victor
Vigneron, Antoine
Végh, László
W
Wahlström, Magnus
Wang, Fu-Hsing
Wang, Hao
Wang, Kangning
Wang, Xiao
Ward, Justin
Wasa, Kunihiro
Wiese, Andreas
Wrochna, Marcin

Wu, Bujiao

Wu, Xiaowei

X

Xia, Zhiyu

Xiao, Mingyu

Xu, Chenyang

Y

Ye, Deshi

Ye, Junjie

You, Jie

Yu, Huacheng

Yu, Yu

Yuan, Pei

Z

Zeman, Peter

Zhang, Haoyu

Zhang, Jia

Zhang, Jingru

Zhang, Peng

Zhang, Tianyi

Zhang, Yuhao

Zhang, Zhijie

Zhao, Dengji

Zhou, Samson

Zhu, Shenglong

Zhu, Xue

Graph Searches and Their End Vertices

Yixin Cao 

Department of Computing, Hong Kong Polytechnic University, Hong Kong, China
yixin.cao@polyu.edu.hk

Zhifeng Wang

School of Computer Science and Engineering, Central South University, Changsha, China

Guozhen Rong

School of Computer Science and Engineering, Central South University, Changsha, China

Jianxin Wang

School of Computer Science and Engineering, Central South University, Changsha, China

Abstract

Graph search, the process of visiting vertices in a graph in a specific order, has demonstrated magical powers in many important algorithms. But a systematic study was only initiated by Corneil et al. a decade ago, and only by then we started to realize how little we understand it. Even the apparently naïve question “which vertex can be the last visited by a graph search algorithm,” known as the end vertex problem, turns out to be quite elusive. We give a full picture of all maximum cardinality searches on chordal graphs, which implies a polynomial-time algorithm for the end vertex problem of maximum cardinality search. It is complemented by a proof of NP-completeness of the same problem on weakly chordal graphs. We also show linear-time algorithms for deciding end vertices of breadth-first searches on interval graphs, and end vertices of lexicographic depth-first searches on chordal graphs. Finally, we present $2^n \cdot n^{O(1)}$ -time algorithms for deciding the end vertices of breadth-first searches, depth-first searches, and maximum cardinality searches on general graphs.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases maximum cardinality search, (lexicographic) breadth-first search, (lexicographic) depth-first search, chordal graph, weighted clique graph, end vertex

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.1

Related Version A full version of the paper is available at <https://arxiv.org/abs/1905.09505>.

Funding Supported by the RGC grants 15201317 and 15226116, and the NSFC grant 61972330.

1 Introduction

Breadth-first search (BFS) and depth-first search (DFS) are the most fundamental graph algorithms, and the standard opening of a course on this subject. Their use can be found, sometimes implicitly, in most graph algorithms. In general, a graph search is a systematic exploration of a graph, and its core lies on the strategy of how to choose the next vertex to visit. Mostly greedy, graph searches are very simple but sometimes have magical powers. DFS has played a significant role in Tarjan’s award-winning work, in testing planarity [19] and in finding strongly connected components [25].

Two other search algorithms, lexicographic breadth-first search (LBFS) [21] and maximum cardinality search (MCS) [26], were invented for the purpose of recognizing chordal graphs, i.e., graphs not containing any induced cycle on four or more vertices. On a chordal graph, both LBFS and MCS produce perfect elimination orderings (see definition in the next section) of the graph, which exist if and only if the graph is chordal. Albeit relatively less well known compared to BFS and DFS, LBFS and MCS did find important applications. LBFS is used in scheduling [22], and is the base of the recent linear-time algorithm for computing modular decomposition of a graph [27]. MCS is used in testing acyclic hypergraphs and in computing minimum cuts of a graph and find forest decompositions.



© Yixin Cao, Zhifeng Wang, Guozhen Rong, and Jianxin Wang;
licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 1; pp. 1:1–1:18

Leibniz International Proceedings in Informatics



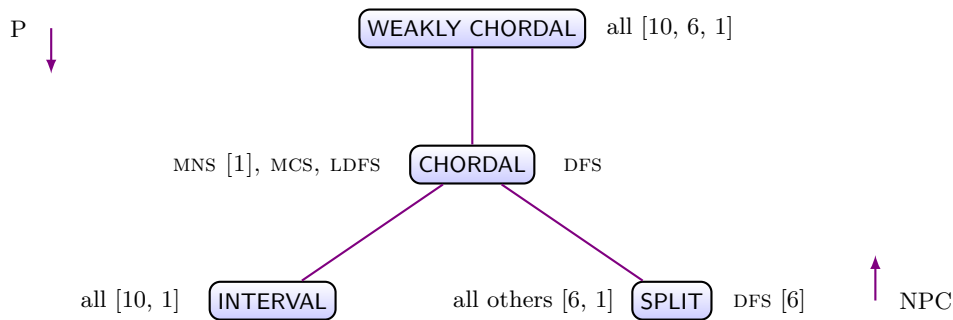
LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Simon [24] proposed an interesting way of using LBFS. It conducts LBFS more than once, and each new run uses previous runs in breaking ties; in particular, except the first, each run starts from the last vertex of the previous run. This generic approach turns out to be very useful, e.g., the extremely simple recognition algorithm for unit interval graphs [7]. See the survey of Corneil [8] for more algorithms using multiple runs of LBFS. Some of these results have a flavor of “ad-hoc”: We do not fully understand the execution process of LBFS.

The outputs of BFS and DFS are usually rooted spanning forests of the graph, while LBFS and MCS produce orderings of its vertices. To have a unified view of them, Corneil et al. [11] focused on the ordering of the vertices being first visited and conducted a systematic study of them. This study motivates them to propose the lexicographic version of DFS, lexicographic depth-first search (LDFS), another very powerful graph search [9], and a very general search paradigm, maximum neighborhood search (MNS). They showed that all the aforementioned graph searches can be characterized by variants of the so-called four-vertex condition. These nice characterizations are however not sufficient to allow us to answer the ostentatiously naïve question: Which vertex can be the last of such an ordering? Corneil et al. [10] defined the end vertex problem and studied it from both combinatorial and algorithmic perspectives. Apart from a natural starting point of understanding the graph searches in general, end vertices of graph searches are of their own interest. Behind the original use of LBFS and MCS, in the recognition of chordal graphs, is nothing but the properties of their end vertices, which are always simplicial on a chordal graph [21, 26, 23, 4]. Moreover, the success of multiple-run LBFS crucially hinges on the end vertices; e.g., an end vertex of a (unit) interval graph can always be assigned an extreme (i.e., leftmost or rightmost) interval [7, 13]. Important properties and use of end vertices of other graph searches can be found in [9, 17, 12].

One may find it surprising, but the end vertex problem is NP-hard for all the six mentioned graph search algorithms [11, 6, 1]. The study has thus been focused on chordal graphs and its closely related superclasses and subclasses. After all, LBFS and MCS were invented for recognition of chordal graphs, and their properties on chordal graphs have been intensively studied. (This renders the stagnation on chordal graphs a little more embarrassing.) Moreover, most applications of LBFS and LDFS are on related graph classes. The most natural superclass of chordal graphs is arguably the weakly chordal graphs, and two important subclasses are interval graphs and split graphs. It has been known that on weakly chordal graphs, the end vertex problems for all but MCS are NP-complete, while only DFS end vertex is NP-complete on chordal graphs [11, 6, 1]. There are other polynomial-time algorithms for interval graphs and split graphs, most of which actually run in linear time. We complete the pictures for, in terms of graph searches, MCS and LDFS, and, in terms of graph classes, weakly chordal graphs and interval graphs. A summary of known results is given in Fig. 1.

Blair and Peyton [5] and Galinier et al. [16] have shown that MCS of a chordal graph are closely related to its maximal cliques. Let G be a chordal graph. An MCS visits all vertices in a maximal clique of G before proceeding to another, and the next maximal clique is always chosen to have the largest intersection with a visited one. Therefore, for a minimum separator S of G , there is an MCS visiting the components of $G - S$ one by one, with S visited together with the first component. If we turn to any component C of $G - S$, and consider its closed neighborhood, (which contains C and S), then we have a similar statement. In other words, this property on minimum separators holds in a recursive way. For an MCS end vertex z , which is necessarily simplicial, we can find a sequence of increasing separators such that the first is a minimum separator of G and the last comprises all the non-simplicial vertices in $N(z)$. An MCS ended with z has to “cross” these separators in order, and for each of them, visit the component containing z in the last. We have thus a full understanding of all MCS orderings of a chordal graph. As it turns out, this result is easier to be presented in the



■ **Figure 1** A summary of the known complexity of the end vertex problem for the six graph search algorithms. For each graph class, the end vertex problem of graph searches listed to the left of it can be solved in polynomial time on this class, while those to the right are NP-hard. The complexity of the BFS end vertex and LBFS end vertex problems on chordal graphs are still open.

so-called weighted clique graph of G [5, 16]. It enables us to show that if we run MCS twice, first starting from z , and the second starting from the end vertex of the first run and using the first ordering to break ties, then the second run ends with z if and only if z is an MCS end vertex. As usual, n denotes the number of vertices in the input graph.

► **Theorem 1.** *The MCS end vertex problem can be solved in $O(n^2)$ time on chordal graphs.*

We complement this result by showing that the MCS end vertex problem becomes NP-complete on weakly chordal graphs; the proof is inspired by and adapted from [1].

► **Theorem 2.** *The MCS end vertex problem is NP-complete on weakly chordal graphs.*

We then turn to LDFS on chordal graphs. Surprisingly, the characterization of Berry et al. [3] for end vertices of MNS on chordal graphs is also true for LDFS: A simplicial vertex z of a chordal graph G is an LDFS end vertex if and only if the minimal separators of G in $N(z)$ are totally ordered by inclusion. We also show a simple algorithm for solving the BFS end vertex problem on interval graphs.

► **Theorem 3.** *There are linear-time algorithms for solving the LDFS end vertex problem on chordal graphs and for solving the BFS end vertex problem on interval graphs.*

We have to, nevertheless, leave open the BFS and LBFS end vertex problems on chordal graphs. Since both can be solved in linear time on split graphs, we conjecture that they can be solved in polynomial time on chordal graphs. It is extremely rare that a problem is hard on chordal graphs but easy on split graphs.

We also consider algorithms for solving the end vertex problems on general graphs. By enumerating all possible orderings, a trivial algorithm can find all end vertices of any graph search in $n! \cdot n^{O(1)}$ time. On the other hand, with the only exception of BFS, the reductions used in proving NP-hardness of the end vertex problems are linear reductions from (3-)SAT. As a result, these problems cannot be solved in subexponential time, unless the exponential time hypothesis fails [20]. A natural question is thus which of them can be solved in $2^{O(n)}$ time. If we put them under closer scrutiny, we will see that these graph searches are somewhat different: When selecting the next vertex, MCS only needs to know which vertices have been visited, while the order of visiting them is immaterial. In contrast, the other graph searches are not *oblivious* and need to keep track of the whole visiting history. It is straightforward to use dynamic programming to solve the MCS end vertex problem in $2^n \cdot n^2$ time. Interestingly, a similar approach actually works for the BFS and DFS end vertex problems.

► **Theorem 4.** *There are $2^n \cdot n^{O(1)}$ -time algorithms that solve the end vertex problems of the following graph searches: MCS, BFS, and DFS.*

2 Preliminaries

All graphs discussed in this paper are undirected and simple. The vertex set and edge set of a graph G are denoted by, respectively, $V(G)$ and $E(G)$, and we use $n = |V(G)|$ and $m = |E(G)|$ to denote their cardinalities. For a subset $X \subseteq V(G)$, denote by $G[X]$ the subgraph of G induced by X , and by $G - X$ the subgraph $G[V(G) \setminus X]$. The *degree* of a vertex v is the number of neighbors it has, i.e., $d(v) = |N(v)|$. A vertex v is *simplicial* if $N[v]$ induces a complete graph. Two distinct vertices u and v are *true twins* if $N[u] = N[v]$, and *false twins* if $N(u) = N(v)$; note that true twins are adjacent while false twins are not.

A vertex set S is a *u - v separator* if u and v are not in S and they are not connected in $G - S$, and a *u - v separator* is *minimal* if no proper subset of S is a *u - v separator*. We call S a (minimal) separator if it is a (minimal) *u - v separator* for some pair of u and v , and it is a *minimum separator* of G if it has the smallest cardinality among all separators of G .

An *ordering* σ of the vertices of G is a bijection from $V(G) \rightarrow \{1, \dots, n\}$. For two vertices u and v , we use $u <_\sigma v$ to denote $\sigma(u) < \sigma(v)$. The *end vertex* of σ is the vertex z with $\sigma(z) = n$. Given a graph G and a vertex $z \in V(G)$, the *end vertex problem* for graph search S is to determine whether there is an S -ordering of G of which z is the end vertex.

A graph is *chordal* if it contains no induced cycle on four or more vertices. A graph is chordal if and only if it can be made empty by removing simplicial vertices from the remaining graph one by one; the order of the vertices removed is called a *perfect elimination ordering* [15]. The greedy strategy of MCS is to choose an unvisited vertex with the maximum number of visited neighbors. On a chordal graph G , the last vertex of any MCS is simplicial, and thus the reversal of an MCS ordering is always a perfect elimination ordering [26].

To avoid unnecessary digressions, we consider only connected graphs.

3 Maximum cardinality search on chordal graphs

Another important characterization of chordal graphs is through its maximal cliques. A graph G is chordal if and only if we can arrange its maximal cliques as a tree such that for each vertex $v \in V(G)$, maximal cliques containing v induce a subtree; such a tree is called a *clique tree* of G [14]. A chordal graph G has at most n maximal cliques [14], and for any pair of adjacent K_i and K_j on the clique tree, intersection $K_i \cap K_j$ is a minimal separator of G .

Out of a chordal graph G , we can define a *weighted clique graph* $C(G)$ as follows. It has ℓ vertices, where ℓ is the number of maximal cliques of G , and each vertex is labeled by a distinct maximal clique of G . To simplify the presentation, we will refer to vertices of $C(G)$ as cliques; note that we are not going to use cliques of the graph $C(G)$ in this paper. There is an edge between maximal cliques K_i and K_j , $1 \leq i, j \leq \ell$, if and only if $K_i \cap K_j$ is a minimal x - y separator for all $x \in K_i \setminus K_j$ and $y \in K_j \setminus K_i$. We label this edge with $K_i \cap K_j$, and set its weight to be $|K_i \cap K_j|$. It is known that a tree on the maximal cliques of G is a clique tree of G if and only if it is a maximum spanning tree of $C(G)$ [2, 5, 16], i.e., a spanning tree of $C(G)$ with the maximum total edge weights.

► **Proposition 5.** *Let G be a chordal graph and $C(G)$ the weighted clique graph of G . A set $S \subseteq V(G)$ is a minimal separator of G if and only if it is the label for some edge of $C(G)$.*

One can use Prim's algorithm to find a maximum spanning tree of G . (Although proposed for finding a minimum spanning tree, Prim's algorithm can be easily modified to find a maximum one.) Starting from an arbitrary clique, it grows the tree by including one edge and one clique at a time, while the edge is chosen to have the largest weight among those

crossing the partial tree that has been built, i.e., with one end in the current tree and the other not. In the same spirit of graph search orderings, we can define a *Prim ordering* to be the order maximal cliques of G being included by Prim's algorithm, applied to $C(G)$.

Let π be an ordering of the maximal cliques of G . We say that an ordering σ of $V(G)$ is *generated by* π if $K_u <_\pi K_v$ implies $u <_\sigma v$, where K_u and K_v are the first maximal cliques in π containing u , and respectively, v . If $\pi = \langle K_1, K_2, \dots, K_\ell \rangle$ and $c_i = |K_i \setminus \bigcup_{j=1}^{i-1} K_j|$ for $1 \leq i \leq \ell$, then σ can be represented as

$$\underbrace{\sigma^{-1}(1), \dots, \sigma^{-1}(c_1)}_{K_1}, \underbrace{\sigma^{-1}(c_1 + 1), \dots, \sigma^{-1}(c_1 + c_2)}_{K_2 \setminus K_1}, \dots, \underbrace{\sigma^{-1}(n - c_\ell + 1), \dots, \sigma^{-1}(n)}_{K_\ell \setminus \bigcup_{j=1}^{\ell-1} K_j}.$$

The following has been essentially observed by Blair and Peyton [5], who however only stated explicitly one direction. For the sake of completeness, we give a proof here.

► **Lemma 6.** *Let G be a chordal graph. An ordering σ of $V(G)$ is an MCS ordering of G if and only if it is generated by some Prim ordering π of $C(G)$.*

Proof. The only if direction has been proved by Blair and Peyton [5, Lemma 4.8 and Theorem 4.10]. Here we show the if direction. Suppose that σ is generated by π . We may renumber the vertices in G such that $\sigma = \langle v_1, v_2, \dots, v_n \rangle$, and renumber the maximal cliques such that $\pi = \langle K_1, K_2, \dots, K_\ell \rangle$. Let $K'_i = K_i \setminus \bigcup_{j=1}^{i-1} K_j$ for $1 \leq i \leq \ell$; note that $\{K'_1, K'_2, \dots, K'_\ell\}$ is a partition of $V(G)$. We show by induction that for each $1 \leq i \leq n$, there is an MCS ordering of G of which the first i vertices are v_1, \dots, v_i ; in other words, among vertices v_i, \dots, v_n , vertex v_i has the maximum number of neighbors in the first $i - 1$ vertices. It is vacuously true for $i = 1$. Now suppose that it is true for v_p , we show that it is also true for v_{p+1} .

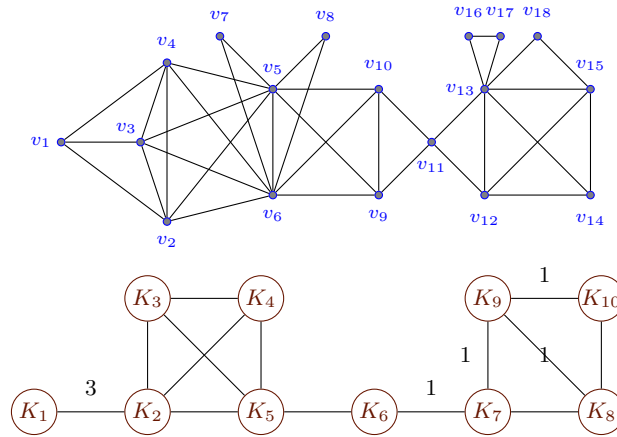
When $v_{p+1} \in K'_1 = K_1$, it is adjacent to all previous vertices and we are done. In the rest $v_{p+1} \in K'_t$ for some $t > 1$. Let $A = \bigcup_{j=1}^{t-1} K_j$; note that $v_{p+1} \notin A$. For any $q > p$, let G_q denote the subgraph of G induced by v_1, v_2, \dots, v_p , and v_q . By the induction hypothesis, $\langle v_1, v_2, \dots, v_p, v_q \rangle$ is an MCS ordering of G_q . Since G_q is chordal, v_q is simplicial in it. Therefore, $N(v_q) \cap A$ is a clique for all $q > p$; denote it by X_q . We argue by contradiction that there must be $1 \leq s < t$ such that $X_q \subseteq K_s$. We find an i with $1 \leq i < t$ such that $K_i \cap X_q$ is maximal. If $X_q \not\subseteq K_i$, then there is a vertex $x \in X_q \setminus K_i$; let K_j , where $1 \leq j < t$, contain x . By the maximality of $K_i \cap X_q$, there exists $y \in (X_q \cap K_i) \setminus K_j$. Of the first $t - 1$ maximal cliques, those containing $K_i \cap X_q$ and those containing $X_q \setminus K_i$ are disjoint. Prim's algorithm always maintains a tree of visited cliques, and this tree is a subtree of a clique tree of G . Therefore, there is an x - y separator. But this is impossible because x and y are both in X_q , hence adjacent.

For each $q > p$, there is some maximal clique K of G that contains $(N(v_q) \cap A) \cup \{v_q\}$. It cannot be one of K_1, \dots, K_{t-1} because $v_q \notin A$. Since K_1, \dots, K_ℓ is a Prim ordering of $C(G)$, we have $|N(v_{p+1}) \cap A| \geq |N(v_q) \cap A|$ for all $q > p$. On the other hand, v_{p+1} is adjacent to all vertices in K_t . We can thus conclude that v_{p+1} has the maximum number of neighbors in $\{v_1, \dots, v_p\}$, and this completes the proof. ◀

By Lemma 6, MCS orderings of a chordal graph G can be fully characterized by Prim orderings of its weighted clique graph $C(G)$. In particular, the MCS end vertices are the private vertices of the cliques last visited by Prim's algorithm. Note that a vertex v is simplicial if and only if it belongs to precisely one maximal clique, namely, $N[v]$, and a set of true twins can be visited in any order.

► **Corollary 7.** *Let z be a simplicial vertex in a chordal graph G . There is an MCS ordering of G ended with z if and only if there exists a Prim ordering of $C(G)$ ended with $N[z]$.*

Let S be a separator of G . We abuse notation to use $C(G) - S$ to denote the subgraph of $C(G)$ obtained by deleting all edges whose labels are subsets of S . The component of $C(G) - S$ containing $N[z]$ is called the z -component of $C(G) - S$. It is worth noting that $C(G) - S$ cannot be mapped back to G . In Fig. 2, for example, $C(G) - \{v_5, v_6\}$ does not have edges among K_2, \dots, K_5 , while edges $K_7K_8, K_7K_9, K_8K_9, K_9K_{10}$ will be removed in $C(G) - \{v_{12}, v_{13}\}$.



■ **Figure 2** On the top is a chordal graph G on 18 vertices, and below the weighted clique graph of G , where all the omitted edge weights are 2. There are 10 maximal cliques $K_1 = \{v_1, v_2, v_3, v_4\}$, $K_2 = \{v_2, \dots, v_6\}$, $K_3 = \{v_5, v_6, v_7\}$, $K_4 = \{v_5, v_6, v_8\}$, $K_5 = \{v_5, v_6, v_9, v_{10}\}$, $K_6 = \{v_9, v_{10}, v_{11}\}$, $K_7 = \{v_{11}, v_{12}, v_{13}\}$, $K_8 = \{v_{12}, \dots, v_{15}\}$, $K_9 = \{v_{13}, v_{16}, v_{17}\}$, $K_{10} = \{v_{13}, v_{15}, v_{18}\}$. There are 7 simplicial vertices $v_1, v_7, v_8, v_{14}, v_{16}, v_{17}, v_{18}$, of which v_{14} and v_{18} are not MCS end vertices.

► **Proposition 8.** *Let S be a separator of a chordal graph G . For any vertex $v \notin S$, maximal cliques containing v remain connected in $C(G) - S$. For any two distinct vertices $u, v \notin S$, maximal cliques containing u and v are not connected in $C(G) - S$ iff S is a u - v separator.*

Proof. By definition, the maximal cliques containing v are connected in any clique tree of G . Since a clique tree of G is a subgraph of $C(G)$, these cliques also induce a connected subgraph in $C(G)$. For any edge in this subgraph, its label contains v , hence not a subset of S . Therefore, these cliques induce the same connected subgraph in $C(G) - S$ as in $C(G)$.

For the second assertion, we may assume $uv \notin E(G)$: Both sides are trivially false when $uv \in E(G)$. Suppose to the contradiction of the if direction that there is a path K_0, \dots, K_p in $C(G) - S$ such that $u \in K_0$ and $v \in K_p$ while $u, v \notin K_i$ for $0 < i < p$. For each $1 \leq i \leq p$, we can find a vertex $x_i \in (K_{i-1} \cap K_i) \setminus S$. (These p vertices may or may not be distinct.) Then $ux_1, x_p v \in E(G)$, while x_i and x_{i+1} are either the same or adjacent for all $1 \leq i < p$. We have thus a u - v path in G avoiding S , contradiction that S is a u - v separator.

We now consider the only if direction. Let $u = x_0, x_1, \dots, x_p = v$ be any u - v path in G . Note that for each $0 \leq i \leq p$, maximal cliques containing x_i induce a connected subgraph, while for each $1 \leq j \leq p$, there is a maximal clique containing both x_{j-1} and x_j . We can find a path in $C(G)$ of which one end contains u and the other contains v . For each edge on this path, its label contains one of $x_i, 0 < i < p$. Since maximal cliques containing u and v are not connected in $C(G) - S$, the label of at least one edge on this path is a subset of S . By the first assertion, at least one of x_1, \dots, x_{p-1} is in S . In other words, every u - v path intersects S . Therefore, S is a u - v separator. This concludes the proof. ◀

We say that a minimum-weight edge e of $C(G)$ – by Proposition 5, its label is a minimum separator of G , – is a *critical edge* for maximal clique K if one end of e is in the same component as K after all minimum-weight edges, including e , are removed from $C(G)$. In other words, there is a path connecting K and e on which every edge has weight larger than e . In Fig. 2, e.g., K_6K_7 is a critical edge for all cliques but K_9 , while K_8K_9 and $K_{10}K_9$ are critical edges for K_8 and K_{10} respectively. The following fact explains “critical” in the name.

► **Proposition 9.** *Let z be a simplicial vertex of a connected chordal graph G , and let S_1, \dots, S_k be the labels of all critical edges for $N[z]$. In any Prim ordering of $C(G)$, cliques in the z -component of $C(G) - S_1 - \dots - S_k$ appear consecutively. Moreover, if $S_1 = \dots = S_k$, then the z -component of $C(G) - S_1$ can be visited in the end.*

Proof. Note that $C(G)$ is connected since G is connected. Let T denote the z -component of $C(G) - S_1 - \dots - S_k$. Being minimum separators of G , all of S_1, \dots, S_k have the same size; let it be t . Note that the weight of every edge in T is strictly larger than t ; otherwise, we can find a path from $N[z]$ to such an edge in T , and identify another critical edge for $N[z]$ on this path.

Let π be any Prim ordering of $C(G)$. We consider the first maximal clique K in T visited by π . If $\pi(K) \neq 1$, the edge leading to K has weight t . By Prim’s algorithm, when K is visited, for each clique K' with $K' <_\pi K$, all the edges between K' and its unvisited neighbors have weight t . All edges between T and other components have weight t as well, while all edges inside T have weight $> t$. Therefore, the maximal cliques in T must be finished before a clique out of T is visited. This concludes the first assertion.

For the second assertion, suppose that $S = S_1 = \dots = S_k$. We give a Prim ordering that visits cliques in T in the end. It starts from a clique not in T , and it suffices to show that all cliques out of T have been visited before the first in T . By the definition of $C(G)$, in each component of $C(G) - S$, there is a maximal clique containing S . Therefore, by Proposition 8, there is an edge with label S between any two components of $C(G) - S$. In other words, the cliques not in T are connected in $C(G)$. Since the edges connecting T and other components of $C(G) - S$ have weight t , the minimum in $C(G)$, Prim’s algorithm can always choose another edge. Therefore, we can finish them before entering T . ◀

Whether a simplicial vertex z can be an MCS end vertex turns out to be closely related to the critical edges for $N[z]$. We first present a necessary condition, which is not satisfied by v_{14} and v_{18} in Fig. 2; we leave it to the reader to verify that they cannot be MCS end vertices.

► **Lemma 10.** *Let z be a simplicial vertex of a connected chordal graph G . If $N[z]$ is the end clique of a Prim ordering of $C(G)$, then all critical edges for $N[z]$ have the same label.*

Proof. Suppose for contradiction that there are two critical edges e_1 and e_2 for $N[z]$ with different labels. For $i = 1, 2$, let S_i be the label of e_i , and let \mathcal{C}_i denote the set of components of $C(G) - S_i$ not containing $N[z]$. We argue that for any $U_1 \in \mathcal{C}_1$ and $U_2 \in \mathcal{C}_2$, they are different and there is no edge between them.

For $i = 1, 2$, by the definition of critical edges, there is a path from $N[z]$ to e_i ; let K_i denote the end of e_i that is closer to $N[z]$ on this path. There must be some clique K'_i in U_i containing S_i . Note that $K_i \cap K'_i = S_i$ because K'_i and K_i are in different components of $C(G) - S_i$. Hence, $K_iK'_i$ is also a critical edge with label S_i for $N[z]$. There is a $N[z]$ - K'_2 path in $C(G) - S_1$, and hence K'_2 and $N[z]$ are connected in $C(G) - S_1$. Likewise, K'_1 and $N[z]$ are connected in $C(G) - S_2$.

Since $S_1 \neq S_2$ and they have the same cardinality, we can find $v_2 \in S_2 \setminus S_1 \subset K'_2$. By Proposition 8, S_1 is not a z - v_2 separator. Thus, no maximal clique in U_1 contains v_2 . It follows that U_1 remains connected in $C(G) - S_2$ (note that S_2 is a minimum separator). For

the same reason, U_2 remains connected in $C(G) - S_1$. If there exists an edge between U_1 and U_2 , then this edge remains in at least one of $C(G) - S_1$ and $C(G) - S_2$: It cannot have both labels S_1 and S_2 . But then U_1 and U_2 are connected in $C(G) - S_1$ or $C(G) - S_2$, neither of which is possible. We can thus conclude that components in $\mathcal{C}_1 \cup \mathcal{C}_2$ are disjoint and there is no edge among them.

Let π be a Prim ordering of $C(G)$ ended with $N[z]$. Assume without loss of generality that the first visited clique in these components is from $U_1 \in \mathcal{C}_1$, then we show that $N[z]$ is visited before all components $U_2 \in \mathcal{C}_2$. Since there is no edge between U_1 and U_2 , before visiting U_2 , it must visit a clique from the z -component of $C(G) - S_1$. After that, however, it will not visit any edge of label S_2 before finishing this component. Therefore $N[z]$ cannot be the end clique, a contradiction. This concludes the proof. ◀

In other words, if z is an MCS end vertex, then there is a unique minimum separator of G that is “closest to z ” in a sense. This, although not sufficient, can be extended to a sufficient condition for MCS end vertices as follows. To decide whether a simplicial vertex z is an MCS end vertex, we can find the minimum separator S in Proposition 9 and focus on how the z -component of $C(G) - S$ is explored. We have to start from a maximal clique not in it, and after that visit all maximal cliques in other components of $C(G) - S$ before the z -component. In this juncture we may view the z -component as a separate graph and find all critical edges for $N[z]$ with respect to this component. They also need to have the same label; suppose it is S' , which is strictly larger than S . But this is not sufficient because we need to make sure that when S is crossed, it can reach a maximal clique not in the z -component of $C(G) - S'$. In Fig. 2, if we delete vertices v_{16} and v_{17} , (hence K_9), then K_6K_7 is the only critical edge for K_8 . The condition of Lemma 10 is satisfied, but v_{14} is still not an MCS end vertex.

Repeating this step recursively, we should obtain a sequence of separators with increasing cardinalities. Note that we only need to keep track of how these separators are crossed, while the ordering in each layer is irrelevant. This observation leads us to the following characterization, which subsumes Theorem 13 of Beisegel et al. [1]. For example, the sequence of critical edges for $N[v_1]$ in Fig. 2 are K_6K_7 , K_2K_5 , and K_1K_2 , which correspond to minimal separators $\{v_{11}\}$, $\{v_5, v_6\}$, and $\{v_2, v_3, v_4\}$, respectively.

► **Theorem 11.** *Let z be a simplicial vertex of a connected chordal graph G . The clique $N[z]$ is a Prim end clique if and only if there is a sequence of edges e_1, e_2, \dots, e_k in $C(G)$, where the label of e_i is S_i , on a path ended with $N[z]$ such that*

- (i) S_1 is the label of critical edges for $N[z]$ and S_k is the set of non-simplicial vertices in $N[z]$; and
- (ii) for $1 \leq i < k$, in the z -component of $C(G) - S_i$, all the critical edges for $N[z]$ have the same label, which is S_{i+1} .

Moreover, every clique not in the z -component of $C(G) - S_1$ can be the start clique.

Proof. We first show the if direction. We may denote the two ends of e_i by K_i and K'_i , where K'_i is in the z -component of $C(G) - S_i$. (It is possible that $K'_i = K_{i+1}$ for some $1 \leq i < k$.) For each $1 \leq i \leq k$, we visit all the other components of $C(G) - S_i$ before using the edge $K_iK'_i$ to enter the z -component, visiting K'_i . This is possible because of Proposition 9, and as such we produce a Prim ordering of $C(G)$ that ends with $N[z]$.

Now consider the only if direction, for which we construct the stated path by induction: We find the edges e_1, e_2, \dots, e_k in order, and show that for each $1 \leq i \leq k$, the first i edges can be extended to a path that ends with $N[z]$ and satisfies both conditions. The first edge e_1 can be any critical edge for $N[z]$, and it is on a path ended with $N[z]$ because $C(G)$ is connected. Now suppose that the first i edges, namely, e_1, \dots, e_i , have been selected, and we find e_{i+1} as follows. For each $1 \leq j \leq i$, let T_j denote the z -component of $T_{j-1} - S_j$, where $T_0 = C(G)$. If T_i comprises the only maximal clique $N[z]$, we are done.

Containing $N[z]$, cliques in T_i are last visited by Proposition 9. It is also a Prim ordering of the component itself. Therefore, Lemma 10 applies, and all the critical edges for $N[z]$ in T_i have the same label. Let S_{i+1} be this label, and let T_{i+1} be the z -component of $T_i - S_{i+1}$. We argue that there must be a maximal clique K in $T_i - T_{i+1}$ containing S_i ; otherwise, the first component visited in $T_i - S_{i+1}$ would be the z -component, and then $N[z]$ cannot be the last visited clique. We can use edge $K_i K$ to replace e_i , – note that they have the same label, – and choose any edge between K and $N[z]$ with label S_{i+1} as e_{i+1} . This concludes the inductive step and the proof. ◀

The proof of the only if direction of Theorem 11 can be directly translated into an algorithm to decide Prim end cliques, implying a polynomial-time algorithm for the MCS end vertex problem on chordal graphs. This algorithm however has to take $\Omega(n^2)$ time because the size of $C(G)$. We show a very simple algorithm below, which itself best reveals the spirit of graph searches. As long as we cross the separators in the order specified in Theorem 11, and make sure we finish other components before visiting the z -component, then it is the Prim ordering we need. On the other hand, a run of Prim’s algorithm started from $N[z]$ will cross the separators in the reversed order, and before crossing the i th separator S_i , it has to exhaust the whole z -component $C(G) - S_i$.

■ **Algorithm 1** Algorithm for deciding whether a vertex z is an MCS end vertex of a chordal graph.

INPUT: A graph G and an MCS ordering σ of G started with z .

OUTPUT: Whether z can be an MCS end vertex of G .

1. **for** $i \leftarrow 1$ to n **do**
 - 1.1. $D \leftarrow$ the set of unvisited vertices with the maximum number of visited neighbors;
 - 1.2. visit the vertex $\arg \max_{v \in D} \sigma(v)$;
2. **if** the last visited vertex is z **then return** “yes”;
else return “no.”

Proof of Theorem 1. Let G be a connected chordal graph. We find an MCS ordering σ of G started with z , and then use Algorithm 1. We first show its correctness: Vertex z is an MCS end-vertex of G if and only if z is the last visited vertex. The if direction is correct because the algorithm conducts MCS, and Hence we focus on the only if direction. Let S_1, \dots, S_k be the set of separators specified in Theorem 11, and let σ^+ denote the ordering returned by Algorithm 1. We show by induction that for each $1 \leq i \leq k$, vertices in all the other components of $G - S_i$ are visited before those in the same component with z .

Let T'_1 be the component of $G - S_1$ containing z . By Proposition 9 and Corollary 7, vertices in T'_1 are at the beginning of σ . In each component of $G - S_1$, there is a vertex adjacent to all vertices in S_1 . When the first vertex in T'_1 is being visited, it has precisely $|S_1|$ visited neighbors, i.e., S_1 . By the selection of vertices in step 1, all other components have been finished. Thus, T'_1 is the last visited component of $G - S_1$.

For the inductive step, suppose that the induction hypothesis is true for all p with $1 \leq i \leq p < k$, we show it is also true for $p + 1$. For $1 < i \leq k$, let T'_i be the component of $T_{i-1} - S_i$ containing z , and let T_i be the subgraph induced by $V(T'_i) \cup S_i$. Let $v \in T_{p+1}$ be the vertex satisfying $v <_{\sigma^+} u$ for all $u \in T_{p+1} \setminus \{v\}$. Then $S_{p+1} \subseteq N(v)$ and $x <_{\sigma^+} v$ for all $x \in S_{p+1}$. Since S_{p+1} is a minimum separator of T_p , any other component of $T_p - S_{p+1}$ has a vertex adjacent to all of S_{p+1} . Such a vertex x would satisfy $v <_{\sigma} x$ because of Proposition 9 and Corollary 7, and then be chosen by step 1 before v . Now that all the vertices in $G - N[z]$ and the non-simplicial vertices in $N[z]$ have been visited, the only remaining vertices are true twins of z . Since $\sigma(z) = 1$, it has to be the last visited. We have proved the correctness.

We now analyze the running time. The only difference between the algorithm and the original MCS algorithm is step 1.2. We need to compare the σ -numbers of vertices in D . It needs to be done n times, and each time takes $O(n)$ time, and hence the extra time is $O(n^2)$. Together with the time for MCS itself, the total running time is $O(n^2 + m) = O(n^2)$. ◀

4 Maximum cardinality search on weakly chordal graphs

A graph G is *weakly chordal* if neither G nor its complement contains an induced cycle on five or more vertices. It is well known that all chordal graphs are weakly chordal. To prove NP-completeness of the MCS end vertex problem on weakly chordal graphs, we use a reduction from the 3-satisfiability problem (3-SAT), in which each clause comprises precisely three literals.

Given an instance \mathcal{I} of 3-SAT with p variables and q literals, we construct a graph G as follows (see Fig. 3 for an example). Let the variables and clauses of \mathcal{I} be denoted by x_1, x_2, \dots, x_p and c_1, c_2, \dots, c_q , respectively. For each literal, (including those that do not occur in any clause,) we introduce a vertex; let L denote this set of $2p$ literal vertices. For each literal vertex, we add edges between it and other vertices in L , with the only exception of its negation. We also introduce a set C of q clause vertices, each for a different clause; they forms an independent set. For each $\ell \in L$ and $c \in C$, we add an edge ℓc if the literal ℓ does not occur in the clause c . Therefore, each clause vertex has $2p - 3$ neighbors in L . Finally, we add seven extra vertices $a_1, a_2, u_1, u_2, b, y, z$ and edges $a_1 a_2, u_1 u_2, yz, \{b, z\} \times L$ and $\{a_2, u_1, u_2, y\} \times (L \cup C)$.

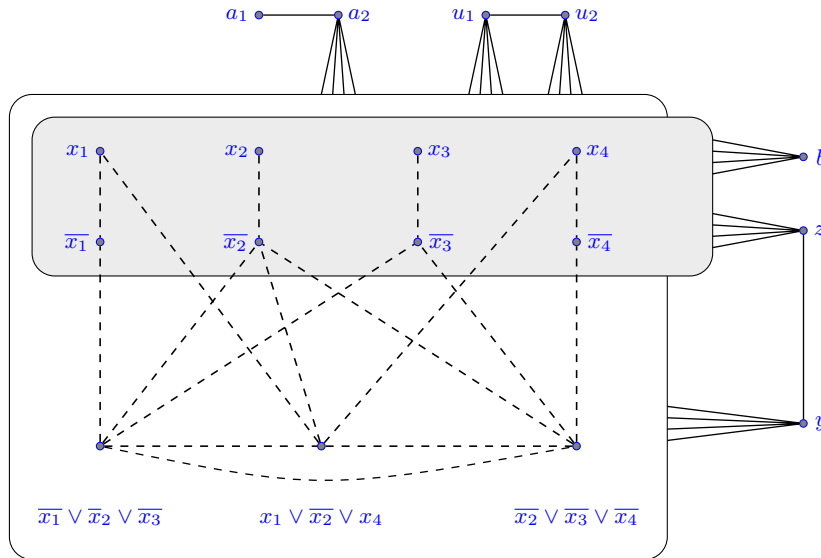


Figure 3 Construction for NP-completeness proof of the MCS end vertex problem on weakly chordal graphs. The 3-SAT instance has four variables and three clauses, $(\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)$, $(x_1 \vee \bar{x}_2 \vee x_4)$, $(\bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_4)$, i.e., $p = 4$ and $q = 3$. The $2p$ literal vertices are shown in the small gray box, and the q clause vertices are in the big box. In the boxes, two vertices are nonadjacent if there is a dashed line between them, and adjacent otherwise. Vertices b and z are adjacent to all literal vertices, while vertices a_2, u_1, u_2 , and y are adjacent to all literal vertices and all clause vertices. The MCS ordering $\langle a_1, a_2, x_1, \bar{x}_2, x_3, x_4, b, \bar{x}_1, x_2, \bar{x}_3, \bar{x}_4, u_1, u_2, y, c_1, c_2, c_3, z \rangle$ of G corresponds to the satisfying assignment in which all variables but x_2 are set to be true.

► **Proposition 12.** *The graph G constructed above is a weakly chordal graph.*

Proof. We need to show that neither G nor \overline{G} contains an induced cycle on five or more vertices. We proceed as follows: We identify a vertex $v \in V(G)$ such that G contains an induced cycle on five or more vertices if and only if $G - v$ contains an induced cycle on five or more vertices, and then consider $G - v$. The following properties are straightforward:

- (i) A vertex on any induced cycle on five or more vertices has degree at least two.
- (ii) A simplicial vertex is not on any induced cycle on five or more vertices.
- (iii) An induced cycle on five or more vertices cannot contain a pair of true twins or false twins, and when it contains one of them, this vertex can be replaced by the other.
- (iv) If a vertex is on an induced cycle on five or more vertices, then it has at least two non-neighbors, and there is at least one edge among these non-neighbors.

We can reduce G to $G - \{a_1\}$ because $d(a_1) = 1$ and (i); then to $G - \{a_1, u_2\}$ because u_1 and u_2 are true twins and (iii); to $G - \{a_1, u_1, u_2\}$ because u_1 and a_2 are false twins in $G - \{a_1, u_2\}$ and (iii); to $G - \{a_1, u_1, u_2, y\}$ because the only two remaining non-neighbors of y , namely, a_2 and b , are not adjacent to each other and (iv); to $G - \{a_1, u_1, u_2, y, a_2\}$ for the same reason; to $G - \{a_1, u_1, u_2, y, a_2, b\}$ because z and b are false twins in $G - \{a_1, u_1, u_2, y, a_2\}$ and (iii); and finally to $G - \{a_1, u_1, u_2, y, a_2, b, z\}$ because the only non-neighbors of z , namely, C , are independent and (iv). The remaining graph is $G[L \cup C]$. Suppose that there is an induced cycle H on five or more vertices. It must intersect both L and C , since each vertex in L has only one non-neighbor in it, and since C is independent. Let $v \in C$ be a vertex on this cycle. Its two neighbors on H have to be from L ; and since they are nonadjacent to each other, they have to be x and \bar{x} for some variable x . Since both x and \bar{x} are adjacent to all other vertices in L , the other ≥ 2 vertices on H have to be from C . But this is impossible because C is independent.

Now we consider \overline{G} . It can be reduced to $\overline{G} - \{a_1\}$ because a_1 has only one non-neighbor and (iv); then to $\overline{G} - \{a_1, u_2\}$ because u_1 and u_2 are false twins and (iii); to $\overline{G} - \{a_1, u_1, u_2\}$ because u_1 and a_2 are true twins in $\overline{G} - \{a_1, u_2\}$ and (iii); to $\overline{G} - \{a_1, u_1, u_2, y\}$ because y is simplicial in $\overline{G} - \{a_1, u_1, u_2\}$ and (ii); to $\overline{G} - \{a_1, u_1, u_2, y, b\}$ because z and b are true twins in $\overline{G} - \{a_1, u_1, u_2, y\}$ and (iii); to $\overline{G} - \{a_1, u_1, u_2, y, b, a_2\}$ because the degree of a_2 is one in $\overline{G} - \{a_1, u_1, u_2, y, b\}$ and (i); and finally to $\overline{G} - \{a_1, u_1, u_2, y, a_2, b, z\}$ because z is simplicial in $\overline{G} - \{a_1, u_1, u_2, y, b, a_2\}$ and (ii). The remaining graph is $\overline{G}[L \cup C]$. Suppose that there is an induced cycle H on five or more vertices. Since C is a clique, H contains at most two vertices from C . In other words, at least 3 vertices on H are from L , but this is impossible because each vertex in L has only one neighbor in L . Thus, G is weakly chordal. ◀

Proof of Theorem 2. It is clear that the MCS end vertex problem is in NP, and we now show that it is NP-hard. Let \mathcal{I} be an instance of 3-SAT, and let G be the graph constructed from \mathcal{I} . We show that z is an MCS end-vertex of G if and only if \mathcal{I} has a satisfying assignment.

For the if direction, suppose that \mathcal{I} is satisfiable, and we give an MCS ordering σ as follows. Let us fix a satisfying assignment of \mathcal{I} , and let T be the set of variables that are set to be true. The starting vertex is a_1 , which is followed by a_2 ; visited after them are $\{x \mid x \in T\} \cup \{\bar{x} \mid x \notin T\}$, (i.e., the literal vertices corresponding to true literals,) in any order. After these $p + 2$ vertices, each of y, z, u_1, u_2, b , and each of the unvisited literal vertices has p visited neighbors. On the other hand, each clause vertex has at most p visited neighbors: Each clause contains a true literal, and hence each clause vertex has at least one non-neighbor in the visited literal vertices.

Then $\sigma(b) = (p + 3)$. Since b is adjacent to only literal vertices, the next vertex is one of them. On the other hand, since vertices $L \setminus T$ form a clique, they have to be visited between $p + 4$ and $2p + 3$, i.e., before others.

The remaining vertices are u_1, u_2, y, z , and clause vertices. Each of u_1, u_2, y , and z has $2p$ visited neighbors, while each clause vertex has only $2p - 2$, because each clause is nonadjacent to three literal vertices. Let u_1, u_2 , and y be visited next. After that, all the remaining vertices (z and all clause vertices) have the same number of visited neighbors, $2p + 1$. There is no edge among these vertices, so they can be visited in any order. We have thus obtained an MCS ordering of G ended with z .

We now prove the only if direction. Suppose that σ is an MCS ordering of G with $\sigma(z) = n$. Since $N(z) = N(b) \cup \{y\}$, visiting y before b would force z to be visited before b ; therefore, $b <_\sigma y <_\sigma z$. Likewise, $N(b) = L \subset L \cup C \subset N(y)$ and $b <_\sigma y$ demand

$$b <_\sigma c \text{ for all } c \in C. \quad (\star)$$

Since $d(a_1) = 1$, it is easy to verify that $\{\sigma(a_1), \sigma(a_2)\} = \{1, 2\}$; otherwise, σ must end with a_1 . The third vertex of σ has to be from $N(a_2)$, i.e., $L \cup C$. It cannot be from C because of (\star) . Therefore, $X = \{\ell \mid 3 \leq \sigma(\ell) \leq p + 2\} \subset L$: (1) For each variable, one literal vertex has more visited neighbors than b, z, y, u_1, u_2 ; (2) clause vertices cannot be visited before b . There cannot be any variable x such that both $x, \bar{x} \in X$, because $x\bar{x} \notin E(G)$. We claim that assigning a variable x to be true if and only if $x \in X$ is a satisfying assignment for \mathcal{I} . Suppose for contradiction that some clause c is not satisfied by this assignment. By the construction of G , the clause vertex c is adjacent to all vertices of X . After visiting the first $p + 2$ vertices, c has $p + 1$ visited neighbors, $(\{a_2\} \cup X)$, while any other unvisited vertex in $V(G) \setminus C$ has at most p visited neighbors. But then $\sigma(c) = k + 3$, contradicting (\star) . Therefore, all clauses are satisfied, and this completes the proof. \blacktriangleleft

5 Lexicographic depth-first search on chordal graphs

Berry et al. [3, Characterization 8.1] have given a full characterization of MNS end vertices on chordal graphs: A vertex z is an MNS end vertex if and only if it is simplicial and the minimal separators of G in $N(z)$ are totally ordered by inclusion. Since LDFS is a special case of MNS, its end vertices also have this property. We show that this condition is also sufficient for a vertex to be an LDFS end vertex.

Similar as DFS, LDFS visits a neighbor of the most recent vertex, or backtracks if all its neighbors have been visited. The difference lies on the choice when the vertex has more than one unvisited neighbors. Each unvisited vertex has a label, which is all its visited neighbors. When there are ties, it chooses a vertex with the lexicographically largest label. The following is actually a simple property of DFS.

► **Proposition 13.** *Let $X \subseteq V(G)$ such that $G[X]$ is connected. If an LDFS visits all vertices in $N(X)$ before the first vertex in X , then it visits vertices in X consecutively.*

► **Lemma 14.** *A vertex z of a chordal graph G is an LDFS end vertex if and only if it is simplicial and the minimal separators of G in $N(z)$ are totally ordered by inclusion.*

Proof. The only if direction follows from that all LDFS orderings are MNS orderings [11] and the result of Berry et al. [3]. For the if direction, suppose that S_1, \dots, S_k are the minimal separators in $N(z)$ and $S_1 \subset \dots \subset S_k$. It is easy to see that for all $1 \leq i \leq k$, each component

of $G - S_i$ not containing z is a component of $G - S_k$; let \mathcal{C} denote these components. We show an LDFS ordering σ of G as follows. It starts from visiting all vertices in S_1 , followed by components $C \in \mathcal{C}$ with $N(C) = S_1$, visited one by one. In the same manner, it deals with $S_2 \dots S_k$ in order. After that the only unvisited vertex are z and its true twins, of which it chooses z the last. We now verify that this is indeed a valid LDFS ordering. It is clear for S_1 . Since vertices in each component $C \in \mathcal{C}$ are visited after $N(C)$, By Proposition 13, it suffices to show the correctness when it visits a vertex in $N(z)$ and when it visits the first vertex of a new component $C \in \mathcal{C}$. When such a decision is made, the label of an unvisited vertex is either \emptyset or all visited vertices in $N(z)$, i.e., the most recently visited separator. So it is always correct to select a vertex from $N(z)$. When a vertex v in a component C is selected, the visited vertices in $N(z)$ are precisely $N(C)$, hence v does have the largest label. ◀

6 Breadth-first search on interval graphs

Interval graphs are intersection graphs of intervals on the real line. An interval graph is always chordal, and in particular, it has a clique tree that is a path [15]. Corneil et al. [10] gave a very simple linear-time algorithm for deciding whether a vertex z is an LBFS end vertex of an interval graph, which is very similar to Algorithm 1. They conducted an LBFS started from z , and then another LBFS that uses the first run to break ties. They proved that z is an LBFS end vertex if and only if it is the last of the second run. As shown in Fig. 4, however, this algorithm cannot be directly adapted to the BFS end vertex problem.



■ **Figure 4** A BFS started from z may end with s or w , but a BFS started from w has to end with u . (Note that a BFS started from s may end with z .)

If a graph has one and only one universal vertex, then each of the other vertices is a BFS end-vertex, but not itself. If it has two or more universal vertices, then every vertex can be a BFS end-vertex. Therefore, we may focus on graphs with no universal vertex. Such an interval graph has at least three maximal cliques.

► **Proposition 15** ([13]). *Let G be a connected interval graph, and let K_1, \dots, K_p be a clique path of G . Let $u \in K_1$ and $w \in K_p$ be two simplicial vertices.*

- (i) *Both u and w are LBFS end vertices.*
- (ii) *For any vertex $v \in V(G)$, one of u and w has the largest distance to v .*

It is known that a vertex z of an interval graph G can be an LBFS end vertex if and only if it is simplicial and $N[z]$ can be one of the two ends of a clique path of G [13]. However, a BFS may satisfy neither of the two conditions. In Fig. 4, for example, vertex z is not simplicial but can be a BFS end vertex. When z is not in an end clique, it should be close to one. Actually, it should be at distance at most two to one of the u and w as specified in Proposition 15. However, a BFS end vertex might be at distance two to both u and w .

For a fixed clique path K_1, \dots, K_p of an interval graph G , we let $\text{lp}(v)$ and $\text{rp}(v)$ denote, respectively, the smallest and the largest number i such that $v \in K_i$. We use $\text{dist}(u, v)$ to denote the distance between u and v .

► **Lemma 16.** *The BFS end vertex problem can be solved in $O(n + m)$ time on interval graphs.*

■ **Algorithm 2** Main procedure for BFS end vertex on interval graphs.

INPUT: A connected interval graph G , a clique path K_1, \dots, K_p of G , simplicial vertices $u \in K_1$ and $w \in K_p$, and $z \in V(G)$.

OUTPUT: Whether there exists a BFS ordering σ of G with $\sigma(z) = n$ and $u <_\sigma w$.

1. **if** $z = w$ **then return** “yes”;
2. **if** there exists a universal vertex in $V(G) \setminus \{z\}$ **then return** “yes”;
3. $X \leftarrow \{x \in V(G) : \text{dist}(x, z) = \text{dist}(x, w) \geq \text{dist}(x, u)\}$;
4. **if** $X = \emptyset$ **then return** “no”;
5. $s \leftarrow$ any vertex in $\arg \min_{v \in X} \text{lp}(v)$;
6. **if** $\text{rp}(z) < \text{lp}(s)$ **then return** “no”;
7. **if** $s = u$ **then return** “yes”;
8. **for each** vertex $v \in N(s)$ at distance $\text{dist}(s, u) - 1$ to u **do**
 if $\text{dist}(v, z) > \text{dist}(v, u)$ **then return** “yes”;
9. **return** “no.”

Proof. Let G be an interval graph; we may assume without loss of generality that G is connected. We use the algorithm of Corneil et al. [13] to build a clique path for G , and take simplicial vertices v_1, v_2 from the first and last cliques of the clique path. We call the procedure described in Algorithm 2 twice, first with $u = v_1, w = v_2$; in the second call, we reverse the clique path, and use $u = v_2, w = v_1$. Suppose that the procedure is correct, then vertex z is a BFS end vertex if and only if at least one of the two calls returns yes. In the rest we prove the correctness of the procedure and analyze its running time.

We start from characterizing the first vertex s of a BFS ordering σ with $\sigma(z) = n$ and $u <_\sigma w$, if one exists. Since $u <_\sigma w <_\sigma z$, we must have $\text{dist}(s, u) \leq \text{dist}(s, w) \leq \text{dist}(s, z)$. On the other hand, Proposition 15 implies $\text{dist}(s, z) \leq \max\{\text{dist}(s, u), \text{dist}(s, w)\} = \text{dist}(s, w)$. Therefore, a desired BFS ordering σ , if it exists, must start from a vertex s satisfying

$$\text{dist}(s, z) = \text{dist}(s, w) \geq \text{dist}(s, u). \quad (\dagger)$$

We argue that at least one of the following is true for z :

- on any shortest s - u path, z is adjacent to the 2nd to last vertex but no vertex before it.
 - on any shortest s - w path, z is adjacent to the 2nd to last vertex but no vertex before it.
- Let P_u be any s - u path and P_w any s - w path. Since they together form a u - w path that visits all the maximal cliques of G , vertex z is adjacent to at least one of these two paths. If z is adjacent to a vertex on P_u , then it has to be the last two; otherwise $\text{dist}(s, z) < \text{dist}(s, u)$. Since u is simplicial, z is adjacent to its neighbor on the path if $zu \in E(G)$. Therefore, z is always adjacent to the second to last vertex on this path. The same argument applies if z is adjacent to P_w .

The correctness of step 1 follows from Proposition 15. For step 2, note that if $v \neq z$ is a universal vertex, then $\langle v, u, w, \dots, z \rangle$ is such a BFS ordering. Steps 3 and 4 are justified by (\dagger) . When the algorithm reaches step 5, X is not empty, and hence s is well defined. Let $q = \text{dist}(s, z) = \text{dist}(s, w)$. Note that $q \geq 2$ because s is not universal. Hence, $z, w \notin N(s)$.

We show the correctness of step 6 by contradiction. Suppose that $\text{rp}(z) < \text{lp}(s)$ but there exists a BFS ordering σ with $\sigma(z) = n$ and $u <_\sigma w$. Let s' be the first vertex of σ . Since $s' \in X$, the selection of s implies $\text{lp}(s) \leq \text{lp}(s')$. Then $\text{rp}(u) = 1 \leq \text{rp}(z) < \text{lp}(s) \leq \text{lp}(s')$, therefore, $\text{dist}(s', u) \geq 2$. In this case, on any shortest s' - u path, z is adjacent to the second to last vertex but no vertex before it. Hence, $\text{dist}(s', z) = \text{dist}(s', u) = \text{dist}(s', w)$; let it be q' . Since $u <_\sigma w$, there must be some neighbor u'' of u at distance $q' - 1$ to s' visited before neighbors of w . The vertex u'' cannot be universal, hence nonadjacent to w . But u''

is adjacent to z , which implies $z <_{\sigma} w$, a contradiction. Therefore, step 6 is correct, which means $\text{rp}(s) < \text{lp}(z)$ because s and z are not adjacent. Let $s = w_0, w_1, \dots, w_{q-1}, w_q = w$ be a shortest s - w path. Note that $w_{q-1} \in N(z)$.

For step 7, it suffices to give the following BFS ordering, which starts with $s = u$. Of all vertices at distance i to s , $1 \leq i \leq q$, the first visited vertex is w_i . Note that every vertex is adjacent to w_1, \dots, w_{q-1} . From $\text{rp}(w_{q-1}) = p$ it can be inferred that all vertices at distance q to s are adjacent to w_{q-1} . Since w_{q-1} is the first visited vertex at level $q-1$, vertices at distance q to s can be visited in any order. Therefore, we can have a BFS ordering σ of G with $u <_{\sigma} w$ and $\sigma(z) = n$.

We now consider step 8, for which we show that there exists a BFS ordering σ with $\sigma(s) = 1$, $\sigma(v) = 2$, $\sigma(z) = n$, and $u <_{\sigma} w$. Note that $\text{dist}(w_1, z) = \text{dist}(w_1, w) = q-1$. Therefore $v \neq w_1$; otherwise step 5 should have chosen v because $\text{lp}(v) < \text{lp}(s)$. For $1 \leq i \leq q-1$, vertex w_i is always visited in the earliest possible time; in particular, $\sigma(w_1) = 3$. Since v is on a shortest s - u path, u is a descendant of v in the BFS tree generated by σ . On the other hand, since both $\text{dist}(v, z)$ and $\text{dist}(v, w)$ are larger than $\text{dist}(v, u)$, either vertices z and w are not descendants of v , or they are at a lower level than u . In either case, we have $u <_{\sigma} w$. When w_q is visited, all the unvisited vertices are at distance q to s and adjacent to w_{q-1} . Thus, we can have $\sigma(z) = n$.

We are now at the last step. Note that the algorithm can reach here only when $\text{dist}(s, z) = \text{dist}(s, w) = \text{dist}(s, u)$: The condition of step 8 must be true if $\text{dist}(s, u) < q$. Suppose for contradiction that there exists a BFS ordering σ with $\sigma(z) = n$ and $u <_{\sigma} w$ but no vertex satisfies the condition in step 8. Let s' be the starting vertex of σ . Since $s' \in X$ and by the selection of s , we have $\text{lp}(s') \geq \text{lp}(s)$, which implies $\text{dist}(s', u) \geq \text{dist}(s, u)$. Note that s' is adjacent to any s - w path, and hence its distance to w is at most $q+1$. In summary,

$$q = \text{dist}(s, u) \leq \text{dist}(s', u) \leq \text{dist}(s', w) \leq q+1.$$

Let Y denote all vertices at distance $q-1$ to u , and let Z denote all vertices at distance $q-1$ to w . Note that Y is disjoint from Z : A vertex in $v \in Y \cap Z$ would be adjacent to s , and have the same distance to u, w , and z , but then it contradicts the selection of s because $\text{lp}(v) < \text{lp}(s)$. Since no vertex in Y satisfies the condition of step 8, $\text{dist}(v, z) = \text{dist}(v, u)$ for all $v \in Y \cap N(s)$.

If $\text{dist}(s', u) = \text{dist}(s', z) = \text{dist}(s', w) = q$, then to have $u <_{\sigma} w$, one vertex in $Y \cap N(s)$ must be visited before Z . But this would force z to be visited before w , because z is at distance $q-1$ to all vertices in $Y \cap N(s)$. Now that $\text{dist}(s', w) = q+1$, if $\text{dist}(s', u) = q$, then at least one vertex $v \in Y$ is adjacent to s' ; it is in $N(s)$ because $\text{lp}(s) \leq \text{lp}(s')$. But then $\text{dist}(s', z) \leq 1 + \text{dist}(v, z) = 1 + q - 1 = q < \text{dist}(s', w)$. Therefore, $\text{dist}(s', u) = q+1$ as well. Each vertex in $Y \cup Z$ has distance at least two to s' . Of vertices at distance two to s' , one vertex in $Y \cap N(s)$ must be visited before Z , but then we have the same contradiction as in the first case of this paragraph. Therefore, step 9 is also correct and this concludes the proof of correctness.

We now analyze the running of the algorithm. Steps 1 and 2 can be easily checked in $O(n+m)$ time. For step 3, it suffices to calculate the distances between z, w, u and all other vertices; this can be done by visiting the maximal cliques one by one. Steps 4–7 can be done in $O(n)$ time. Step 8 can be checked in $O(n)$ time: We have already calculated the distance between z and v . Therefore, the total running time is $O(n+m)$. ◀

7 Graph searches on general graphs

We now describe an algorithm for deciding whether a vertex z of a general graph is an MCS end vertex. For each subset $X \subseteq V(G) \setminus \{z\}$, we define $f(X)$ to be true if there exists an MCS visiting X before others, and false otherwise. The question whether z can be an end vertex is then simply the value of $f(V(G) \setminus \{z\})$. For a set X with $f(X)$ is true and $v \notin X$, let $g(X, v)$ indicate whether there exists a search ordering that visits v after X and before others. We have $f(X) = \bigvee_{v \in X} (f(X \setminus \{v\}) \wedge g(X \setminus \{v\}, v))$. For MCS, $g(X, v)$ can be calculated in linear time, and thus we have a simple $O(2^n n^{O(1)})$ -time algorithm similar to the classic Held–Karp algorithm [18].

Let us consider then BFS. We may fix the starting vertex s , which can be found by enumerating all the other $n - 1$ vertices. Let $\ell = \max_{v \in V(G)} \text{dist}(s, v)$, and for $1 \leq i \leq \ell$, let L_i denote the set of vertices at distance i to s . Suppose that there is a BFS ordering σ started with s and ended with z , then $z \in L_\ell$. Clearly, vertices in $L_{\ell-1}$ are visited after those in $L_{\ell-2}$ and before L_ℓ . Let u be the first visited vertex in $L_{\ell-1}$ that is adjacent to z , and let X be those vertices in $L_{\ell-1}$ visited before u . Since z is the last vertex, all vertices in $L_\ell \setminus N(X)$ must be adjacent to u . We do not need any constraint on the order of vertices in $L_{\ell-1} \setminus (X \cup \{u\})$ being visited. Therefore, the information we need at level $\ell - 1$ are the set X and the vertex u . We can generalize this observation to give a recursive formula for the BFS end vertex problem. For lack of space, the proofs in this section are left for the full version of the paper.

► **Lemma 17.** *There is a $2^n \cdot n^{O(1)}$ -time algorithm for solving the BFS end vertex problem.*

In the last we consider DFS. Recall that a DFS sets two timestamps for a vertex v , first when it is visited, and second when it is *finished*, i.e., when all its neighbors have been examined and the search backtracks to the vertex that discovered v (or terminates when v is the source vertex). Note that when a vertex is finished, all its neighbors have been visited, and all but one of them have been finished. In particular, when the last vertex is visited, no vertex in its neighborhood has been finished. At any moment, the set of vertices that have been visited but not finished form a path in the depth-first tree. Suppose that z is the end vertex of a DFS ordering σ of G . If v is the earliest visited neighbor of z , then all the vertices after v are descendants of v in the depth-first tree.

The following simple property of DFS is stronger than Proposition 13. In a DFS ordering σ , if the set of vertices after v , i.e., $\{u : v <_\sigma u\}$, and v induce a connected subgraph, then their visiting order is irrelevant to vertices visited before v .

► **Proposition 18.** *Let σ be a DFS ordering of a graph G , and let X be the set of last visited $|X|$ vertices in σ . The sub-ordering $\sigma|_X$ is a DFS ordering of $G[X]$. Moreover, if $G[X]$ is connected, then σ remains a DFS ordering of G after replacing $\sigma|_X$ with any DFS ordering of $G[X]$ that starts with $\arg \min_{v \in X} \sigma(v)$.*

► **Lemma 19.** *There is a $2^n \cdot n^{O(1)}$ -time algorithm for solving the DFS end vertex problem.*

References

- 1 Jesse Beisegel, Carolin Denkert, Ekkehard Köhler, Matjaz Krnc, Nevena Pivac, Robert Scheffler, and Martin Strehler. On the End-Vertex Problem of Graph Searches. *Discrete Mathematics & Theoretical Computer Science*, 21(1), 2019. URL: <http://dmtcs.episciences.org/5572>.
- 2 Philip A. Bernstein and Nathan Goodman. Power of Natural Semijoins. *SIAM Journal on Computing*, 10(4):751–771, 1981. doi:10.1137/0210059.

- 3 Anne Berry, Jean R. S. Blair, Jean Paul Bordat, and Geneviève Simonet. Graph Extremities Defined by Search Algorithms. *Algorithms*, 3(2):100–124, 2010. doi:10.3390/a3020100.
- 4 Anne Berry and Jean Paul Bordat. Separability Generalizes Dirac’s Theorem. *Discrete Applied Mathematics*, 84(1-3):43–53, 1998. doi:10.1016/S0166-218X(98)00005-5.
- 5 Jean R. S. Blair and Barry W. Peyton. An introduction to chordal graphs and clique trees. In J. A. George, J. R. Gilbert, and J. W.-H. Liu, editors, *Graph Theory and Sparse Matrix Computation*, volume 56 of *IMA*, pages 1–29. Springer-Verlag, 1993.
- 6 Pierre Charbit, Michel Habib, and Antoine Mamcarz. Influence of the tie-break rule on the end-vertex problem. *Discrete Mathematics & Theoretical Computer Science*, 16(2):57–72, 2014. URL: <http://dmtcs.episciences.org/2081>.
- 7 Derek G. Corneil. A simple 3-sweep LBFS algorithm for the recognition of unit interval graphs. *Discrete Applied Mathematics*, 138(3):371–379, 2004. doi:10.1016/j.dam.2003.07.001.
- 8 Derek G. Corneil. Lexicographic Breadth First Search - A Survey. In Juraž Hromkovic, Manfred Nagl, and Bernhard Westfechtel, editors, *Graph-Theoretic Concepts in Computer Science (WG)*, volume 3353 of *LNCS*, pages 1–19. Springer, 2004. doi:10.1007/978-3-540-30559-0_1.
- 9 Derek G. Corneil, Barnaby Dalton, and Michel Habib. LDFS-based certifying algorithm for the minimum path cover problem on cocomparability graphs. *SIAM Journal on Computing*, 42(3):792–807, 2013. doi:10.1137/11083856X.
- 10 Derek G. Corneil, Ekkehard Köhler, and Jean-Marc Lanlignel. On end-vertices of Lexicographic Breadth First Searches. *Discrete Applied Mathematics*, 158(5):434–443, 2010. doi:10.1016/j.dam.2009.10.001.
- 11 Derek G. Corneil and Richard Krueger. A Unified View of Graph Searching. *SIAM Journal on Discrete Mathematics*, 22(4):1259–1276, 2008. doi:10.1137/050623498.
- 12 Derek G. Corneil, Stephan Olariu, and Lorna Stewart. Linear Time Algorithms for Dominating Pairs in Asteroidal Triple-free Graphs. *SIAM Journal on Computing*, 28(4):1284–1297, 1999. A preliminary version appeared in ICALP 1995. doi:10.1137/S0097539795282377.
- 13 Derek G. Corneil, Stephan Olariu, and Lorna Stewart. The LBFS Structure and Recognition of Interval Graphs. *SIAM Journal on Discrete Mathematics*, 23(4):1905–1953, 2009. doi:10.1137/S0895480100373455.
- 14 Gabriel A. Dirac. On rigid circuit graphs. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 25(1):71–76, 1961. doi:10.1007/BF02992776.
- 15 Delbert R. Fulkerson and Oliver A. Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15(3):835–855, 1965. doi:10.2140/pjm.1965.15.835.
- 16 Philippe Galinier, Michel Habib, and Christophe Paul. Chordal Graphs and Their Clique Graphs. In Manfred Nagl, editor, *Graph-Theoretic Concepts in Computer Science (WG)*, volume 1017 of *LNCS*, pages 358–371. Springer, 1995. doi:10.1007/3-540-60618-1_88.
- 17 Michel Habib, Ross M. McConnell, Christophe Paul, and Laurent Viennot. Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. *Theoretical Computer Science*, 234(1-2):59–84, 2000. doi:10.1016/S0304-3975(97)00241-7.
- 18 Michael Held and Richard M. Karp. A Dynamic Programming Approach to Sequencing Problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210, 1962. doi:10.1137/0110015.
- 19 John E. Hopcroft and Robert Endre Tarjan. Efficient Planarity Testing. *Journal of the ACM*, 21(4):549–568, 1974. doi:10.1145/321850.321852.
- 20 Russell Impagliazzo and Ramamohan Paturi. On the Complexity of k -SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001. A preliminary version appeared in CCC 1999. doi:10.1006/jcss.2000.1727.
- 21 Donald J. Rose, Robert Endre Tarjan, and George S. Lueker. Algorithmic Aspects of Vertex Elimination on Graphs. *SIAM Journal on Computing*, 5(2):266–283, 1976. A preliminary version appeared in STOC 1975. doi:10.1137/0205021.

- 22 Ravi Sethi. Scheduling Graphs on Two Processors. *SIAM Journal on Computing*, 5(1):73–82, 1976. doi:10.1137/0205005.
- 23 Douglas R. Shier. Some aspects of perfect elimination orderings in chordal graphs. *Discrete Applied Mathematics*, 7(3):325–331, 1984. doi:10.1016/0166-218X(84)90008-8.
- 24 Klaus Simon. A New Simple Linear Algorithm to Recognize Interval Graphs. In *Computational Geometry - Methods, Algorithms and Applications, International Workshop on Computational Geometry CG'91, Bern, Switzerland, March 21-22, 1991*, pages 289–308, 1991. doi:10.1007/3-540-54891-2_22.
- 25 Robert Endre Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972. A preliminary version appeared in SWAT (FOCS) 1971. doi:10.1137/0201010.
- 26 Robert Endre Tarjan and Mihalis Yannakakis. Simple Linear-Time Algorithms to Test Chordality of Graphs, Test Acyclicity of Hypergraphs, and Selectively Reduce Acyclic Hypergraphs. *SIAM Journal on Computing*, 13(3):566–579, 1984. With Addendum in the same journal, 14(1):254-255, 1985. doi:10.1137/0213035.
- 27 Marc Tedder, Derek G. Corneil, Michel Habib, and Christophe Paul. Simpler linear-time modular decomposition via recursive factorizing permutations. In *Automata, Languages and Programming (ICALP)*, volume 5125 of *LNCS*, pages 634–645. Springer-Verlag, 2008. doi:10.1007/978-3-540-70575-8_52.

Lower Bound for Non-Adaptive Estimation of the Number of Defective Items

Nader H. Bshouty

Department of Computer Science, Technion, Haifa, Israel
bshouty@cs.technion.ac.il

Abstract

We prove that to estimate within a constant factor the number of defective items in a non-adaptive randomized group testing algorithm we need at least $\tilde{\Omega}(\log n)$ tests. This solves the open problem posed by Damaschke and Sheikh Muhammad in [6, 7].

2012 ACM Subject Classification Mathematics of computing; Mathematics of computing → Discrete mathematics; Mathematics of computing → Probabilistic algorithms; Theory of computation → Probabilistic computation

Keywords and phrases Group Testing, Estimation, Defective Items

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.2

Related Version <https://eccc.weizmann.ac.il/report/2018/053/>

1 Introduction

Let X be a set of *items* that contains *defective items* $I \subseteq X$. In Group testing, we *test* (*query*) a subset $Q \subset X$ of items and the answer to the query is 1 if Q contains at least one defective item, i.e., $Q \cap I \neq \emptyset$, and 0 otherwise. Group testing was originally introduced as a potential approach to the economical mass blood testing, [8]. However it has been proven to be applicable in a variety of problems, including DNA library screening, [18], quality control in product testing, [21], searching files in storage systems, [14], sequential screening of experimental variables, [16], efficient contention resolution algorithms for multiple-access communication, [14, 25], data compression, [12], and computation in the data stream model, [5]. See a brief history and other applications in [4, 9, 10, 13, 17, 18] and references therein.

Estimating the number of defective items to within a constant factor λ is the problem of finding an integer D that satisfies¹ $|I| \leq D \leq \lambda|I|$. This problem is extensively used in biological and medical applications [2, 22]. It is used to estimate the proportion of organisms capable of transmitting the aster-yellows virus in a natural population of leafhoppers [23], estimating the infection rate of yellow-fever virus in a mosquito population [24] and estimating the prevalence of a rare disease using grouped samples to preserve individual anonymity [15].

In *adaptive algorithms*, the queries can depend on the answers to the previous ones. In the *non-adaptive algorithms* they are independent of the previous one and; therefore, one can ask all the queries in one parallel step. In many applications in group testing non-adaptive algorithms are most desirable.

Estimating the number of defective items to within a constant factor with an *adaptive* deterministic, Las Vegas and Monte Carlo algorithms is studied in [1, 3, 6, 7, 11, 20]. For $|X| = n$ items and $|I| = d$ defective items the bounds are $\Theta(d \log(n/d))$ queries for Las Vegas and Deterministic algorithms and $\Theta(\log \log d + \log(1/\delta))$ queries for Monte Carlo algorithm [1, 11]. There are also polynomial time algorithms that achieve such bounds [1, 11].

¹ In all the applications in group testing the estimation $\lambda|I| \leq D \leq \lambda|I|$ is not interesting.



© Nader H. Bshouty;

licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 2; pp. 2:1–2:9

Leibniz International Proceedings in Informatics



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

In this paper we study this problem in the non-adaptive setting. We first show that any deterministic and Las Vegas algorithm must ask at least $\Omega(n)$ queries. For randomized algorithm with any *constant* failure probability δ , Damaschke and Sheikh Muhammad give in [7] a non-adaptive randomized algorithm that asks $O(\log n)$ queries and with probability at least $1 - \delta$ returns an integer D such that $D \geq d$ and $\mathbf{E}[D] = O(d)$. In this paper we give a polynomial time Monte Carlo algorithm that asks $O(\log(1/\delta) \log n)$ queries and with probability at least $1 - \delta$ estimates the number of defective items to within a constant factor. They then prove in [6] the lower bound $\Omega(\log n)$ queries, but only for algorithms that choose each item in each query randomly and independently with some fixed probability. They conjecture that $\Omega(\log n)$ queries are needed for *any* randomized algorithm with constant failure probability. In this paper we prove this conjecture (up to $\log \log n$ factor). We show that for any non-adaptive randomized algorithm that with probability at least $3/4$ estimates the number of defective items to within a constant factor must ask at least

$$s = \Omega\left(\frac{\log n}{\log \log n}\right) = \tilde{\Omega}(\log n)$$

queries.

This paper is organised as follows: In Section 2 we give some preliminary results. In Section 3 we give the proof of the above lower bound. The lower bounds will be for an estimation within the factor of 1.5 and confidence $3/4$, but it will be clear from the proof that this can be replaced by any constant factor λ and any constant confidence δ . In Section 4 we give the lower bound $\Omega(n)$ for any deterministic algorithm. In Section 5 we give the upper bound. The technique for the upper bound is standard and implicitly follows from [7, 11]. It is given for completeness.

2 Preliminary Results

In this section we give some definitions and then prove some preliminary results.

We will consider the set of *items* $X = [n] = \{1, 2, \dots, n\}$ and the set of *defective items* $I \subseteq X$. The algorithm knows n and has an access to an oracle \mathcal{O}_I . The algorithm can ask the oracle \mathcal{O}_I a *query* $Q \subset X$ and the oracle answers $\mathcal{O}_I(Q) := 1$ if $Q \cap I \neq \emptyset$ and $\mathcal{O}_I(Q) := 0$ otherwise. We say that algorithm A λ -*estimates* the number of defective items if for every $I \subseteq X$ it runs in polynomial time in n , asks queries to the oracle \mathcal{O}_I and returns an integer D such that $|I| \leq D \leq \lambda|I|$. If λ is constant then we say that the algorithm *estimates the number of defective items to within a constant factor*. Our goal is to find such an algorithm that asks a minimum number of queries in the worst case.

For an algorithm A that asks queries we denote by $A(I)$ the output of A when it runs with the oracle \mathcal{O}_I . When the algorithm is randomized then we write $A(\sigma, I)$ where σ is the random seed of the algorithm.

We now prove two results that will be used for the lower bound:

► **Lemma 1.** *Let k be any real number. Let N' be a finite set of elements and s be an integer. Let S be a probability space of s -tuples $W = (w_1, w_2, \dots, w_s) \in N'^s$. Let $N \subseteq N'$ and $N = N_1 \cup N_2 \cup \dots \cup N_r$ be a partition of N to r disjoint sets. There is i_0 such that for a random $W \in S$, the probability that at least k of the elements (coordinates) of W are in N_{i_0} , is at most $s/(kr)$.*

Equivalently, there is i_0 such that, with probability at least $1 - s/(kr)$, the number of elements in W that are in N_{i_0} is less than k .

Proof. Define the random variables $X_i, i = 1, \dots, r$, where $X_i(W) = 1$ if at least k of the elements of W are in N_i and 0 otherwise. Obviously, $k(X_1 + \dots + X_r) \leq s$ and therefore

$$\mathbf{E}[X_1] + \dots + \mathbf{E}[X_r] = \mathbf{E}[X_1 + \dots + X_r] \leq \frac{s}{k}.$$

Therefore there is i_0 such that $\Pr[X_{i_0} = 1] = \mathbf{E}[X_{i_0}] \leq s/(kr)$. ◀

► **Lemma 2.** *Let $X' \subseteq X = [n]$. Let \mathcal{D} be the probability space of random uniform subsets $I \subset X'$ of size d and \mathcal{D}' be the probability space of random uniform and independent d chosen elements $I = \{x_1, \dots, x_d\} \subseteq X'$ with replacement. Let A be any event and B be the event that $I \in \mathcal{D}'$ has size d , i.e., x_1, \dots, x_d are distinct. Then*

$$\Pr_{\mathcal{D}'}[A] + \Pr_{\mathcal{D}'}[\bar{B}] \geq \Pr_{\mathcal{D}}[A] \geq \Pr_{\mathcal{D}'}[A] - \Pr_{\mathcal{D}'}[\bar{B}].$$

Proof. Since

$$\begin{aligned} \Pr_{\mathcal{D}'}[A] &= \Pr_{\mathcal{D}'}[A|B]\Pr_{\mathcal{D}'}[B] + \Pr_{\mathcal{D}'}[A|\bar{B}]\Pr_{\mathcal{D}'}[\bar{B}] \\ &\leq \Pr_{\mathcal{D}'}[A|B] + \Pr_{\mathcal{D}'}[\bar{B}] = \Pr_{\mathcal{D}}[A] + \Pr_{\mathcal{D}'}[\bar{B}], \end{aligned}$$

we have $\Pr_{\mathcal{D}}[A] \geq \Pr_{\mathcal{D}'}[A] - \Pr_{\mathcal{D}'}[\bar{B}]$. In the same way we have $\Pr_{\mathcal{D}}[\bar{A}] \geq \Pr_{\mathcal{D}'}[\bar{A}] - \Pr_{\mathcal{D}'}[\bar{B}]$ which implies the left-hand side inequality. ◀

3 Lower Bound for Randomized Algorithms

In this section we prove the lower bound for the number of queries in any non-adaptive randomized algorithm that λ -estimates the number of defective items. We give the proof for $\lambda = 1.5$ and confidence $\delta = 1/4$. The proof for any other constants λ and δ is similar.

The idea of the proof is the following. Suppose there is a randomized algorithm A that asks $\log n/(c \log \Delta)$ queries where $\Delta = \log n$ and c is a large constant. We partition the interval $[0, n]$ of all the possible sizes $|Q|$ of the queries Q into $\Theta(\log n/\log \Delta)$ disjoint sets $N_i = [n/\Delta^{4i+4}, n/\Delta^{4i}]$ for integers i . We then show, by Lemma 2, that with high probability, there is an interval N_{i_0} such that no query Q asked by the algorithm satisfies $|Q| \in N_{i_0}$. That is, with high probability, there is no query with a size that falls in $N_{i_0} = [n/\Delta^{4i_0+4}, n/\Delta^{4i_0}]$. We then show that if we choose a random uniform set of defective items I of size $d' := \Delta^{4i_0+2}$ or $2d' = 2\Delta^{4i_0+2}$ then, with high probability, all the queries of sizes more than n/Δ^{4i_0} will have answer 1 and all the queries of sizes less than n/Δ^{4i_0+4} will have answer 0. So the only useful queries are those that fall in N_{i_0} that, by Lemma 1, with high probability, there are none. Therefore, with high probability, the algorithm fails to distinguish between sets of defective sets of size d' and of size $2d'$. This implies that algorithm A cannot estimate, with high probability, the size of the defective sets within a factor of 1.5. Therefore any randomized algorithm that, with high probability, estimates the size of the defective sets within a factor of 1.5 must ask at least $\log n/(c \log \Delta) = \Omega(\log n/\log \log n)$ queries.

We now give the proof.

► **Theorem 3.** *Any non-adaptive Monte Carlo randomized algorithm that with probability at least $3/4$, 1.5-estimates the number of defective items must ask at least*

$$s = \Omega\left(\frac{\log n}{\log \log n}\right)$$

queries.

2:4 Estimation of the Number of Defective Items

Proof. Let c be a large enough constant. Suppose, for the contrary, there is a non-adaptive Monte Carlo algorithm $A(\sigma, I)$ that chooses a random sequence of queries $M := Q_1, \dots, Q_s \subseteq X = [n]$ from some probability space where $s = \Delta/(c \log \Delta)$ and $\Delta = \log n$, asks queries to \mathcal{O}_I and with probability at least $3/4$, 1.5-estimates the number of defective items $|I|$. Let $r = \Delta/(16 \log \Delta)$ and let

$$N_i = [n/\Delta^{4i+4}, n/\Delta^{4i}] := \{x \mid n/\Delta^{4i+4} < x \leq n/\Delta^{4i}\},$$

$i = 0, 1, \dots, r-1$, be a partition of $N = [n^{3/4}, n]$. By Lemma 1, for $k = 1/16$ and the s -tuple $W := (|Q_1|, \dots, |Q_s|)$, there is i_0 such that, with probability at least

$$1 - \frac{s}{kr} = 1 - \frac{256}{c} \geq \frac{15}{16}$$

the number of queries Q in M that satisfy $|Q| \in N_{i_0}$ is at most k . Therefore, with probability at least $15/16$ there are no queries Q in M of size $|Q| \in N_{i_0}$. Let C be the event that there is no query Q in M of size $|Q| \in N_{i_0}$. Then

$$\Pr[\bar{C}] \leq \frac{1}{16}.$$

Let $d' = \Delta^{4i_0+2}$. For a random uniform set $I \subset X$ of size $d = d'$, with probability at least $3/4$, $A(\sigma, I)$ returns an integer in the interval $[d', 1.5d']$. For a random uniform set $I \subset X$ of size $d = 2d'$, with probability at least $3/4$, $A(\sigma, I)$ returns an integer in the interval $[2d', 3d']$. Since both intervals are disjoint, algorithm A , with success probability at least $3/4$, can distinguish between defective sets of size d' and $2d'$. We have constructed an algorithm, call it A' , that distinguishes, with success probability $3/4$, between defective sets of size d' and defective sets of size $2d'$. The probability that A' fails is at most $1/4$.

Let \mathcal{D} , \mathcal{D}' and $\{x_1, \dots, x_d\}$ be as in Lemma 2. Here $d \in \{d', 2d'\}$. Let B be the event that x_1, \dots, x_d are distinct. Since $i_0 \leq r$ we have

$$d \leq 2d' = 2\Delta^{4i_0+2} \leq 2\Delta^{4r+2} = 2n^{1/4} \log^2 n$$

and therefore, for large enough n ,

$$\Pr_{\mathcal{D}'}[\bar{B}] = 1 - \prod_{i=1}^{d-1} \left(1 - \frac{i}{n}\right) \leq \frac{d(d-1)}{2n} \leq \frac{2 \log^4 n}{n^{1/2}} \leq \frac{1}{16}.$$

Now partition the queries in M to three sets of queries $M_1 \cup M_2 \cup M_3$ where M_1 are the queries that contain at most n/Δ^{4i_0+4} items, M_2 are the queries that contains at least n/Δ^{4i_0} items and $M_3 = M \setminus (M_1 \cup M_2)$, i.e., M_3 are the queries Q that satisfies $|Q| \in N_{i_0}$. Let $A_1(I)$ be the event that for $I \subseteq X$ all the queries in M_1 give answer 0. Then

$$\begin{aligned} \Pr_{\mathcal{D}'}[\bar{A}_1] &= \Pr[(\exists Q \in M_1) Q \cap I \neq \emptyset] \\ &\leq s \Pr[Q \cap I \neq \emptyset | Q \in M_1] \\ &= s(1 - \Pr[Q \cap I = \emptyset | Q \in M_1]) \\ &\leq s \left(1 - \left(1 - \frac{1}{\Delta^{4i_0+4}}\right)^d\right) \\ &\leq \frac{sd}{\Delta^{4i_0+4}} = \frac{2}{c\Delta \log \Delta} \leq \frac{1}{16}. \end{aligned} \tag{1}$$

Then by Lemma 2, $\Pr_{\mathcal{D}}[\bar{A}_1] \leq 2/16$. Let $A_2(I)$ be the event that for $I \subseteq X$ all the queries in M_2 give answer 1. Then

$$\begin{aligned} \Pr_{\mathcal{D}}[\bar{A}_2] &= \Pr[(\exists Q \in M_2)Q \cap I = \emptyset] \\ &\leq s\Pr[Q \cap I = \emptyset | Q \in M_2] \\ &\leq s \left(1 - \frac{1}{\Delta^{4i_0}}\right)^d \\ &\leq se^{-\frac{d}{\Delta^{4i_0}}} = \frac{\Delta}{ce^{\Delta^2 \log \Delta}} \leq \frac{1}{16}. \end{aligned}$$

Thus, by Lemma 2, $\Pr_{\mathcal{D}}[\bar{A}_2] \leq 2/16$.

Now

$$\begin{aligned} \Pr[A' \text{ fails}] &\geq \Pr[A_1 \wedge A_2 \wedge C] \\ &= 1 - \Pr[\bar{A}_1 \vee \bar{A}_2 \vee \bar{C}] \\ &\geq 1 - \Pr[\bar{A}_1] - \Pr[\bar{A}_2] - \Pr[\bar{C}] \\ &\geq \frac{1}{2}. \end{aligned}$$

We got $\Pr[A' \text{ fails}] \geq 1/2$ which gives a contradiction. \blacktriangleleft

In the proof of Theorem 3, one cannot take smaller intervals for N_i (for example $[n/2^{4i+4}, n/2^{4i}]$). This is because, with the multiplicand s for the union bound in (1), the probability of \bar{A}_1 cannot then be bounded by $1/16$.

The proof is also true for estimating the number of defective items to within a factor $\lambda = \Theta(\log n)$. In fact, for such λ the lower bound is tight.

4 Lower Bound for Deterministic Algorithms

In this section we prove

► Theorem 4. *Let $c > 1$ be any constant. Any non-adaptive deterministic algorithm that c -estimates the number of defective items must ask at least $\Omega(n)$ queries.*

Proof. Let A be a non-adaptive deterministic algorithm that c -estimates the number of defective items. Let Q_1, \dots, Q_s be the queries that A asks. Let $d = n/2c$. For possible answers $a_1, \dots, a_s \in \{0, 1\}$ to the queries we define $S_{(a_1, \dots, a_s)}$, the set of all defective sets of size d that give the answers a_1, \dots, a_s to the queries Q_1, \dots, Q_s , respectively. That is, for every $I \in S_{(a_1, \dots, a_s)}$ we have $|I| = d$ and for every $i = 1, \dots, s$ we have $Q_i \cap I \neq \emptyset$ if $a_i = 1$ and $Q_i \cap I = \emptyset$ if $a_i = 0$. For $a = (a_1, \dots, a_s) \in \{0, 1\}^s$ let $I_a = \cup_{I \in S_a} I$. We now prove two claims:

▷ **Claim 5.** If the defective set is I_a then the algorithm gets the answers a to the queries.

Proof. If $Q_i \cap I_a \neq \emptyset$ then there is $I \in S_a$ such that $Q_i \cap I \neq \emptyset$ and then $a_i = 1$. If $Q_i \cap I_a = \emptyset$ then for every $I \in S_a$ we have $Q_i \cap I = \emptyset$ and then $a_i = 0$. \blacktriangleleft

▷ **Claim 6.** $|I_a| \leq cd$.

Proof. If $|I_a| > cd$ then the algorithm returns a value in $[cd + 1, c^2d]$ and then for the sets in S_a , that are of size d , this answer is not a c -estimation. A contradiction. \blacktriangleleft

2:6 Estimation of the Number of Defective Items

Since each $I \in S_a$ is of size d and is a subset of I_a we have

$$|S_a| \leq \binom{cd}{d}.$$

Since there are $\binom{n}{d}$ sets of size d we get

$$\binom{n}{d} = \sum_{a \in \{0,1\}^s} |S_a| \leq 2^s \binom{cd}{d}.$$

Since $d = n/(2c)$,

$$s \geq \log \binom{n}{\frac{n}{2c}} - \log \binom{\frac{n}{2}}{\frac{n}{2c}} = \Omega(n). \quad \blacktriangleleft$$

5 Upper Bounds

In this section is written for completeness. We use techniques similar to the ones in [7, 11] to prove

► **Theorem 7.** *Let c be any constant. There is a non-adaptive Monte Carlo randomized algorithm that asks*

$$s = O\left(\log \frac{1}{\delta} \log n\right)$$

queries and with probability at least $1 - \delta$, c -estimates the number of defective items.

We recall the Chernoff Bound.

► **Lemma 8 (Chernoff Bound).** *Let X_1, \dots, X_t be independent random variables that takes values in $\{0, 1\}$. Let $X = (X_1 + \dots + X_t)/t$ and $\mathbf{E}[X] \leq \mu$. Then for any $\Delta \geq \mu$*

$$\Pr[X \geq \Delta] \leq \left(\frac{e^{1-\frac{\mu}{\Delta}}}{\Delta}\right)^{\Delta t} \quad (2)$$

$$\leq \left(\frac{e\mu}{\Delta}\right)^{\Delta t}. \quad (3)$$

We will assume that $d \geq 6$. Otherwise, d can be estimated exactly in $O(\log n)$ more queries. Just run the algorithm that finds the defective items that asks $O(\log n)$ queries [19]. Here we give a 2-estimation algorithm. This can be extended in a straightforward manner to c -estimation for any constant c .

A p -query is a query Q that contains each item $i \in [n]$ randomly and independently with probability p . In the algorithm, $\mathcal{O}_I(Q) = 1$ if $Q \cap I \neq \emptyset$ and 0 otherwise.

Consider the following algorithm We now prove

► **Lemma 9.** *Let $|I| = d \geq 6$. If $u \leq d \leq w$ then with probability at least $1 - \delta$, $d \leq D \leq 2d$. The algorithm asks $O(\log(1/\delta) \log(w/u))$ queries.*

In particular, for $u = 1$ and $w = n$, the algorithm asks

$$O\left(\log \frac{1}{\delta} \log n\right)$$

queries.

■ **Algorithm 1** Estimate (u, w, δ) .

Input: u and w such that $u \leq d \leq w$ and a failure probability δ

Output: D such that w.p. at least $1 - \delta$, $d \leq D \leq 2d$.

1. For each $p_i = 1/(u \cdot 2^{i/4})$, $i = 0, 1, 2, 3, \dots, 8 \log(w/u)$,
2. For $t = O(\log(1/\delta))$ independent p_i -queries $Q_{i,1}, \dots, Q_{i,t}$ do:
3. $q_i = (\mathcal{O}_I(Q_{i,1}) + \dots + \mathcal{O}_I(Q_{i,t}))/t$.
4. Choose the first i_0 such that $q_{i_0} < 0.83$.
5. If no such i_0 exists then output (“ $d > w$ ”).
6. Otherwise output ($D := 2/p_{i_0}$).

Proof. Let i_1 be such that $p_{i_1-1} > 2/d$ and $p_{i_1} \leq 2/d$. Then for $j = 0, 1, \dots$,

$$2^{j/4}/d < p_{i_1+3-j} \leq 2^{(j+1)/4}/d.$$

For every i, j we have

$$\mu_i := \mathbf{E}[q_i] = \mathbf{E}[\mathcal{O}_I(Q_{i,j})] = \Pr[I \cap Q_{i,j} \neq \emptyset] = 1 - (1 - p_i)^d.$$

Since $d \geq 6$ we have $\mathbf{E}[q_{i_1+3}] = \mu_{i_1+3} \leq 1 - (1 - 2^{1/4}/d)^d \leq 0.74$ and

$$\begin{aligned} \Pr[D > 2d] &= \Pr[p_{i_0} < 1/d] = \Pr[i_0 > i_1 + 3] \\ &\leq \Pr[q_{i_1+3} \geq 0.83] \leq \delta/2. \end{aligned} \tag{4}$$

The first inequality in (4) follows from the fact that if $i_0 > i_1 + 3$ then $q_{i_1+3} \geq 0.83$. The second inequality follows from Chernoff bound (2) with $\mu = 0.74$ and $\Delta = 0.83$.

Now, since

$$\begin{aligned} \mathbf{E}[1 - q_{i_1+3-j}] &= 1 - \mu_{i_1+3-j} = (1 - p_{i_1+3-j})^d \\ &\leq e^{-p_{i_1+3-j}d} < e^{-2^{j/4}}, \end{aligned}$$

we have $\mathbf{E}[1 - q_{i_1-2}] \leq \mathbf{E}[1 - q_{i_1-1}] \leq 0.136$ and

$$\begin{aligned} \Pr[D < d] &= \Pr[p_{i_0} > 2/d] = \Pr[i_0 \leq i_1 - 1] \\ &= \sum_{i=0}^{i_1-1} \Pr[i_0 = i] \leq \sum_{i=0}^{i_1-1} \Pr[q_i < 0.83] \\ &= \sum_{i=0}^{i_1-3} \Pr[1 - q_i > 0.17] + \sum_{i=i_1-2}^{i_1-1} \Pr[1 - q_i > 0.17] \\ &\leq \sum_{i=0}^{i_1-3} \left(\frac{e \cdot e^{-2^{(i_1-i+3)/4}}}{0.17} \right)^{0.17 \cdot t} + \frac{\delta}{4} \\ &\leq \sum_{k=0}^{\infty} \left(0.95 \cdot e^{-2^{k/4}} \right)^{0.17 \cdot t} + \frac{\delta}{4} \leq \frac{\delta}{4} + \frac{\delta}{4} = \frac{\delta}{2}. \end{aligned} \tag{5}$$

In the first summand of (5) we use Chernoff bound (3). In the second summand we use Chernoff bound (2) for $\mu = 0.136$ and $\Delta = 0.17$. ◀

References

- 1 Nader H. Bshouty, Vivian E. Bshouty-Hurani, George Haddad, Thomas Hashem, Fadi Khoury, and Omar Sharafy. Adaptive Group Testing Algorithms to Estimate the Number of Defectives. *ALT*, 2017. [arXiv:1712.00615](#).
- 2 Chao L. Chen and William H. Swallow. Using Group Testing to Estimate a Proportion, and to Test the Binomial Model. *Biometrics.*, 46(4):1035–1046, 1990.
- 3 Yongxi Cheng and Yinfeng Xu. An efficient FPRAS type group testing procedure to approximate the number of defectives. *J. Comb. Optim.*, 27(2):302–314, 2014. [doi:10.1007/s10878-012-9516-5](#).
- 4 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](#).
- 5 Graham Cormode and S. Muthukrishnan. What’s hot and what’s not: tracking most frequent items dynamically. *ACM Trans. Database Syst.*, 30(1):249–278, 2005. [doi:10.1145/1061318.1061325](#).
- 6 Peter Damaschke and Azam Sheikh Muhammad. Bounds for Nonadaptive Group Tests to Estimate the Amount of Defectives. In *Combinatorial Optimization and Applications - 4th International Conference, COCOA 2010, Kailua-Kona, HI, USA, December 18-20, 2010, Proceedings, Part II*, pages 117–130, 2010. [doi:10.1007/978-3-642-17461-2_10](#).
- 7 Peter Damaschke and Azam Sheikh Muhammad. Competitive Group Testing and Learning Hidden Vertex Covers with Minimum Adaptivity. *Discrete Math., Alg. and Appl.*, 2(3):291–312, 2010. [doi:10.1142/S179383091000067X](#).
- 8 R. Dorfman. The detection of defective members of large populations. *Ann. Math. Statist.*, pages 436–440, 1943.
- 9 D. Du and F. K Hwang. Combinatorial group testing and its applications. *World Scientific Publishing Company.*, 2000.
- 10 D. Du and F. K Hwang. Pooling design and nonadaptive group testing: important tools for DNA sequencing. *World Scientific Publishing Company.*, 2006.
- 11 Moein Falahatgar, Ashkan Jafarpour, Alon Orlitsky, Venkatadheeraj Pichapati, and Ananda Theertha Suresh. Estimating the number of defectives with group testing. In *IEEE International Symposium on Information Theory, ISIT 2016, Barcelona, Spain, July 10-15, 2016*, pages 1376–1380, 2016. [doi:10.1109/ISIT.2016.7541524](#).
- 12 Edwin S. Hong and Richard E. Ladner. Group testing for image compression. *IEEE Trans. Image Processing*, 11(8):901–911, 2002. [doi:10.1109/TIP.2002.801124](#).
- 13 F. K. Hwang. A method for detecting all defective members in a population by group testing. *Journal of the American Statistical Association*, 67:605—608, 1972.
- 14 William H. Kautz and Richard C. Singleton. Nonrandom binary superimposed codes. *IEEE Trans. Information Theory*, 10(4):363–377, 1964. [doi:10.1109/TIT.1964.1053689](#).
- 15 Joseph L. Gastwirth and Patricia A. Hammick. Estimation of the prevalence of a rare disease, preserving the anonymity of the subjects by group testing: application to estimating the prevalence of aids antibodies in blood donors. *Journal of Statistical Planning and Inference.*, 22(1):15–27, 1989.
- 16 C. H. Li. A sequential method for screening experimental variables. *J. Amer. Statist. Assoc.*, 57:455–477, 1962.
- 17 Anthony J. Macula and Leonard J. Popyack. A group testing method for finding patterns in data. *Discrete Applied Mathematics*, 144(1-2):149–157, 2004. [doi:10.1016/j.dam.2003.07.009](#).
- 18 Hung Q. Ngo and Ding-Zhu Du. A survey on combinatorial group testing algorithms with applications to DNA Library Screening. In *Discrete Mathematical Problems with Medical Applications, Proceedings of a DIMACS Workshop, December 8-10, 1999*, pages 171–182, 1999. [doi:10.1090/dimacs/055/13](#).
- 19 Ely Porat and Amir Rothschild. Explicit Nonadaptive Combinatorial Group Testing Schemes. *IEEE Trans. Information Theory*, 57(12):7982–7989, 2011. [doi:10.1109/TIT.2011.2163296](#).

- 20 Dana Ron and Gilad Tsur. The Power of an Example: Hidden Set Size Approximation Using Group Queries and Conditional Sampling. *CoRR*, abs/1404.5568, 2014. [arXiv:1404.5568](#).
- 21 M. Sobel and P. A. Groll. Group testing to eliminate efficiently all defectives in a binomial sample. *Bell System Tech. J.*, 38:1179–1252, 1959.
- 22 William H. Swallow. Group Testing for Estimating Infection Rates and Probabilities of Disease Transmission. *Phytopathology*, 1985.
- 23 Keith H. Thompson. Estimation of the Proportion of Vectors in a Natural Population of Insects. *Biometrics*, 18(4):568–578, 1962.
- 24 S. D. Walter, S. W. Hildreth, and B. J. Beaty. Estimation of infection rates in population of organisms using pools of variable size. *Am J Epidemiol.*, 112(1):124–128, 1980.
- 25 Jack K. Wolf. Born again group testing: Multiaccess communications. *IEEE Trans. Information Theory*, 31(2):185–191, 1985. [doi:10.1109/TIT.1985.1057026](#).

A Polynomial-Delay Algorithm for Enumerating Connectors Under Various Connectivity Conditions

Kazuya Haraguchi¹

Otaru University of Commerce, Midori 3-5-21, Otaru, Hokkaido 047-8501, Japan
haraguchi@res.otaru-uc.ac.jp

Hiroshi Nagamochi

Graduate School of Informatics, Kyoto University, Yoshida-Honmachi, Sakyo-ku,
Kyoto 606-8501, Japan
nag@amp.i.kyoto-u.ac.jp

Abstract

We are given an instance (G, I, σ) with a graph $G = (V, E)$, a set I of items, and a function $\sigma : V \rightarrow 2^I$. For a subset X of V , let $G[X]$ denote the subgraph induced from G by X , and $I_\sigma(X)$ denote the common item set over X . A subset X of V such that $G[X]$ is connected is called a connector if, for any vertex $v \in V \setminus X$, $G[X \cup \{v\}]$ is not connected or $I_\sigma(X \cup \{v\})$ is a proper subset of $I_\sigma(X)$.

In this paper, we present the first polynomial-delay algorithm for enumerating all connectors. For this, we first extend the problem of enumerating connectors to a general setting so that the connectivity condition on X in G can be specified in a more flexible way. We next design a new algorithm for enumerating all solutions in the general setting, which leads to a polynomial-delay algorithm for enumerating all connectors for several connectivity conditions on X in G , such as the biconnectivity of $G[X]$ or the k -edge-connectivity among vertices in X in G .

2012 ACM Subject Classification Mathematics of computing → Graph enumeration

Keywords and phrases Graph with itemsets, Enumeration, Polynomial-delay algorithms, Connectors

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.3

Related Version The preprint appeared as Technical Report 2019-002, Department of Applied Mathematics and Physics, Kyoto University (<http://www.amp.i.kyoto-u.ac.jp/tecrep/>).

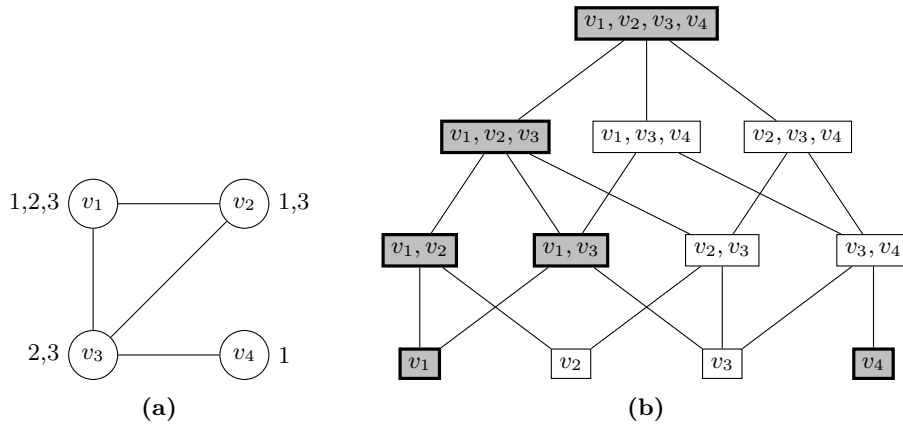
1 Introduction

In this paper, we consider enumeration of subgraphs in a given *attributed graph*, that is, a graph in which vertices are given items. The subgraphs should be connected, and at the same time, be maximal with respect to the common item set.

Formally, we are given an instance (G, I, σ) with a graph $G = (V, E)$, a set I of items, and a function $\sigma : V \rightarrow 2^I$. For a subset $X \subseteq V$, let $G[X]$ denote the subgraph induced from G by X , and $I_\sigma(X)$ denote the common item set $\bigcap_{u \in X} \sigma(u)$. A subset $X \subseteq V$ such that $G[X]$ is connected is called a *connector*, if for any vertex $v \in V \setminus X$, $G[X \cup \{v\}]$ is not connected or $I_\sigma(X \cup \{v\}) \subsetneq I_\sigma(X)$; i.e., there is no proper superset Y of X such that $G[Y]$ is connected and $I_\sigma(Y) = I_\sigma(X)$. We show a brief example of an instance in Figure 1(a). Note that we admit a connector whose common item set is empty. In the figure, it is $\{v_1, v_2, v_3, v_4\}$. If such a connector exists, then it is a connected component of the graph, but the converse does not necessarily hold.

¹ Corresponding author





■ **Figure 1** (a) An instance that has connectors $\{v_1\}$, $\{v_4\}$, $\{v_1, v_2\}$, $\{v_1, v_3\}$, $\{v_1, v_2, v_3\}$, and $\{v_1, v_2, v_3, v_4\}$, where an item is represented by an integer; (b) Hasse diagram of the transitive system (V, C_G) of the instance in (a), where solutions are indicated by shade.

We consider the problem of enumerating connectors. The problem is a generalization of the *frequent item set mining problem*, a well-known problem in data mining, such that G is a clique and a vertex corresponds to a transaction.

Let us introduce an application example of the problem from [11]. Suppose a biological network such that a vertex corresponds to a gene and an edge represents a protein-protein interaction between genes. A gene produces RNAs under a certain condition, and the phenomenon is called gene expression. A condition at which gene expression occurs is given to a vertex as an item. A biologist is particularly interested in a large-sized connector with a large common item set, that is, a large connected set of genes that make expressions simultaneously under common (possibly complex) conditions. Enumeration of connectors is a basic problem for discovering such meaningful gene sets.

Let us review related studies. For a usual graph (i.e., a non-attributed graph), there are some studies on enumeration of connected subgraphs. Avis and Fukuda [3] showed that all connected induced subgraphs are enumerable in output-polynomial time and in polynomial space, by means of reverse search. Nutov [7] showed that minimal undirected Steiner networks, and minimal k -connected and k -outconnected spanning subgraphs are enumerable in incremental polynomial time.

For an attributed graph, the *frequent subgraph mining problem* [5] is among significant graph mining issues. This problem asks to enumerate all subgraphs that appear in a given set of attributed graphs frequently, where the graph isomorphism is defined by taking into account the items. For the problem, gSpan [15] should be one of the most successful algorithms.

When it comes to the connector enumeration problem, Sese et al. [12] proposed the first algorithm, named COPINE, which explores the search space by utilizing the similar search tree as gSpan. Okuno et al. [9, 10] and Okuno [8] studied the parallelization of COPINE. Haraguchi et al. [4] proposed the first output-polynomial algorithm, named COOMA, which enumerates connectors by means of dynamic programming rather than a search tree.

In this paper, we present the first polynomial-delay algorithm for enumerating all connectors. For this, we first extend the problem of enumerating connectors to a general setting so that the connectivity condition on a vertex subset X in G can be specified in a more flexible way. Concretely, we introduce a new notion of a family of sets, called a “transitive system,” which is a generalization of the family of all vertex subsets that induce connected

subgraphs. The notion of connector is also extended to the transitive system and it will be called a solution. We then design a new algorithm for enumerating all solutions in the transitive system, which leads to a polynomial-delay algorithm for enumerating all connectors for several connectivity conditions on X in G , such as the biconnectivity of $G[X]$ or the k -edge-connectivity among vertices in X in G . The proposed algorithm enumerates the solutions by traversing a *family tree*. Traversal of a family tree is a frequently used technique in various enumeration algorithms (e.g., [6]).

The paper is organized as follows. After we make preparations in Section 2, we explain the structure of the family tree in Section 3. Then in Section 4, we provide an algorithm that enumerates all the solutions by traversing the family tree, which yields a polynomial-delay algorithm for the connector enumeration problem. In Section 5, we explain how we deal with various notions of edge- and vertex-connectivity in the enumeration algorithm, followed by concluding remarks in Section 6. For some proofs, details are included in the appendix.

2 Preliminaries

For two integers a and b , let $[a, b]$ denote the set of integers i with $a \leq i \leq b$. For two subsets $J = \{j_1, j_2, \dots, j_{|J|}\}$ and $K = \{k_1, k_2, \dots, k_{|K|}\}$ of a set A with a total order, where $j_1 < j_2 < \dots < j_{|J|}$ and $k_1 < k_2 < \dots < k_{|K|}$, we denote by $J \prec K$ if $J = \{k_i \mid 1 \leq i \leq j\}$ for some $j < |K|$ or the sequence $(j_1, j_2, \dots, j_{|J|})$ is lexicographically smaller than the sequence $(k_1, k_2, \dots, k_{|K|})$. We denote $J \preceq K$ if $J \prec K$ or $J = K$.

A system (V, \mathcal{C}) consists of a finite set V and a family $\mathcal{C} \subseteq 2^V$, where an element in V is called a *vertex*, and a set in \mathcal{C} is called a *component*. A system (V, \mathcal{C}) (or \mathcal{C}) is called *transitive* if any tuple of $Z, X, Y \in \mathcal{C}$ with $Z \subseteq X \cap Y$ implies $X \cup Y \in \mathcal{C}$. For a subset $X \subseteq V$, a component $Z \in \mathcal{C}$ with $Z \subseteq X$ is called *X -maximal* if no other component $W \in \mathcal{C}$ satisfies $Z \subsetneq W \subseteq X$. Let $\mathcal{C}_{\max}(X)$ denote the family of all X -maximal components.

For example, any Sperner family [13], a family of subsets such that none is contained in another subset, is a transitive system. Also the family \mathcal{C}_G of vertex subsets $X \in 2^V$ in a graph $G = (V, E)$ such that $G[X]$ is connected is transitive, where $G[X]$ with $|X| = 1$ (resp., $X = \emptyset$) is connected (resp., disconnected). We illustrate the Hasse diagram of a transitive system \mathcal{C}_G in Figure 1(b).

We define an instance to be a tuple $(V, \mathcal{C}, I, \sigma)$ of a set V of $n \geq 1$ vertices, a family $\mathcal{C} \subseteq 2^V$, a set I of $q \geq 1$ items and a function $\sigma : V \rightarrow 2^I$. For each subset $X \subseteq V$, let $I_\sigma(X) \subseteq I$ denote the common item set over $\sigma(v)$, $v \in X$; i.e., $I_\sigma(X) = \bigcap_{v \in X} \sigma(v)$. A *solution* is defined to be a component $X \in \mathcal{C}$ such that any component $Y \in \mathcal{C}$ with $Y \supsetneq X$ satisfies $I_\sigma(Y) \subsetneq I_\sigma(X)$. Let \mathcal{S} denote the family of all solutions to the instance. Our aim is to design an algorithm for enumerating all solutions in \mathcal{S} when \mathcal{C} is transitive. When an instance $(V, \mathcal{C}, I, \sigma)$ is given, we assume that \mathcal{C} is implicitly given as two oracles L_1 and L_2 such that

- given non-empty subsets $X \subseteq Y \subseteq V$, $L_1(X, Y)$ returns a component $Z \in \mathcal{C}_{\max}(Y)$ with $X \subseteq Z$ (or \emptyset if no such Z exists) in $\theta_{1,t}$ time and $\theta_{1,s}$ space; and
- given a non-empty subset $Y \subseteq V$, $L_2(Y)$ returns $\mathcal{C}_{\max}(Y)$ in $\theta_{2,t}$ time and $\theta_{2,s}$ space.

We also denote by $\delta(Y)$ an upper bound on $|\mathcal{C}_{\max}(Y)|$, where we assume that δ is a non-decreasing function in the sense that $\delta(X) \leq \delta(Y)$ if $X \subseteq Y$. For the example of family \mathcal{C}_G of vertex subsets X such that $G[X]$ is connected in a graph G with n vertices and m edges, we see that $\theta_{i,t} = O(n + m)$, $i = 1, 2$, $\theta_{i,s} = O(n + m)$, $i = 1, 2$, and $\delta(Y) = O(|Y|)$. We will show that the time delay of our algorithm is polynomial with respect to the input size, $\theta_{1,t}$, $\theta_{2,t}$ and $\delta(V)$.

3:4 Enumeration of Connectors

To facilitate our aim, we introduce a total order over the items in I by representing I as a set $[1, q] = \{1, 2, \dots, q\}$ of integers. For each subset $X \subseteq V$, let $\min I_\sigma(X) \in [0, q]$ denote the minimum item in $I_\sigma(X)$, where $\min I_\sigma(X) \triangleq 0$ for $I_\sigma(X) = \emptyset$. For each $i \in [0, q]$, define $\mathcal{S}_i \triangleq \{X \in \mathcal{S} \mid \min I_\sigma(X) = i\}$, where we see that \mathcal{S} is a disjoint union of \mathcal{S}_i , $i \in [0, q]$. We design an algorithm that enumerates all solutions in \mathcal{S}_k for any specified $k \in [0, q]$.

We observe an important property on a transitive family of components.

► **Lemma 1.** *Let (V, \mathcal{C}) be a transitive system. For a component $X \in \mathcal{C}$ and a superset $Y \supseteq X$, there is exactly one component $C \in \mathcal{C}_{\max}(Y)$ that contains X .*

Proof. Since $X \subseteq Y$, $\mathcal{C}_{\max}(Y)$ contains a Y -maximal component C that contains X . For any component $W \in \mathcal{C}$ with $W \neq C$ and $X \subseteq W \subseteq Y$, the transitivity of \mathcal{C} and $X \subseteq C \cap W$ imply $C \cup W \in \mathcal{C}$, where $C \cup W = C$ must hold by the Y -maximality of C . Hence C is unique. ◀

For a component $X \in \mathcal{C}$ and a superset $Y \supseteq X$, we denote by $C(X; Y)$ the component $C \in \mathcal{C}_{\max}(Y)$ that contains X .

3 Defining Family Tree

To generate all solutions in \mathcal{S} efficiently, we use the idea of family tree. Our tasks to establish an enumeration algorithm are as follows:

- Define the roots, called “bases,” over all solutions in \mathcal{S} (Section 3.1);
- Define the “parent” $\pi(S) \in \mathcal{S}$ of each non-base solution $S \in \mathcal{S}$, where S is called a “child” of $T = \pi(S)$ (Section 3.2);
- Design an algorithm A that, given $S \in \mathcal{S}$, returns $\pi(S)$ (Algorithm 1 in Section 3.2); and
- Design an algorithm B that, given a solution $T \in \mathcal{S}$, generates a set \mathcal{X} of components $X \in \mathcal{C}$ such that \mathcal{X} contains all children of T (Algorithm 2 in Section 3.3). For each component $X \in \mathcal{X}$, we construct $\pi(X)$ by algorithm A to see if X is a child of T (i.e., $\pi(X)$ is equal to T).

Starting from each base, we recursively generate the children of a solution. The complexity of delay-time of the entire algorithm is the time complexity of algorithms A and B, where $|\mathcal{X}|$ is bounded from above by the time complexity of algorithm B.

3.1 Defining Base

Let $(V, \mathcal{C}, I = [1, q], \sigma)$ be an instance on a transitive system. We define $V_{\langle 0 \rangle} \triangleq V$ and $V_{\langle i \rangle} \triangleq \{v \in V \mid i \in \sigma(v)\}$, $i \in I$. For each non-empty subset $J \subseteq I$, define $V_{\langle J \rangle} \triangleq \bigcap_{i \in J} V_{\langle i \rangle}$. For $J = \emptyset$, define $V_{\langle J \rangle} \triangleq V$. Define $\mathcal{B}_i \triangleq \{X \in \mathcal{C}_{\max}(V_{\langle i \rangle}) \mid \min I_\sigma(X) = i\}$ for each $i \in [0, q]$, and $\mathcal{B} \triangleq \bigcup_{i \in [0, q]} \mathcal{B}_i$. We call a component in \mathcal{B} a *base*.

► **Lemma 2.** *Let $(V, \mathcal{C}, I = [1, q], \sigma)$ be an instance on a transitive system.*

- (i) *For each non-empty set $J \subseteq [1, q]$ or $J = \{0\}$, it holds that $\mathcal{C}_{\max}(V_{\langle J \rangle}) \subseteq \mathcal{S}$;*
- (ii) *For each $i \in [0, q]$, a solution $S \in \mathcal{S}_i$ is contained in a base in \mathcal{B}_i ; and*
- (iii) *$\mathcal{S}_0 = \mathcal{B}_0$ and $\mathcal{S}_q = \mathcal{B}_q$.*

Proof.

- (i) Let X be a component in $\mathcal{C}_{\max}(V_{\langle J \rangle})$, where $J \subseteq I_\sigma(X)$. When $J = \{0\}$ (i.e., $V_{\langle J \rangle} = V$), no proper superset of X is a component, and X is a solution. Consider the case of $\emptyset \neq J \subseteq [1, q]$. To derive a contradiction, assume that X is not a solution; i.e., there is a proper superset Y of X such that $I_\sigma(Y) = I_\sigma(X)$. Since $\emptyset \neq J \subseteq I_\sigma(X) = I_\sigma(Y)$, we see that $V_{\langle J \rangle} \supseteq Y$. This, however, contradicts the $V_{\langle J \rangle}$ -maximality of X . This proves that X is a solution.

■ **Algorithm 1** PARENT(S): Finding the lex-min solution of a solution S .

Input : An instance $(V, \mathcal{C}, I = [1, q], \sigma)$ on a transitive system, an item $k \in [1, q - 1]$, and a non-base solution $S \in \mathcal{S}_k \setminus \mathcal{B}_k$, where $k = \min I_\sigma(S)$

Output: The lex-min solution $T \in \mathcal{S}_k$ of S

- 1 Let $\{k, i_1, i_2, \dots, i_p\} := I_\sigma(S)$, where $k < i_1 < i_2 < \dots < i_p$;
- 2 $J := \{k\}$; /* $C(S; k) \supseteq S$ by $S \notin \mathcal{B}_k$ */
- 3 **for** $j = 1, 2, \dots, p$ **do**
- 4 **if** $C(S; J \cup \{i_j\}) \neq S$ **then** $J := J \cup \{i_j\}$ /* $J = I_\sigma(T)$ holds */
- 5 **Return** $T := C(S; J)$

- (ii) We prove that each solution $S \in \mathcal{S}_i$ is contained in a base in \mathcal{B}_i , where $i = \min I_\sigma(S)$. By Lemma 1, S is a subset of the component $C(S; V_{(i)}) \in \mathcal{C}_{\max}(V_{(i)})$, where $I_\sigma(S) \supseteq I_\sigma(C(S; V_{(i)}))$. Since $i \in I_\sigma(C(S; V_{(i)}))$ for $i \geq 1$ (resp., $I_\sigma(C(S; V_{(i)})) = \emptyset$ for $i = 0$), we see that $\min I_\sigma(S) = i = \min I_\sigma(C(S; V_{(i)}))$. This proves that $C(S; V_{(i)})$ is a base in \mathcal{B}_i .
- (iii) Let $k \in \{0, q\}$. We see from (i) that $\mathcal{C}_{\max}(V_{(k)}) \subseteq \mathcal{S}$, which implies that $\mathcal{B}_k = \{X \in \mathcal{C}_{\max}(V_{(k)}) \mid \min I_\sigma(X) = k\} \subseteq \{X \in \mathcal{S} \mid \min I_\sigma(X) = k\} = \mathcal{S}_k$. We prove that any solution $S \in \mathcal{S}_k$ is a base in \mathcal{B}_k . By (ii), there is a base $X \in \mathcal{B}_k$ such that $S \subseteq X$, which implies that $I_\sigma(S) \supseteq I_\sigma(X)$, $\min I_\sigma(S) \leq \min I_\sigma(X)$. We see that $I_\sigma(S) = I_\sigma(X)$, since $\emptyset = I_\sigma(S) \supseteq I_\sigma(X)$ for $k = 0$, and $q = \min I_\sigma(S) \leq \min I_\sigma(X) \leq q$ for $k = q$. Hence $S \subseteq X$ would contradict that S is a solution. Therefore $S = X \in \mathcal{B}_k$, as required. ◀

By Lemma 2(iii), we can find all solutions in $\mathcal{S}_0 \cup \mathcal{S}_q$ by calling oracle $L_2(Y)$ for $Y = V_{(0)} = V$ and $Y = V_{(q)}$. In the following, we consider how to generate all solutions in \mathcal{S}_k with $1 \leq k \leq q - 1$.

For a notational convenience, we denote by $C(X; i)$ the component $C(X; V_{(i)})$ with $i \in I_\sigma(X)$ and by $C(X; J)$ the component $C(X; V_{(J)})$ with $J \subseteq I_\sigma(X)$.

► **Lemma 3.** *Let $(V, \mathcal{C}, I = [1, q], \sigma)$ be an instance on a transitive system. Let $S, T \in \mathcal{S}$ be solutions such that $S \subseteq T$. It holds that $T = C(S; I_\sigma(T))$.*

We omit the proof (Appendix A).

3.2 Defining Parent

This subsection defines the “parent” of a non-base solution. For two solutions $S, T \in \mathcal{S}$, we say that T is a *superset solution* of S if $T \supseteq S$ and $S, T \in \mathcal{S}_i$ for some $i \in [1, q - 1]$. A superset solution T of S is called *minimal* if no proper subset $Z \subsetneq T$ is a superset solution of S . Let S be a non-base solution in $\mathcal{S}_k \setminus \mathcal{B}_k$, $k \in [1, q - 1]$. We call a minimal superset solution T of S the *lex-min solution* of S if $I_\sigma(T) \preceq I_\sigma(T')$ for all minimal superset solutions T' of S . For example, in Figure 1(b), $\{v_1, v_2\}$, $\{v_1, v_3\}$, $\{v_1, v_2, v_3\}$ and $\{v_1, v_2, v_3, v_4\}$ are superset solutions of $\{v_1\}$, whereas $\{v_4\}$ has no superset solution. The solution $\{v_1\}$ has two minimal superset solutions, that is $\{v_1, v_2\}$ and $\{v_1, v_3\}$, where $\{v_1, v_2\}$ is its lex-min solution.

► **Lemma 4.** *Let $(V, \mathcal{C}, I = [1, q], \sigma)$ be an instance on a transitive system. For a non-base solution $S \in \mathcal{S}_k \setminus \mathcal{B}_k$ with $k \in [1, q - 1]$, let $I_\sigma(S) = \{k, i_1, i_2, \dots, i_p\}$, where $k < i_1 < i_2 < \dots < i_p$, and let T denote the lex-min solution of S .*

- (i) *For an integer $j \in [1, p]$, let $J = I_\sigma(T) \cap \{k, i_1, i_2, \dots, i_{j-1}\}$. Then $i_j \in I_\sigma(T)$ if and only if $C(S; J \cup \{i_j\}) \supseteq S$; and*
- (ii) *Given S , algorithm PARENT(S) in Algorithm 1 correctly delivers the lex-min solution of S in $O(q(n + \theta_{1,t}))$ time and $O(q + n + \theta_{1,s})$ space.*

Proof.

- (i) By Lemma 2(i) and $\min I_\sigma(S) = k$, we see that $C(S; J \cup \{i_j\}) \in \mathcal{S}_k$.
 Case 1. $C(S; J \cup \{i_j\}) = S$: For any set $J' \subseteq \{i_{j+1}, i_{j+2}, \dots, i_p\}$, the component $C(S; J \cup \{i_j\} \cup J')$ is equal to S and cannot be a minimal superset solution of S . This implies that $i_j \notin I_\sigma(T)$.
 Case 2. $C(S; J \cup \{i_j\}) \supseteq S$: Then $C = C(S; J \cup \{i_j\})$ is a solution by Lemma 2(i). Observe that $k \in J \cup \{i_j\} \subseteq I_\sigma(C) \subseteq I_\sigma(S)$ and $\min I_\sigma(C) = k$, implying that $C \in \mathcal{S}_k$ is a superset solution of S . Then C contains a minimal superset solution $T^* \in \mathcal{S}_k$ of S , where $I_\sigma(T^*) \cap [1, i_{j-1}] = I_\sigma(T^*) \cap \{k, i_1, i_2, \dots, i_{j-1}\} \supseteq J = I_\sigma(T) \cap \{k, i_1, i_2, \dots, i_{j-1}\} = I_\sigma(T) \cap [1, i_{j-1}]$ and $i_j \in I_\sigma(T^*)$. If $I_\sigma(T^*) \cap [1, i_{j-1}] \supsetneq J$ or $i_j \notin I_\sigma(T)$, then $I_\sigma(T^*) \prec I_\sigma(T)$ would hold, contradicting that T is the lex-min solution of S . Hence $I_\sigma(T) \cap [1, i_{j-1}] = J = I_\sigma(T^*) \cap [1, i_{j-1}]$ and $i_j \in I_\sigma(T)$.
- (ii) Based on (i), we can obtain the solution T as follows. First we find the item set $I_\sigma(T)$ by applying (i) to each $j \in [1, p]$, where we construct subsets $J_0 \subseteq J_1 \subseteq \dots \subseteq J_p \subseteq I_\sigma(S)$ such that $J_0 = \{k\}$ and

$$J_j = \begin{cases} J_{j-1} \cup \{i_j\} & \text{if } C(S; J_{j-1} \cup \{i_j\}) \supseteq S, \\ J_{j-1} & \text{otherwise.} \end{cases}$$

Each J_j can be obtained from J_{j-1} by testing whether $C(S; J_{j-1} \cup \{i_j\}) \supseteq S$ holds or not, where $C(S; J_{j-1} \cup \{i_j\})$ is computable by calling the oracle L_1 . By (i), we have $J_j = I_\sigma(T) \cap \{k, i_1, \dots, i_j\}$, and in particular, $J_p = I_\sigma(T)$ holds. Next we compute $C(S; J_p)$ by calling the oracle $L_1(S, V_{\langle J_p \rangle})$, where $C(S; J_p)$ is equal to the solution T by Lemma 3. The above algorithm is described as algorithm PARENT(S) in Algorithm 1. We omit the complexity analysis (Appendix B). \blacktriangleleft

For each non-base solution in $\mathcal{S}_k \setminus \mathcal{B}_k$, $k \in [1, q-1]$, the *parent* $\pi(S)$ of S is defined to be the lex-min solution of S . For a solution $T \in \mathcal{S}_k$, each non-base solution $S \in \mathcal{S}_k \setminus \mathcal{B}_k$ such that $\pi(S) = T$ is called a *child* of T .

3.3 Generating Children

This subsection shows how to construct a family \mathcal{X} of components so that all children of a solution T are included in \mathcal{X} .

► **Lemma 5.** *Let $(V, \mathcal{C}, I = [1, q], \sigma)$ be an instance on a transitive system. For an item $k \in [1, q-1]$, let $T \in \mathcal{S}_k$ be a solution.*

- (i) *For each child $S \in \mathcal{S}_k \setminus \mathcal{B}_k$ of T , it holds that $[k+1, q] \cap (I_\sigma(S) \setminus I_\sigma(T)) \neq \emptyset$ and $S \in \mathcal{C}_{\max}(T \cap V_{\langle j \rangle})$ for any $j \in [k+1, q] \cap (I_\sigma(S) \setminus I_\sigma(T))$.*
- (ii) *The set of all children of T can be constructed in $O(q\theta_{2,t} + q^2(n + \theta_{1,t})\delta(T))$ time and $O(q + n + \theta_{1,s} + \theta_{2,s})$ space.*

Proof.

- (i) Note that $[0, k] \cap I_\sigma(S) = [0, k] \cap I_\sigma(T) = \{k\}$ since $S, T \in \mathcal{S}_k$. Since $S \subseteq T$ are both solutions, $I_\sigma(S) \supseteq I_\sigma(T)$. Hence $[k+1, q] \cap (I_\sigma(S) \setminus I_\sigma(T)) \neq \emptyset$. Let $j \in [k+1, q] \cap (I_\sigma(S) \setminus I_\sigma(T))$. Since $S \subseteq T \cap V_{\langle j \rangle}$, there is a $(T \cap V_{\langle j \rangle})$ -maximal component $C \in \mathcal{C}_{\max}(T \cap V_{\langle j \rangle})$ with $S \subseteq C$, where $S \subseteq C \subseteq T$ and $I_\sigma(S) \supseteq I_\sigma(C) \supseteq I_\sigma(T)$. Then $k = \min I_\sigma(S) = \min I_\sigma(T)$ implies $\min I_\sigma(C) = k$.

We show that $C \in \mathcal{S}$, which implies $C \in \mathcal{S}_k$. Note that $j \in I_\sigma(C) \setminus I_\sigma(T)$, and $C \subsetneq T$. Assume that C is not a solution; i.e., there is a solution $C^* \in \mathcal{S}$ such that $C \subsetneq C^*$ and $I_\sigma(C) = I_\sigma(C^*)$, where $j \in I_\sigma(C) = I_\sigma(C^*)$ means that $C^* \subseteq V_{\langle j \rangle}$. Hence $C^* \setminus T \neq \emptyset$

■ **Algorithm 2** CHILDREN(T, k): Generating all children.

Input : An instance $(V, \mathcal{C}, I, \sigma)$, $k \in [1, q - 1]$ and a solution $T \in \mathcal{S}_k$
Output : All children of T , each of which is output whenever it is generated

```

1 for each  $j \in [k + 1, q] \setminus I_\sigma(T)$  do
2   Compute  $\mathcal{C}_{\max}(T \cap V_{(j)})$ ;
3   for each  $S \in \mathcal{C}_{\max}(T \cap V_{(j)})$  do
4     if  $k = \min I_\sigma(S)$  and  $j = \min\{i \mid i \in [k + 1, q] \cap (I_\sigma(S) \setminus I_\sigma(T))\}$  then
5       if  $T = \text{PARENT}(S)$  (i.e.,  $S$  is a child of  $T$ ) then
6         Output  $S$  as one of the children of  $T$ 

```

by the $(T \cap V_{(j)})$ -maximality of C . Since $C, C^*, T \in \mathcal{C}$ and $C \subseteq C^* \cap T$, we have $C^* \cup T \in \mathcal{C}$ by the transitivity. We also see that $I_\sigma(C^* \cup T) = I_\sigma(C^*) \cap I_\sigma(T) = I_\sigma(C) \cap I_\sigma(T) = I_\sigma(T)$. This, however, contradicts that T is a solution, proving that $C \in \mathcal{S}_k$. If $S \subsetneq C$, then $S \subsetneq C \subsetneq T$ would hold for $S, C, T \in \mathcal{S}_k$, contradicting that T is a minimal superset solution of S . Therefore $S = C$.

- (ii) By (i), the union of families $\mathcal{C}_{\max}(T \cap V_{(j)})$ with $j \in [k + 1, q] \setminus I_\sigma(T)$ contains all children of T . Whether a set S is a child of T or not can be tested by checking if $\text{PARENT}(S)$ is equal to T or not. However, for two items $j, j' \in [k + 1, q] \cap (I_\sigma(S) \setminus I_\sigma(T))$, the same child S can be generated from the different families $\mathcal{C}_{\max}(T \cap V_{(j)})$ and $\mathcal{C}_{\max}(T \cap V_{(j')})$. To avoid this, we output a child S of T when $S \in \mathcal{C}_{\max}(T \cap V_{(j)})$ for the minimum item j in the item set $[k + 1, q] \cap (I_\sigma(S) \setminus I_\sigma(T))$. In other words, we discard any set $S \in \mathcal{C}_{\max}(T \cap V_{(j)})$ if j is not the minimum item in $[k + 1, q] \cap (I_\sigma(S) \setminus I_\sigma(T))$. Algorithm 2 formally describes this procedure. We omit the complexity analysis (Appendix C). ◀

4 Traversing Family Tree

We are ready to describe an entire algorithm for enumerating solutions in \mathcal{S}_k for a given $k \in [0, q]$. We first compute $\mathcal{C}_{\max}(V_{(k)})$. We next compute the set \mathcal{B}_k ($\subseteq \mathcal{C}_{\max}(V_{(k)})$) of bases by testing whether $k = \min I_\sigma(T)$ or not, where $\mathcal{B}_k \subseteq \mathcal{S}_k$. When $k = 0$ or q , we are done with $\mathcal{B}_k = \mathcal{S}_k$ by Lemma 2(iii). Let $k \in [1, q - 1]$. Suppose that we are given a solution $T \in \mathcal{S}_k$, we find all the children of T by CHILDREN(T, k) in Algorithm 2. By applying Algorithm 2 to a newly found child recursively, we can find all solutions in \mathcal{S}_k .

When no child is found to a given solution $T \in \mathcal{S}_k$, we may need to go up to an ancestor by traversing recursive calls $O(n)$ times before we generate the next solution. This would result in $O(n\alpha)$ time delay, where α denotes the time complexity required for a single run of CHILDREN(T, k). To improve the delay to $O(\alpha)$, we employ the *alternative output method* [14], where we output the children of T after (resp., before) generating all descendants when the depth of the recursive call to T is an even (resp., odd) integer.

The entire enumeration algorithm is described in Algorithms 3 and 4. The following theorem summarizes the complexity of the enumeration algorithm. We omit the proof (Appendix D).

▶ **Theorem 6.** *Let $(V, \mathcal{C}, I = [1, q], \sigma)$ be an instance on a transitive system. For each $k \in [0, q]$, the set \mathcal{S}_k of solutions can be enumerated in $O(q\theta_{2,t} + q^2(n + \theta_{1,t})\delta(V_{(k)}))$ time delay and in $O((q + n + \theta_{1,s} + \theta_{2,s})n)$ space.*

■ **Algorithm 3** An algorithm to enumerate solutions in \mathcal{S}_k for a given $k \in [0, q]$.

Input : An instance $(V, \mathcal{C}, I = [1, q], \sigma)$ on a transitive system, and an item $k \in [0, q]$
Output : The set \mathcal{S}_k of solutions to $(V, \mathcal{C}, I, \sigma)$

- 1 Compute $\mathcal{C}_{\max}(V_{\langle k \rangle})$; $d := 1$;
- 2 **for each** $T \in \mathcal{C}_{\max}(V_{\langle k \rangle})$ **do**
- 3 **if** $k = \min I_\sigma(T)$ (i.e., $T \in \mathcal{B}_k$) **then**
- 4 Output T ;
- 5 **if** $k \in [1, q - 1]$ **then** DESCENDANTS($T, k, d + 1$)

■ **Algorithm 4** DESCENDANTS(T, k, d): Generating all descendants.

Input : An instance $(V, \mathcal{C}, I, \sigma)$, $k \in [1, q - 1]$, a solution $T \in \mathcal{S}_k$, and the current depth d of recursive call of DESCENDANTS
Output : All descendants of T in \mathcal{S}_k

- 1 **for each** $j \in [k + 1, q] \setminus I_\sigma(T)$ **do**
- 2 Compute $\mathcal{C}_{\max}(T \cap V_{\langle j \rangle})$;
- 3 **for each** $S \in \mathcal{C}_{\max}(T \cap V_{\langle j \rangle})$ **do**
- 4 **if** $k = \min I_\sigma(S)$ and $j = \min\{i \mid i \in [k + 1, q] \cap (I_\sigma(S) \setminus I_\sigma(T))\}$ **then**
- 5 **if** $T = \text{PARENT}(S)$ (i.e., S is a child of T) **then**
- 6 **if** d is odd **then**
- 7 Output S
- 8 DESCENDANTS($S, k, d + 1$);
- 9 **if** d is even **then**
- 10 Output S

It is worthwhile to mention that the enumeration task can be parallelized by running the algorithm for each item $k \in I$ independently.

For the connector enumeration problem, it is natural to assume that the item set $\sigma(v)$ is given as a list for each $v \in V$, and that every $i \in I$ appears in at least one list. Then the input size is $\Omega(n + m + q)$. Theorem 6 yields a strongly polynomial-delay algorithm for the connector enumeration problem as follows.

► **Theorem 7.** *Given an instance $(G = (V, E), I, \sigma)$, we can enumerate all connectors in $O(q^2(n + m)n)$ time delay and in $O((q + n + m)n)$ space, where $n = |V|$, $m = |E|$ and $q = |I|$.*

Proof. The connector enumeration problem for (G, I, σ) is solved by enumerating all solutions for the instance $(V, \mathcal{C}_G, I, \sigma)$, where \mathcal{C}_G denotes the family of connected components that was introduced in Section 2. For the transitive system (V, \mathcal{C}_G) , we see that $\theta_{i,t} = O(n + m)$, $i = 1, 2$, $\theta_{i,s} = O(n + m)$, $i = 1, 2$, and $\delta(Y) = O(|Y|) = O(n)$. By Theorem 6, we can enumerate all solutions in \mathcal{S} in $O(q^2(n + m)n)$ time delay and in $O((q + n + m)n)$ space. ◀

5 Connectors under Various Connectivity Conditions

We consider enumerating connectors under various connectivity conditions such as the edge- or vertex-connectivity. To treat this issue universally, we present a general method of constructing a transitive system from a graph and a weight function on elements in the graph. We assume that a given graph is undirected, but all the discussions can be extended to a mixed graph (i.e., a graph containing directed edges as well as undirected edges).

5.1 Transitive System Based on k -Connectivity

Let \mathbb{R}_+ denote the set of non-negative reals. For a function $f : A \rightarrow \mathbb{R}_+$ and a subset $B \subseteq A$, we let $f(B)$ denote $\sum_{a \in B} f(a)$.

We assume that the graph $G = (V, E)$ may have multiple edges but no self-loops. For two vertices $u, v \in V$, let $E(u, v)$ denote the set of edges between u and v . For two non-empty subsets $X, Y \subseteq V$, let $E(X; Y) \triangleq \bigcup_{u \in X, v \in Y} E(u, v)$. For two vertices $s, t \in V$, an s, t -cut C is defined to be an ordered pair (S, T) of disjoint subsets $S, T \subseteq V$ such that $s \in S$ and $t \in T$, and the element set $\varepsilon(C)$ of C ($\varepsilon(S, T)$ of (S, T)) is defined to be a union $F \cup R$ of the edge subset $F = E(S, T)$ and the vertex subset $R = V \setminus (S \cup T)$, where $R = \emptyset$ is allowed.

We define a *meta-weight function* on G to be $\omega : 2^V \times (V \cup E) \rightarrow \mathbb{R}_+$. For each subset $X \subseteq 2^V$, we denote $w(X, a)$, $a \in V \cup E$ as a function $\omega_X : V \cup E \rightarrow \mathbb{R}_+$ such that $\omega_X(a) = \omega(X, a)$ for each $a \in V \cup E$. We call ω *monotone* if, for any subsets $X \subseteq Y \subseteq V$, $\omega_Y(a) \geq \omega_X(a)$ holds for any $a \in V \cup E$.

For two vertices $s, t \in V$ and a subset $X \subseteq V$, define $\mu(s, t; X) \triangleq \min\{\omega_X(\varepsilon(C)) \mid s, t\text{-cuts } C = (S, T) \text{ in } G\}$. We call a vertex subset $X \subseteq V$ *k -connected* if $|X| = 1$ or $\mu(u, v; X) \geq k$ for each pair of vertices $u, v \in X$.

► **Lemma 8.** *Let (G, ω) be an undirected mixed graph with a monotone meta-weight function, and $k \geq 0$. For two k -connected subsets $X, Y \subseteq V$ such that $\omega_{X \cap Y}(X \cap Y) \geq k$, the subset $X \cup Y$ is k -connected.*

Proof. To derive a contradiction, assume that $X \cup Y$ is not k -connected; i.e., $|X \cup Y| \geq 2$ and some vertices $s, t \in X \cup Y$ admits an s, t -cut $C = (S, T)$ with $\omega_{X \cup Y}(\varepsilon(C)) < k$. By the monotonicity of ω , it holds that $\omega_{X \cup Y}(a) \geq \omega_X(a), \omega_Y(a)$ for any element $a \in V \cup E$. Hence $\omega_{X \cup Y}(\varepsilon(C)) < k$ implies $\omega_X(\varepsilon(C)) < k$ and $\omega_Y(\varepsilon(C)) < k$. Since each of X and Y is k -connected, we see that neither of $s, t \in X$ and $s, t \in Y$ occurs. Without loss of generality assume that $s \in X \setminus Y$ and $t \in Y \setminus X$. If a vertex $v \in X \cap Y$ belongs to T (resp., S), then C would be an s, v -cut with $s, v \in X$ (resp., v, t -cut with $v, t \in Y$), contradicting the k -connectivity of X (resp., Y). Hence for the set $R = V \setminus (S \cup T)$, it holds $X \cap Y \subseteq R$. By the assumption of $X \cap Y$, we have $k \leq \omega_{X \cap Y}(X \cap Y) \leq \omega_{X \cap Y}(R) \leq \omega_{X \cup Y}(R) \leq \omega_{X \cup Y}(\varepsilon(C))$. This, however, contradicts $\omega_{X \cup Y}(\varepsilon(C)) < k$. ◀

For a graph (G, ω) with a meta-weight function and a real $k \geq 0$, let $\mathcal{C}(G, \omega, k) \subseteq 2^V$ denote the family of k -connected subsets $X \subseteq V$ with $\omega_X(X) \geq k$.

► **Lemma 9.** *For an undirected graph (G, ω) with a monotone meta-weight function and a real $k \geq 0$, let $\mathcal{C} = \mathcal{C}(G, \omega, k)$. Then \mathcal{C} is transitive.*

Proof. Let $Z, X, Y \in \mathcal{C}$ such that $Z \subseteq X \cap Y$, where $\omega_{X \cup Y}(X \cup Y) \geq \omega_{X \cup Y}(Z) \geq \omega_Z(Z) \geq k$. By $\omega_Z(Z) \geq k$ and Lemma 8, $X \cup Y$ is k -connected. Since $\omega_{X \cup Y}(X \cup Y) \geq k$, it holds that $X \cup Y \in \mathcal{C}$. Therefore \mathcal{C} is transitive. ◀

5.2 Construction of Monotone Meta-weight Functions

For a graph $G = (V, E)$, let $w : V \cup E \rightarrow \mathbb{R}_+$ be a weight function. We define a *coefficient function* to be $\gamma = (\alpha, \beta)$ that consists of functions $\alpha : E \rightarrow \mathbb{R}_+$ and $\beta : V \cup E \rightarrow \mathbb{R}_+$. We call γ *monotone* if $1 \geq \alpha(e) \geq \beta(e)$ for each edge $e \in E$ and $1 \geq \beta(v)$ for each vertex $v \in V$. We call a tuple (G, w, γ) a *system*, and define a meta-weight function $\omega : 2^V \times (V \cup E) \rightarrow \mathbb{R}_+$ to the system so that, for each subset $X \subseteq V$, $\omega_X : V \cup E \rightarrow \mathbb{R}_+$ is given by

$$\omega_X(v) = \begin{cases} w(v) & \text{if } v \in X, \\ \beta(v)w(v) & \text{if } v \in V \setminus X, \end{cases} \quad \omega_X(e) = \begin{cases} w(e) & \text{if } e \in E(X; X), \\ \alpha(e)w(e) & \text{if } e \in E(X; V \setminus X), \\ \beta(e)w(e) & \text{if } e \in E(V \setminus X; V \setminus X). \end{cases}$$

We call a system (G, w, γ) *monotone* if γ is monotone.

► **Lemma 10.** *For a monotone system (G, w, γ) , the corresponding meta-weight function $\omega : 2^V \times (V \cup E) \rightarrow \mathbb{R}_+$ is monotone.*

We omit the proof (Appendix E).

For a system (G, w, γ) on a graph G with n vertices and m edges and a real $k \geq 0$, let $\tau(n, m, k)$ and $\sigma(n, m, k)$ denote the time and space complexities for testing if $\mu(u, v; X) < k$ holds or not for two vertices $u, v \in V$ and a subset $X \subseteq V$.

► **Lemma 11.** *For a monotone tuple (G, w, γ) , let ω be the corresponding monotone meta-weight function.*

- (i) $\tau(n, m, k) = O(mn \log n)$ and $\sigma(n, m, k) = O(n + m)$; and
- (ii) *Let $X \subseteq Y \subseteq V$ be non-empty subsets such that $\omega_X(X) \geq k$ and $\mu(u, u'; Y) \geq k$ for all vertices $u, u' \in X$. Given a vertex $t \in Y \setminus X$, whether there is a vertex $u \in X$ such that $\mu(u, t; Y) < k$ or not can be tested in $\tau(n, m, k)$ time and $\sigma(n, m, k)$ space.*

We omit the proof (Appendix F).

We denote by $\mathcal{C}(G, w, \gamma, k)$ the family of k -connected sets X with $\omega_X(X) \geq k$ in a system (G, w, γ) . We consider how to construct oracles L_1 and L_2 to the system. For two non-empty subsets $X \subseteq Y \subseteq V$, let $\mathcal{C}_{\max}(Y)$ denote the family of maximal subsets $Z \in \mathcal{C}(G, w, \gamma, k)$ such that $Z \subseteq Y$, and let $C_k(X; Y)$ denote a maximal set $X^* \in \mathcal{C}_{\max}(Y)$ such that $X \subseteq X^*$; and $C_k(X; Y) \triangleq \emptyset$ if no such set X^* exists.

► **Lemma 12.** *For a monotone system (G, w, γ) , let ω denote the corresponding monotone meta-weight function. Let $X \subseteq Y \subseteq V$ be non-empty subsets such that $\omega_X(X) \geq k$. Then*

- (i) $C_k(X; Y)$ is uniquely determined;
- (ii) *If there are vertices $u \in X$ and $v \in Y$ such that $\mu(u, v; Y) < k$, then $v \notin X^*$;*
- (iii) *Assume that $\mu(u, v; Y) \geq k$ for all vertices $u \in X$ and $v \in Y \setminus X$. Then $C_k(X; Y) = Y$ if $\mu(u, u'; Y) \geq k$ for all vertices $u, u' \in X$; and $C_k(X; Y) = \emptyset$ otherwise; and*
- (iv) *Finding $C_k(X; Y)$ can be done in $O(|Y|^2 \tau(n, m, k))$ time and $O(\sigma(n, m, k) + |Y|)$ space.*

Proof. We omit the proofs for (i) to (iii) (Appendix G). For (iv), we can find $C_k(X; Y)$ as follows. Based on (ii), we first remove the set Z of all vertices $v \in Y \setminus X$ such that $\mu(u, v; Y) < k$ for some vertex $u \in X$ so that $C_k(X; Y) = C_k(X; Y')$ for $Y' = Y \setminus Z$. For a fixed vertex $t \in Y \setminus X$, we can test if there is a vertex $u \in X$ such that $\mu(u, t; Y) < k$ or not in $O(\tau(n, m, k))$ time and $O(\sigma(n, m, k))$ space by Lemma 11(ii). Hence finding such a set Z takes $O(|Y \setminus X| \tau(n, m, k))$ time and $O(\sigma(n, m, k) + |Z|)$ space. We repeat the above procedure until there is no pair of vertices $u \in X$ and $v \in Y' \setminus X$ after executing at most $|Y \setminus X|$ repetitions taking $O(|Y \setminus X|^2 \tau(n, m, k))$ time and $O(\sigma(n, m, k) + |Y \setminus X|)$ space.

Based on (iii), we finally conclude that $C_k(X; Y) = Y'$ ($C_k(X; Y) = \emptyset$) if there is no pair of vertices $u, u' \in X$ such that $\mu(u, u'; Y') < k$ (resp., otherwise), which takes $O(|X|^2 \tau(n, m, k))$ time and $O(\sigma(n, m, k))$ space by Lemma 11(i).

An entire algorithm is described in Algorithm 5. The time and space complexities are then $O(|Y|^2 \tau(n, m, k))$ time and $O(\sigma(n, m, k) + |Y|)$, respectively. ◀

■ **Algorithm 5** $\text{MAXIMAL}(X; Y)$: Finding the maximal set in $\mathcal{C}(G, w, \gamma, k)$ that contains a specified set.

Input : A monotone system (G, w, γ) , a real $k \geq 0$, and non-empty subsets $X \subseteq Y \subseteq V$ such that $\omega_X(X) \geq k$

Output : $C_k(X; Y)$

- 1 $Y' := Y$;
- 2 **while** there are vertices $u \in X$ and $t \in Y' \setminus X$ such that $\mu(u, t; Y') < k$ **do**
- 3 $Z := \{t \in Y' \setminus X \mid \mu(u, t; Y') < k \text{ for some } u \in X\}$;
- 4 $Y' := Y' \setminus Z$
- 5 **if** $\mu(u, u'; Y') \geq k$ for all vertices $u, u' \in X$ **then**
- 6 Output Y' as $C_k(X; Y)$
- 7 **else**
- 8 Output \emptyset as $C_k(X; Y)$

By the lemma, oracle $L_1(X; Y)$ to a monotone system (G, w, γ) runs in $\theta_{1,t} = O(|Y|^2\tau(n, m, k))$ time and $\theta_{1,s} = O(\sigma(n, m, k) + |Y|)$ space.

For a system (G, w, γ) , we define a k -core of a subset $Y \subseteq V$ to be a subset Z of Y such that $\omega_Z(Z) \geq k$ and any proper subset Z' of Z satisfies $\omega_{Z'}(Z') < k$.

► **Lemma 13.** *Let (G, w, γ) be a monotone system, and Y be a subset of V . For the family \mathcal{K} of all k -cores of Y , it holds that $C_{\max}(Y) = \bigcup_{Z \in \mathcal{K}} \{C_k(Z; Y)\}$ and $|C_{\max}(Y)| \leq |\mathcal{K}|$. Given \mathcal{K} , $C_{\max}(Y)$ can be obtained in $O(|\mathcal{K}|(|Y|^2\tau(n, m, k) + |Y| \log |\mathcal{K}|))$ time and $O(\sigma(n, m, k) + |\mathcal{K}| \cdot |Y|)$ space.*

Proof. Clearly each set $X \in C_{\max}(Y)$ satisfies $\omega_X(X) \geq k$ and contains a k -core $Z \in \mathcal{K}$, where $C_k(Z; Y) \neq \emptyset$ and $C_k(Z; Y) = X$ holds by the uniqueness in Lemma 12(i). Therefore $C_{\max}(Y) = \bigcup_{Z \in \mathcal{K}} \{C_k(Z; Y)\}$, from which $|C_{\max}(Y)| \leq |\mathcal{K}|$ follows. Given \mathcal{K} , we compute $C_k(Z; Y)$ for each set $Z \in \mathcal{K}$ taking $O(|Y|^2\tau(n, m, k))$ time and $O(\sigma(n, m, k) + |Y|)$ space by Lemma 12(iv). We can test if the same set $X \in C_{\max}(Y)$ has been generated or not in $O(|Y| \log |\mathcal{K}|)$ time and $O(|\mathcal{K}| \cdot |Y|)$ space. Therefore \mathcal{X} can be constructed in $O(|\mathcal{K}|(|Y|^2\tau(n, m, k) + |Y| \log |\mathcal{K}|))$ time and $O(\sigma(n, m, k) + |\mathcal{K}| \cdot |Y|)$ space. ◀

By the lemma, oracle $L_2(Y)$ to a monotone system (G, w, γ) runs in $\theta_{2,t} = O(|\mathcal{K}|(|Y|^2\tau(n, m, k) + |Y| \log |\mathcal{K}|))$ time and $\theta_{2,s} = O(\sigma(n, m, k) + |\mathcal{K}| \cdot |Y|)$ space, where we assume that the family \mathcal{K} of k -cores of Y is given as input.

For $s, t \in V$, we denote by $\lambda(s, t; G)$ denote the minimum size $|F|$ of a subset $F \subseteq E$ so that the graph $G - F$ obtained from G by removing edges in F has no path between s and t . A graph G is called k -edge-connected if $|V(G)| \geq 1$ and $\lambda(u, v; G) \geq k$ for any two vertices $u, v \in X$. Below we describe how we enumerate connectors X such that $G[X]$ is k -edge-connected. We can apply our framework to enumeration of connectors under other connectivity conditions (e.g., k -vertex-connectivity) in the same way.

► **Theorem 14.** *Let (G, I, σ) be an instance and $k \geq 0$ be an integer, where $G = (V, E)$ is either a digraph or an undirected graph, $n = |V|$, $m = |E|$, and $q = |I|$. We can enumerate all connectors such that the induced subgraphs are k -edge-connected in $O(\min\{k + 1, n\}q^2n^3m)$ time delay and in $O(qn + n^3)$ space.*

Proof. Let (G, w, γ, k) be a system that consists of a graph G , a weight function w and a coefficient function $\gamma = (\alpha, \beta)$ such that $\alpha(e) := 0, e \in E(G)$, and $\beta(a) := 0, a \in V(G) \cup E(G)$. We see that γ is monotone and the family $\mathcal{C}(G, w, \gamma, k)$ is transitive by Lemmas 9 and 10. Set w so that $w(e) := 1, e \in E(G)$ and $w(v) := k, v \in V(G)$. Then we see that $\mathcal{C}(G, w, \gamma, k)$ is identical to the family of connectors X such that $G[X]$ is k -edge-connected.

Whether $\mu(s, t; X) \geq k$ or not can be tested in $O(\min\{k, n\}m)$ time [1, 2]. By Lemma 12(iv), $L_1(X; Y)$ runs in $O(|Y|^2 \min\{k + 1, n\}m)$ time and $O(n^2)$ space. The family \mathcal{K} of k -cores $Z \subseteq Y$ is $\{\{v\} \mid v \in Y\}$. By Lemma 13, $|\mathcal{C}_{\max}(Y)| \leq |\mathcal{K}| \leq |Y|$ and $L_2(Y)$ runs in $O(|Y|^3 \min\{k + 1, n\}m)$ time and $O(n^2)$ space.

By Lemma 12(iv) and Lemma 13, we have $\theta_{1,t} = O(\min\{k + 1, n\}n^2m)$, $\theta_{2,t} = O(\min\{k + 1, n\}n^3m)$, and $\theta_{1,s} = \theta_{2,s} = O(n^2)$, where we can set $\delta(Y) = n$ for any $Y \subseteq V$. The time delay and space complexity follow by Theorem 6. ◀

6 Concluding Remarks

In this paper, we proposed a polynomial delay algorithm for the connector enumeration problem. We treated the problem on what we call a transitive system and proposed an algorithm for enumerating all solutions in the system (Algorithms 3 and 4 in Section 4). We also presented how to treat connectors that satisfy various connectivity conditions.

References

- 1 Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Optimization*, volume 1 of *Handbooks in Management Science and Operations Research*, chapter Network Flows (IV), pages 211–369. North-Holland, 1989.
- 2 Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Englewood Cliffs, NJ, 1993.
- 3 David Avis and Komei Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65(1):21–46, 1996.
- 4 Kazuya Haraguchi, Yusuke Momoi, Aleksandar Shurbevski, and Hiroshi Nagamochi. COOMA: a components overlaid mining algorithm for enumerating connected subgraphs with common itemsets. *Journal of Graph Algorithms and Applications*, 23(2):434–458, 2019. doi:10.7155/jgaa.00497.
- 5 Akihiro Inokuchi, Takashi Washio, and Hiroshi Motoda. An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data. In Djamel A. Zighed, Jan Komorowski, and Jan Żytkow, editors, *Principles of Data Mining and Knowledge Discovery*, pages 13–23, 2000. doi:10.1007/3-540-45372-5_2.
- 6 Shin-ichi Nakano and Takeaki Uno. Efficient Generation of Rooted Trees. Technical Report NII-2003-005E, National Institute of Informatics, July 2003.
- 7 Zeev Nutov. Listing minimal edge-covers of intersecting families with applications to connectivity problems. *Discrete Applied Mathematics*, 157(1):112–117, 2009. doi:10.1016/j.dam.2008.04.026.
- 8 Shingo Okuno. *Parallelization of Graph Mining using Backtrack Search Algorithm*. PhD thesis, Kyoto University, 2017. doi:10.14989/doctor.k20518.
- 9 Shingo Okuno, Tasuku Hiraishi, Hiroshi Nakashima, Masahiro Yasugi, and Jun Sese. Parallelization of Extracting Connected Subgraphs with Common Itemsets. *Information and Media Technologies*, 9(3):233–250, 2014. doi:10.11185/imt.9.233.
- 10 Shingo Okuno, Tasuku Hiraishi, Hiroshi Nakashima, Masahiro Yasugi, and Jun Sese. Reducing Redundant Search in Parallel Graph Mining Using Exceptions. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 328–337, 2016. doi:10.1109/IPDPSW.2016.136.

- 11 Mio Seki and Jun Sese. Identification of active biological networks and common expression conditions. In *2008 8th IEEE International Conference on Bioinformatics and BioEngineering*, pages 1–6, 2008.
- 12 Jun Sese, Mio Seki, and Mutsumi Fukuzaki. Mining Networks with Shared Items. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management (CIKM '10)*, pages 1681–1684, 2010.
- 13 Emanuel Sperner. Ein Satz über Untermengen einer endlichen Menge. *Mathematische Zeitschrift*, 27(1):544–548, 1928. doi:10.1007/BF01171114.
- 14 Takeaki Uno. Two general methods to reduce delay and change of enumeration algorithms. Technical Report NII-2003-004E, National Institute of Informatics, April 2003.
- 15 Xifeng Yan and Jiawei Han. gSpan: Graph-based substructure pattern mining. In *Proceedings of 2002 IEEE International Conference on Data Mining (ICDM '02)*, pages 721–724, 2002.

A Proof of Lemma 3

Proof. Let $T' = C(S; I_\sigma(T)) \in \mathcal{C}_{\max}(V_{\langle I_\sigma(T) \rangle})$, where $S \subseteq T \subseteq V_{\langle I_\sigma(T) \rangle}$. The uniqueness of maximal component $T' = C(S; I_\sigma(T))$ by Lemma 1 indicates $T \subseteq T'$. To derive a contradiction, assume that $T \subsetneq T'$. By Lemma 2(i), $T' \in \mathcal{C}_{\max}(V_{\langle I_\sigma(T) \rangle})$ is a solution. Since T and T' are solutions with $T \subsetneq T'$, it must hold that $I_\sigma(T) \supsetneq I_\sigma(T')$, implying that $V_{\langle I_\sigma(T) \rangle} \not\supseteq T'$, a contradiction. Therefore we have $T = T'$. ◀

B Complexity Analysis of Lemma 4 (ii)

Proof. Let us mention critical parts in terms of time complexity analysis. In line 1, it takes $O(qn)$ time to compute $I_\sigma(S)$. The for-loop from line 3 to 4 is repeated $O(q)$ times. In line 4, the oracle $L_1(S, V_{\langle J \cup \{i_j\} \rangle})$ is called to obtain a component $Z = C(S; J \cup \{i_j\})$ and whether $S = Z$ or not is tested. This takes $O(\theta_{1,t} + n)$ time. The overall running time is $O(q(n + \theta_{1,t}))$. It takes $O(q)$ space to store $I_\sigma(S)$ and J , and $O(n)$ space to store S and Z . An additional $O(\theta_{1,s})$ space is needed for the oracle L_1 . ◀

C Complexity Analysis of Lemma 5 (ii)

Proof. Now we analyze the time and space complexities of the algorithm. Note that T may have no children. The outer for-loop is repeated $O(q)$ times. Computing $\mathcal{C}(T \cap V_{\langle j \rangle})$ in line 2 takes $\theta_{2,t}$ time by calling the oracle L_2 . The inner for-loop is repeated at most $\delta(T \cap V_{\langle j \rangle})$ times for each j , and the most time-consuming part of the inner for-loop is algorithm PARENT(S) in line 5, which takes $O(q(n + \theta_{1,t}))$ time by Lemma 4(ii). Recall that δ is a non-decreasing function. Then the running time of algorithm CHILDREN(T, k) is evaluated by

$$O\left(q\theta_{2,t} + q(n + \theta_{1,t}) \sum_{j \in [k+1, q] \setminus I_\sigma(T)} \delta(T \cap V_{\langle j \rangle})\right) = O(q\theta_{2,t} + q^2(n + \theta_{1,t})\delta(T)).$$

For the space complexity, we do not need to share the space between iterations of the outer for-loop. In each iteration, we use the oracle L_2 and algorithm PARENT(S), whose space complexity is $O(q + n + \theta_{1,s})$ by Lemma 4(ii). Then algorithm CHILDREN(T, k) uses $O(q + n + \theta_{1,s} + \theta_{2,s})$ space. ◀

D Proof of Theorem 6

Proof. First we analyze the time delay. Let α denote the time complexity required for a single run of $\text{CHILDREN}(T, k)$. By Lemma 5(ii) and $\delta(T) \leq \delta(V_{(k)})$, we have $\alpha = O(q\theta_{2,t} + q^2(n + \theta_{1,t})\delta(V_{(k)}))$. Hence we see that the time complexity of Algorithm 3 and DESCENDANTS without including recursive calls is $O(\alpha)$.

From Algorithm 3 and DESCENDANTS , we observe:

- (i) When d is odd, the solution S for any call $\text{DESCENDANTS}(S, k, d + 1)$ is output immediately before $\text{DESCENDANTS}(S, k, d + 1)$ is executed; and
- (ii) When d is even, the solution S for any call $\text{DESCENDANTS}(S, k, d + 1)$ is output immediately after $\text{DESCENDANTS}(S, k, d + 1)$ is executed.

Let m denote the number of all calls of DESCENDANTS during a whole execution of Algorithm 3. Let $d_1 = 1, d_2, \dots, d_m$ denote the sequence of depths d in each $\text{DESCENDANTS}(S, k, d + 1)$ of the m calls. Note that $d = d_i$ satisfies (i) when d_{i+1} is odd and $d_{i+1} = d_i + 1$, whereas $d = d_i$ satisfies (ii) when d_{i+1} is even and $d_{i+1} = d_i - 1$. Therefore we easily see that during three consecutive calls with depth d_i, d_{i+1} and d_{i+2} , at least one solution will be output. This implies that the time delay for outputting a solution is $O(\alpha)$.

We analyze the space complexity. Observe that the number of calls DESCENDANTS whose executions are not finished during an execution of Algorithm 3 is the depth d of the current call $\text{DESCENDANTS}(S, k, d + 1)$. In Algorithm 4, $|T| + d \leq n + 1$ holds initially, and $\text{DESCENDANTS}(S, k, d + 1)$ is called for a nonempty subset $S \subsetneq T$, where $|S| < |T|$. Hence $|S| + d \leq n + 1$ holds when $\text{DESCENDANTS}(S, k, d + 1)$ is called. Then Algorithm 3 can be implemented to run in $O(n\beta)$ space, where β denotes the space required for a single run of $\text{CHILDREN}(T, k)$. We have $\beta = O(q + n + \theta_{1,s} + \theta_{2,s})$ by Lemma 5(ii). Then the overall space complexity is $O((q + n + \theta_{1,s} + \theta_{2,s})n)$. ◀

E Proof of Lemma 10

Proof. Let $X \subseteq Y \subseteq V$. To prove $\omega_Y(A) \geq \omega_X(A)$ for any set $A \subseteq V \cup E$, it suffices to show that $\omega_Y(a) \geq \omega_X(a)$ for any element $a \in V \cup E$. For each vertex $v \in V$, we see that $\omega_Y(v) = \omega_X(v) + |\{v\} \cap (Y \setminus X)|(1 - \beta(v))w(v) \geq \omega_X(v)$. For each edge $e \in E$, we see that $\omega_Y(e) = \omega_X(e) + \Delta|V(e) \cap (Y \setminus X)|w(e) \geq \omega_X(e)$, where Δ is one of $1 - \alpha(e)$, $\alpha(e) - \beta(e)$, and $(1 - \beta(e))/2$. ◀

F Proof of Lemma 11

Proof.

- (i) The problem of computing $\mu(u, v; X)$ can be formulated as a problem of finding a maximum flow in a graph (G, ω_X) with an edge-capacity $\omega_X(e)$, $e \in E$ and a vertex-capacity $\omega_X(v)$, $v \in V$, and $\mu(u, v; X)$ can be computed in $O(mn \log n)$ time and $O(n + m)$ space by using the maximum flow algorithm [1, 2]. Hence $\tau(n, m, k) = O(mn \log n)$ and $\sigma(n, m, k) = O(n + m)$.
- (ii) Let $t \in Y \setminus X$. To find a vertex $u \in X$ with $\mu(u, t; Y) < k$ if any by using (i) only once, we augment the weighted graph (G, ω_Y) into (G^*, ω_Y) with a new vertex s^* and $|X|$ new directed edges $e_u = (s^*, u)$, $u \in X$ such that $\omega_Y(e_u) := k$. We denote by $V(G)$ and $V(G^*)$ the vertex sets of G and G^* , respectively. We claim that $\mu(s^*, t; Y) \geq k$ if and only if $\mu(u, t; Y) \geq k$, $\forall u \in X$.

First consider the case of $\mu(s^*, t; Y) < k$ in (G^*, ω_Y) ; i.e., (G^*, ω_Y) has an s^*, t -cut $C^* = (S, T)$ with $\omega_Y(\varepsilon(C^*)) < k$, where $s^* \in S$ and $t \in T$. Let $R = V(G^*) \setminus (S \cup T)$, where $R = V(G) \setminus (S \cup T)$. Note that $X \subseteq S \cup R$, since otherwise $u \in T \cap X$ would mean that $e_u = (s^*, u) \in E(S, T)$ and $\omega_Y(\varepsilon(C^*)) \geq \omega_Y(e_u) = k$, contradicting that $\omega_Y(\varepsilon(C^*)) < k$. Also $S \cap X \neq \emptyset$, since otherwise $X \subseteq R$ would mean that $\omega_Y(\varepsilon(C^*)) \geq \omega_Y(R) \geq \omega_X(X) \geq k$, contradicting that $\omega_Y(\varepsilon(C^*)) < k$. Let $u \in S \cap X$. Then $C = (S \setminus \{s^*\}, T)$ is a u, t -cut in (G, ω_Y) with $\omega_Y(\varepsilon(C)) \leq \omega_Y(\varepsilon(C^*)) < k$. This means that $\mu(u, t; Y) < k$.

Next consider the case of $\mu(s^*, t; Y) \geq k$ in (G^*, ω_Y) . In this case, we show that $\mu(u, t; Y) \geq k$ for all $u \in X$. To derive a contradiction, assume that $\mu(u, t; Y) < k$ for some vertex $u \in X$; i.e., (G, ω_Y) has a u, t -cut $C = (S, T)$ with $\omega_Y(\varepsilon(C)) < k$. Note that $T \cap X = \emptyset$, since otherwise $u' \in T \cap X$ would contradict the assumption that $\mu(u, u'; Y) \geq k$ holds for $u, u' \in X$. Then $C' = (S' = S \cup \{s^*\}, T)$ is an s^*, t -cut in (G^*, ω_Y) , and satisfies $\omega_Y(\varepsilon(C')) = \omega_Y(\varepsilon(C)) < k$ since $T \cap X = \emptyset$. This, however, contradicts that $\mu(s^*, t; Y) \geq k$ holds in (G^*, ω_Y) .

By the claim, it suffices to test if $\mu(s^*, t; Y) \geq k$ or not in $\tau(n, m, k)$ time and $\sigma(n, m, k)$ space. \blacktriangleleft

G Proof of Lemma 12

Proof.

- (i) To derive a contradiction, assume that there are two maximal sets $X_1, X_2 \in \mathcal{C}_{\max}(Y)$ such that $X \subseteq X_1 \cap X_2$. From this and the monotonicity of ω , it holds that $\omega_{X_1 \cup X_2}(X_1 \cup X_2) \geq \omega_{X_1 \cap X_2}(X_1 \cap X_2) \geq \omega_X(X) \geq k$. From this and Lemma 8, $X_1 \cup X_2$ is also k -connected and $X_1 \cup X_2 \in \mathcal{C}_{\max}(Y)$, contradicting the maximality of X_1 and X_2 . Therefore $C_k(X; Y)$ is unique.
- (ii) When $C_k(X; Y) = \emptyset$, $v \notin C_k(X; Y)$ is trivial. Assume that $C_k(X; Y) = X^* \in \mathcal{C}_{\max}(Y)$. By the monotonicity of ω and $X^* \subseteq Y$, it holds that $\mu(u, v; X^*) \leq \mu(u, v; Y) < k$. Hence $u, v \in X^*$ would contradict the k -connectivity of X^* . Since $u \in X^*$, we have $v \notin X^*$.
- (iii) Obviously if $\mu(u, u'; Y) < k$ for some vertices $u, u' \in X$, then no subset Y' of Y with $X \subseteq Y'$ can be k -connected, and $C_k(X; Y) = \emptyset$. Assume that $\mu(u, u'; Y) \geq k$ for all vertices $u, u' \in X$. By the monotonicity of ω and $X \subseteq Y$, it holds that $\omega_Y(Y) \geq \omega_X(X) \geq k$. To prove that $C_k(X; Y) = Y$, it suffices to show that $\mu(u, v; Y) \geq k$ for all pairs of vertices $u, v \in Y$. By assumption, $\mu(u, v; Y) \geq k$ for all vertices $u \in X$ and $v \in Y$. To derive a contradiction, assume that there is a pair of vertices $s, t \in Y \setminus X$ with $\mu(s, t; Y) < k$; i.e., there is an s, t -cut $C = (S, T)$ with $\omega_Y(\varepsilon(C)) < k$. Let $R = V \setminus (S \cup T)$. We observe that $X \subseteq R$, since $u \in X \cap S$ (resp., $u \in X \cap T$) would imply that C is a u, t -cut (resp., s, u -cut), contradicting that $\mu(u, v; Y) \geq k$ for all vertices $v \in Y \setminus X$. By the monotonicity of ω and $X \subseteq R$, it would hold that $k \leq \omega_X(X) \leq \omega_Y(R) \leq \omega_Y(\varepsilon(C)) < k$, a contradiction. \blacktriangleleft

Top Tree Compression of Tries

Philip Bille 

Technical University of Denmark, DTU Compute, Denmark
phbi@dtu.dk

Paweł Gawrychowski 

University of Wrocław, Poland
gawry@cs.uni.wroc.pl

Inge Li Gørtz 

Technical University of Denmark, DTU Compute, Denmark
inge@dtu.dk

Gad M. Landau 

University of Haifa, Israel
landau@cs.haifa.ac.il

Oren Weimann 

University of Haifa, Israel
oren@cs.haifa.ac.il

Abstract

We present a compressed representation of tries based on top tree compression [ICALP 2013] that works on a standard, comparison-based, pointer machine model of computation and supports efficient prefix search queries. Namely, we show how to preprocess a set of strings of total length n over an alphabet of size σ into a compressed data structure of worst-case optimal size $O(n/\log_{\sigma} n)$ that given a pattern string P of length m determines if P is a prefix of one of the strings in time $O(\min(m \log \sigma, m + \log n))$. We show that this query time is in fact optimal regardless of the size of the data structure.

Existing solutions either use $\Omega(n)$ space or rely on word RAM techniques, such as tabulation, hashing, address arithmetic, or word-level parallelism, and hence do not work on a pointer machine. Our result is the first solution on a pointer machine that achieves worst-case $o(n)$ space. Along the way, we develop several interesting data structures that work on a pointer machine and are of independent interest. These include an optimal data structures for random access to a grammar-compressed string and an optimal data structure for a variant of the level ancestor problem.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases pattern matching, tree compression, top trees, pointer machine

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.4

Related Version Draft of full version available at <https://arxiv.org/abs/1902.02187>.

Funding *Philip Bille*: Supported by the Danish Research Council (DFR – 4005-00267, DFR – 1323-00178).

Inge Li Gørtz: Supported by the Danish Research Council (DFR – 4005-00267, DFR – 1323-00178).

Oren Weimann: Supported by the Israel Science Foundation grant 592/17.

1 Introduction

A *string dictionary* compactly represents a set of strings $S = S_1, \dots, S_k$ to support efficient *prefix queries*, that is, given a pattern string P determine if P is a prefix of some string in S . Designing efficient string dictionaries is a fundamental data structural problem dating back to the 1960's. String dictionaries are a key component in a wide range of applications in areas such as computational biology, data compression, data mining, information retrieval, natural language processing, and pattern matching.



© Philip Bille, Paweł Gawrychowski, Inge Li Gørtz, Gad M. Landau, and Oren Weimann;
licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 4; pp. 4:1–4:18

Leibniz International Proceedings in Informatics



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

A key challenge and the focus of most of the recent work is to design efficient *compressed string dictionaries*, that take advantage of repetitions in the strings to minimize space, while still supporting efficient queries. While many efficient solutions are known, they all rely on powerful word-RAM techniques, such as tabulation, address arithmetic, word-level parallelism, hashing, etc., to achieve efficient bounds. A natural question is whether or not such techniques are necessary for obtaining efficient compressed string dictionaries or if simpler and more basic computational primitives such as pointer-based data structures and character comparison suffice.

In this paper, we answer this question to the affirmative by introducing a new compressed string dictionary based on *top tree compression* that works on a standard comparison-based, pointer machine model of computation. We achieve the following bounds: let $n = \sum_{i=1}^k |S_i|$ be the total length of the strings in S , let σ be the size of the alphabet, and m be the length of a query string P . Our compressed string dictionary uses $O(n/\log_\sigma n)$ space (space is measured as the number of words and not bits, see discussion below) and supports queries in $O(\min(m \log \sigma, m + \log n))$ time. The space matches the information-theoretic worst-case space lower bound, and we further show that the query time is optimal for any comparison-based query algorithm regardless of the space. Compared to previous work our string dictionary is the first $o(n)$ space solution in this model of computation.

1.1 Computational Models

We consider three computational models. In the *comparison-based model* algorithms only interact with the input by comparing elements. Hence they cannot exploit the internal representation of input elements, e.g., for hashing or word-level parallelism. The comparison-based model is a fundamental and well-studied computational model, e.g., in textbook results for sorting [44], string matching [43], and computational geometry [53]. Modern programming languages and libraries, such as the C++ standard template library, implement comparison-based algorithms by supporting abstract and user-specified comparison functions as function arguments. In our context, we say that a string dictionary is comparison-based if the query algorithm can only access the input string P via single character comparisons of the form $P[i] \leq c$, where c is a character.

In the *pointer machine model*, a data structure is a directed graph with bounded out-degree. Each node contains a constant number of data fields or pointer to other nodes and algorithms must access the data structure by traversing the graph. Hence, a pointer machine algorithm cannot implement random access structures such as arrays or perform address arithmetic. The pointer machine captures linked data structures such as linked-lists and search trees. The pointer machine model is a classic and well-studied model, see e.g. [1, 20, 21, 36, 59].

Finally, in the word RAM model of computation [35] the memory is an array of memory words, that each contain a logarithmic number of bits. Memory words can be operated on in unit-time using a standard set of arithmetic operations, boolean operations, and shifts. The word RAM model is strictly more powerful than the comparison-based model and the pointer-machine model and supports random access, hashing, address arithmetic, word-level parallelism, etc. (these are not possible in the other models).

The space of a data structure in the word RAM model is the number of memory words used and the space in the pointer machine model is the total number of nodes. To compare the space of the models, we assume that each field in a node in the pointer machine stores a logarithmic number of bits. Hence, the total number of bits we can represent in a given space in both models is within a constant factor of each other.

1.2 Previous work

The classic textbook string dictionary solution, due to Fredkin [30] from 1960, is to store the *trie* T of the strings in S and to answer prefix queries using a top-down traversal of T , where at each step we match a single character from P to the labels of the outgoing edges of a node. If we manage to match all characters of P then P is a prefix of a string in S and otherwise it is not.

Depending on the representation of the trie and the model of computation we can obtain several combinations of space and time complexity. On a comparison-based, pointer machine model of computation, we can store the outgoing edges of each in a biased search tree [14], leading to an $O(n)$ space solution with query time $O(\min(m \log \sigma, m + \log n))$.

We can compress this solution by merging maximal identical complete subtrees of T [27], thus replacing T by a directed acyclic graph (DAG) D that represents T . This leads to a solution with the same query time as above but using only $O(d)$ space, where d is the size of the smallest DAG D representing T . The size of D can be exponentially smaller than n , but may not compress at all. Consider for instance the case where T is a single path of length n where all edges have the same label (i.e., corresponding to a single string of the same letter). Even though T is highly compressible (we can represent it by the label and the length of the path) it does not contain any identical subtrees and hence its smallest DAG has size $\Omega(n)$.

Using the power of the word RAM model improved representations are possible. Benoit et al. [13] and Raman et al. [54] gave *succinct* representations of tries that achieve $O(n/\log_\sigma n)$ space and $O(m)$ query time, thus simultaneously achieving optimal query time and matching the worst-case information theoretic space lower bounds. These results rely on powerful word RAM techniques to obtain the bounds, such as tabulation and hashing. Numerous trie representations are known, see e.g., [4, 5, 6, 7, 8, 17, 25, 33, 39, 40, 52, 58, 60, 61, 62], but these all use word RAM techniques to achieve near optimal combinations of time and space.

Another approach is to compress the strings according to various measures of repetitiveness, such as the empirical k -th order entropy [34, 45, 49, 55], the size of the Lempel-Ziv parse [9, 15, 22, 31, 32, 41, 51], the size of the smallest grammar [23, 24, 31], the run-length encoded Burrows-Wheeler transform, [46, 47, 48, 56], and others [5, 10, 11, 29, 50, 57]. The above solutions are designed to support more general queries on the strings, but as noted by Ars and Fischer [5] they are straightforward to adapt to prefix queries. For example, if z is size of the Lempel-Ziv parse of the concatenation of the strings in S , the result of Christiansen and Etienne [22] implies a string dictionary of size $O(z \log(n/z))$ that supports queries in time $O(m + \log^\epsilon n)$. Since z can be exponentially smaller than n , the space is significantly improved on highly-compressible strings. Since $z = O(n/\log_\sigma n)$ in the worst-case, the space is always $O(\frac{n}{\log_\sigma n} \log(\frac{n}{\log_\sigma n})) = O(\frac{n \log \log_\sigma n}{\log_\sigma n})$ and thus almost optimal compared to the information theoretic lower bound. Similar bounds are known for the other measures of repetitiveness. As in the case of succinct representations of tries, all of these solutions use word RAM techniques.

1.3 Our results

We propose a new compressed string dictionary that achieves the following bounds:

► **Theorem 1.** *Let S be a set of strings of total length n over an alphabet of size σ . On a comparison-based, pointer machine model of computation, we can construct a compressed string dictionary that uses $O(n/\log_\sigma n)$ space and answer queries in $O(\min(m \log \sigma, m + \log n))$ time.*

Note that the space bound for Theorem 1 matches the information theoretic lower bound and the time bound matches the classic linear space implementation of tries with biased search trees. The result is the first $o(n)$ space solution in this model of computation. Furthermore, we show that this time bound is optimal.

► **Theorem 2.** *For any n , $m \leq n$, and $\sigma \geq 2$, there exists a set S of strings of total length n over an alphabet of size σ such that any comparison-based algorithm that checks if a given pattern P of length m belongs to S needs to perform $\Omega(\min(m \log \sigma, m + \log n))$ comparisons in the worst case.*

Note that Theorem 2 holds regardless of the space used, holds even for weaker membership queries, and only assumes that the algorithm is a comparison-based algorithm. We note that the upper bound holds on a pointer machine with comparisons and additions as arithmetic operations, while the lower bound only assumes comparisons.

1.4 Techniques

In *top tree compression* [18] one transforms a labeled tree T into another tree \mathcal{T} (called a *top tree*) that is of height $O(\log n)$ and represents a hierarchical decomposition of T into connected subgraphs (called *clusters*). Each cluster overlaps with other clusters in at most two nodes. Every leaf in \mathcal{T} corresponds to a cluster consisting of a single edge in T and every internal node in \mathcal{T} corresponds to a merge of two clusters. The top tree \mathcal{T} is then compressed using the classical DAG compression resulting in the *top DAG* \mathcal{TD} . The top DAG supports basic navigational queries on T in $O(\log n)$ time, has size $O(n/\log_\sigma n)$, can compress exponentially better than DAG compression, and is never worse than DAG compression by more than a $O(\log n)$ factor [16, 18, 28, 38].

Our main technical contribution is implementing prefix search optimally on the top DAG. To this end, we develop several optimal pointer machine data structures of independent interest:

- A data structure for the *path extraction problem*, that asks to compactly represent an edge-labeled tree T such that given a node v we can efficiently return the labels on the root-to- v path in T . While an optimal solution for this problem can be obtained by plugging in known tools, more specifically a fully persistent queue [37], we believe that our self-contained solution is simpler and elegant.
- A data structure for the *weighted level ancestor* problem, that asks to compactly represent an edge-weighted tree T such that given a node v and a positive number x we can efficiently return the rootmost ancestor of v whose distance from the root is at least x . An immediate implication of our weighted level ancestor data structure is an optimal data structure for the *random access problem* on grammar compressed strings. This improves a SODA'11 result [19] that required word RAM bit tricks.
- A data structure for the *spine path extraction problem*, that asks to compactly represent a top-tree compression \mathcal{TD} such that given a cluster C we can efficiently return the characters of the unique path between the two boundary nodes of C .
- For the lower bound, we show that any algorithm that given a string $P[1, m]$ checks if $\sum_{i=1}^m P[i] = 0 \pmod{2}$ needs to perform $\Omega(m \log \sigma)$ comparisons in the worst case. We then show that when $n \geq m\sigma^m$ this implies the $\Omega(m \log \sigma)$ bound for our problem and when $n < m\sigma^m$ it implies the $\Omega(m + \log n)$ bound for our problem.

1.5 Roadmap

In Section 2 we recall *top trees* and how a top tree of a tree T is obtained by merging (either vertically or horizontally) the top trees of two subtrees of T that overlap on a single node. In Section 3 we present a simple randomized Monte-Carlo word RAM solution to the compressed string indexing problem that is the basis of our deterministic pointer machine solutions in the following sections. The solution is based on top trees and efficiently handles horizontal merges (deterministically) and vertical merges (randomized Monte-Carlo). In Section 4 we show how to handle vertical merges deterministically on a pointer machine, and in Section 5 we show that this suffices to achieve the $O(m + \log n)$ query time in Theorem 1. We show a different way to handle vertical merges in Section 6 and horizontal merges in Section 7. In Section 8 we show that these suffice to achieve the $O(m \log \sigma)$ query time in Theorem 1. Due to space constraints we defer the details of the lower bound and all proofs to the full version of the paper.

2 Preliminaries

In this section we briefly review Karp-Rabin fingerprints [42], top trees [3], and top tree compression [18].

2.1 Karp-Rabin Fingerprints

The Karp-Rabin fingerprint [42] of a string x is defined as $\phi(x) = \sum_{i=1}^{|x|} x[i] \cdot c^i \bmod p$, where c is a randomly chosen positive integer, and $2N^{c+4} \leq p \leq 4N^{c+4}$ is a prime. Karp-Rabin fingerprints guarantee that given two strings x and y , if $x = y$ then $\phi(x) = \phi(y)$. Furthermore, if $x \neq y$, then with high probability $\phi(x) \neq \phi(y)$. Fingerprints can be composed and subtracted as follows.

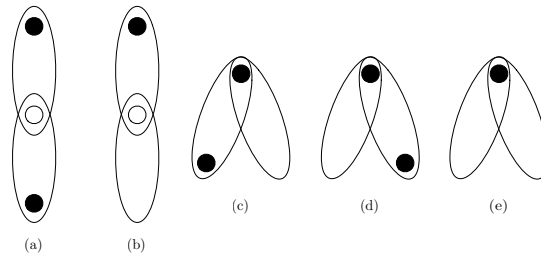
► **Lemma 3.** *Let $x = yz$ be a string decomposable into a prefix y and suffix z . Given any two of the Karp-Rabin fingerprints $\phi(x)$, $\phi(y)$ and $\phi(z)$, it is possible to calculate the remaining fingerprint in constant time.*

2.2 Clustering

Let v be a node in T with children v_1, \dots, v_k in left-to-right order. Define $T(v)$ to be the subtree induced by v and all proper descendants of v . Define $F(v)$ to be the forest induced by all proper descendants of v . For $1 \leq s \leq r \leq k$ let $T(v, v_s, v_r)$ be the connected component induced by the nodes $\{v\} \cup T(v_s) \cup T(v_{s+1}) \cup \dots \cup T(v_r)$.

A *cluster* with *top boundary node* v is a connected component of the form $T(v, v_s, v_r)$, $1 \leq s \leq r \leq k$. A *cluster* with *top boundary node* v and *bottom boundary node* u is a connected component of the form $T(v, v_s, v_r) \setminus F(u)$, $1 \leq s \leq r \leq k$, where u is a node in $T(v_s) \cup \dots \cup T(v_r)$. We denote the top boundary node of a cluster C by $\text{top}(C)$. Clusters can therefore have either one or two boundary nodes. For example, let $p(v)$ denote the parent of v then a single edge $(v, p(v))$ of T is a cluster where $p(v)$ is the top boundary node. If v is a leaf then there is no bottom boundary node, otherwise v is a bottom boundary node. Nodes that are not boundary nodes are called *internal nodes*. The path between the top and bottom boundary nodes in a cluster C is called the cluster's *spine*, and the string obtained by concatenating the labels on the spine from top to bottom is denoted $\text{spine}(C)$.

Two edge disjoint clusters A and B whose vertices overlap on a single boundary node can be *merged* if their union $C = A \cup B$ is also a cluster. There are five ways of merging clusters (see Figure 1). Merges of type (a) and (b) are called *vertical merges* (C is then a *vertical*



■ **Figure 1** Five ways of merging clusters. The \bullet nodes are boundary nodes that remain boundary nodes in the merged cluster. The \circ nodes are boundary nodes that become internal (non-boundary) nodes in the merged cluster. Note that in the last four merges at least one of the merged clusters has a top boundary node but no bottom boundary node.

cluster) and can be done only if the common boundary node is not a boundary node of any other cluster except A and B . Merges of type (c), (d), and (e) are called *horizontal merges* (C is then a *horizontal cluster*) and can be done only if at least one of A or B does not have a bottom boundary node.

2.3 Top Trees

A *top tree* \mathcal{T} of T is a hierarchical decomposition of T into clusters. It is an ordered, rooted, labeled, and binary tree defined as follows (see Figure 2(a)–(c)).

- The nodes of \mathcal{T} correspond to clusters of T .
- The root of \mathcal{T} corresponds to the cluster T itself. The top boundary node of the root of \mathcal{T} is the root of T .
- The leaves of \mathcal{T} correspond to the edges of T . The label of each leaf is the label of the corresponding edge (u, v) in T .
- Each internal node of \mathcal{T} corresponds to the merged cluster of its two children. The label of each internal node is the type of merge it represents (out of the five merging options). The children are ordered so that the left child is the child cluster visited first in a preorder traversal of T .

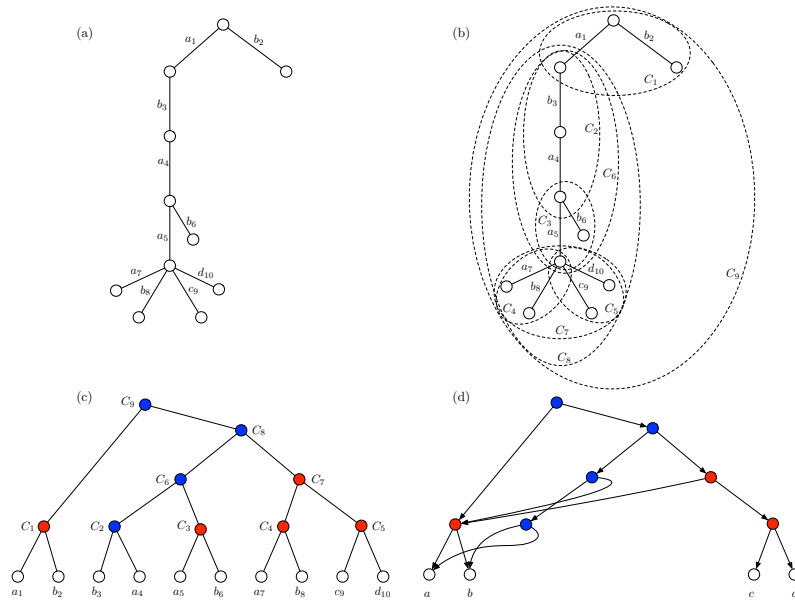
► **Lemma 4** (Alstrup et al. [3]). *Given a tree T of size n_T , we can construct in $O(n_T)$ time a top tree \mathcal{T} of T that is of size $O(n_T)$ and height $O(\log n_T)$.*

2.4 Top Dags

Every labeled tree can be represented with a directed acyclic graph (DAG) by identifying identical rooted subtrees and replacing them with a single copy. The *top DAG* of T , denoted \mathcal{TD} , is the minimal DAG representation of the top tree \mathcal{T} of T . We can compute it in $O(n_{\mathcal{T}})$ time from \mathcal{T} [27]¹. Top DAGs have important properties for compression and computation [16, 18, 28, 38]. We need the following optimal worst-case compression bound.

► **Lemma 5** (Dudek and Gawrychowski [28]). *Given an ordered tree with n_T nodes over an alphabet of size σ , we can construct a top DAG \mathcal{TD} in $O(n_T)$ time of size $n_{\mathcal{TD}} = O(n_T / \log_{\sigma} n_T)$.*

¹ Here we use edge labels instead of nodes label. The two definitions are equivalent and edge labels are more natural for tries.



■ **Figure 2** (a) A trie. Each edge label has a subscript to identify the corresponding leaf in the top tree in (c). (b) A hierarchical clustering of (a). (c) The top tree corresponding to (a). Blue nodes are vertical clusters and red nodes are horizontal clusters. (d) The top DAG of (c).

3 A Simple Index

We first present a simple randomized Monte-Carlo word RAM string index, that will be the starting point for our deterministic, comparison-based pointer machine solution in the later sections.

3.1 Data Structure

Let T be the trie of the strings $S = S_1, \dots, S_k$ and let \mathcal{TD} be the corresponding top DAG of T . Our data structure augments \mathcal{TD} with additional information. For each cluster C in \mathcal{TD} we store the following information.

- If C is a leaf cluster representing an edge e , we store the label of e .
- If C is an internal cluster with left and right child A and B , we store the label of the edge to the *rightmost child* of the top boundary node, the fingerprint $\phi(\text{spine}(C))$, and the length $|\text{spine}(C)|$.

This requires constant space for each cluster and hence $O(n_{\mathcal{TD}})$ space in total.

3.2 Searching

Given a pattern P of length m , we denote the unique node in T whose path from the root matches the longest prefix of P , the

Given a pattern P of length m we find the longest matching prefix of P in T , i.e., the unique node $\text{locus}_T(P)$ in T whose path from the root matches the longest prefix of P , as follows. First, compute and store all fingerprints of prefixes of P in $O(m)$ time and space. By Lemma 3, we can then compute the fingerprint of any substring of P in $O(1)$ time.

Next, we traverse \mathcal{TD} top-down while matching P . Initially, we search for $P[1, m]$ starting at the root of \mathcal{TD} . Suppose we have reached cluster C and have matched $P[1, i]$. If $i = m$ we return m . Otherwise ($i < m$) there are three cases:

Case 1: C is a leaf cluster. Let e be the edge stored in C . We compare $P[i+1]$ with the label of e . We return $i+1$ if they match and otherwise i .

Case 2: C is a horizontal cluster. Let A and B be the left and right child of C , respectively. We compare $P[i+1]$ with the label α of the edge to the rightmost child of A . If $P[i+1] \leq \alpha$, we continue the search in A for $P[i+1 \dots m]$. Otherwise, we continue the search in B for $P[i+1 \dots m]$.

Case 3: C is vertical cluster. Let A and B be the left and right child of C , respectively. If $|\text{spine}(A)| > m - i$ we continue the search in A for $P[i+1 \dots m]$. Otherwise, we compare the fingerprint $\phi(\text{spine}(A))$ with $\phi(P[i+1 \dots i+1 + |\text{spine}(A)|])$. If they match, we continue the search in B for $P[i+1 + |\text{spine}(A)| \dots m]$. Otherwise, we continue the search in A for $P[i+1 \dots m]$.

► **Lemma 6.** *The algorithm correctly computes the longest matching prefix of P in T .*

Next consider the running time. We compute all fingerprints of P in $O(m)$ time. Each step of top-down traversal requires constant time and since the depth of \mathcal{TD} is $O(\log n)$ the total time is $O(m + \log n)$. In summary, we have the following theorem.

► **Theorem 7.** *Let $S = S_1, \dots, S_k$ be a set of strings of total length n , and let \mathcal{TD} be the corresponding top DAG for the trie of S . On a word RAM model of computation, we can solve the compressed string indexing problem in $O(n_{\mathcal{TD}}) = O(n/\log_\sigma n)$ space and $O(m + \log n)$ time for any pattern of length m . The solution is randomized Monte-Carlo.*

In the next sections we show how to convert the above algorithm from a randomized algorithm on a word RAM machine into a deterministic algorithm on a pointer machine. We note that Theorem 7 and our subsequent solutions can be extended to other variants of prefix queries, such as *counting queries*, that return the number of occurrences of P . To do so, we store the size of each cluster in \mathcal{TD} and use the above top-down search modified to also record the highest cluster E whose top boundary is $\text{locus}_T(P)$. Since the size of E is the number of occurrences of P , we obtain a solution that also supports counting within the same complexities. From E we can also support *reporting queries*, that return the strings in S with prefix P , by simply decompressing E incurring additional linear time in the lengths of the strings with matching prefix.

4 Spine Extraction

We first consider how to handle vertical clusters (Case 3) deterministically on a pointer machine. The key challenge is to efficiently extract the characters on the spine path of a vertical cluster from top to bottom without decompressing the whole cluster. We will use this to efficiently compute longest common prefixes between spine paths and substrings of P in order to achieve total $O(m + \log n)$ time.

Given the top DAG \mathcal{TD} , the *spine path extraction problem* is to compactly represent \mathcal{TD} such that given any vertical cluster C we can return the characters of $\text{spine}(C)$. We require that the characters are reported online and from top-to-bottom, that is, the characters must be reported in sequence and we can stop extraction at any point in time. The goal is to obtain a solution that is efficient in the length of the reported prefix. In the following sections we show how to solve the problem in $O(n_{\mathcal{TD}})$ space and $O(m + \log n)$ total time over all spine path extractions.

We present a new data structure derived from the top DAG called the *vertical top DAG* and show how to use this to extract characters from a spine path. We then use this to compute the longest common prefixes between a spine path and any string and plug this in to the top down traversal in the simple solution from Section 3 to obtain Theorem 1.

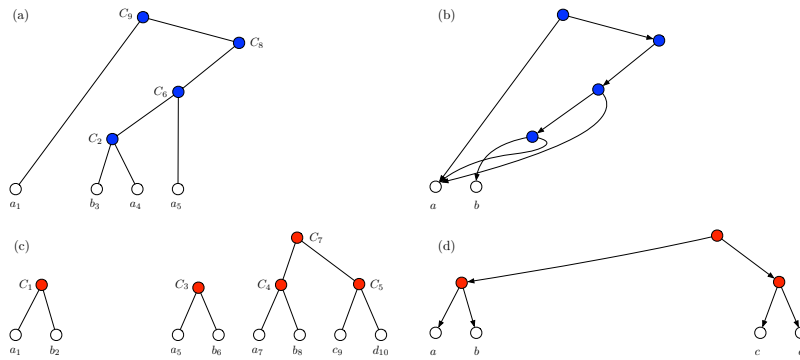


Figure 3 (a) The vertical top forest of the top tree from Figure 2(c). (b) The vertical top DAG of (a). (c) The horizontal top forest of Figure 2(a). (d) The horizontal top DAG of (a).

4.1 Vertical Top Forest and Vertical Top DAG

The *vertical top forest* \mathcal{V} of \mathcal{T} is a forest of ordered, rooted, and labeled binary trees. The nodes in \mathcal{V} are all the vertical clusters of \mathcal{T} and the leaf clusters of \mathcal{T} that correspond to edges of a spine path of some cluster in \mathcal{T} . The edges of \mathcal{V} are defined as follows. A cluster C of type (a) with children A and B in \mathcal{T} has two children in \mathcal{V} . The left and right children are the unique vertical or leaf descendants of C in \mathcal{T} whose spine path is $\text{spine}(A)$ and $\text{spine}(B)$, respectively. A cluster C of type (b) with children A and B in \mathcal{T} has a single child in \mathcal{V} , which is the unique vertical or leaf descendant of C in \mathcal{T} whose spine path is $\text{spine}(A)$. See Figure 3(a). We have the following correspondence between spine paths and subtrees in \mathcal{V} .

► **Lemma 8.** *Let C be a vertical merge in \mathcal{V} and L be the leaves of $\mathcal{V}(C)$. Then, L are the edges on $\text{spine}(C)$ and $|\mathcal{V}(C)| = O(|L|)$. Furthermore, the left-to-right ordering of L corresponds to the top-down ordering of the edges on $\text{spine}(C)$.*

For instance in Figure 3(a), the descendant leaves of C_6 are b_3, a_4, a_5 in left-to-right ordering corresponding to the edges in the spine of C_6 in Figure 2(b).

The *vertical top DAG* \mathcal{VD} is the DAG obtained by merging identical subtrees of \mathcal{V} according to the DAG compression of \mathcal{TD} . See Figure 3(b).

4.2 Spine Extraction

We now show how to solve spine path extraction using the vertical top DAG \mathcal{VD} . The key idea is to simulate a depth-first left-to-right order traversal of $\mathcal{V}(C)$ using a recursive traversal of \mathcal{VD} . In order to use spine path extraction to search for a pattern we also need to be able to continue the search in some horizontal cluster of the top DAG after extracting characters on the spine. We will therefore define what we call a *vertical exit cluster*, from which we can quickly find the cluster to continue the search from.

Define the *vertical exit cluster*, $\text{vexit}(C, \ell)$, for C at position ℓ , $1 < \ell \leq |\text{spine}(C)|$ to be the lowest common ancestor of leaves $\ell - 1$ and ℓ in $\mathcal{V}(C)$. Intuitively, if we have extracted the first ℓ characters of $\text{spine}(C)$, then $\text{vexit}(C, \ell)$ is the cluster such that all leaves in the left subtree have been extracted and only one leaf in the right subtree (corresponding to the ℓ th character) has been extracted. Our goal is to implement spine path extraction in time $O(\ell + \text{height}(C) - \text{height}(\text{vexit}(C, \ell)))$. This will yield a telescoping sum when doing multiple extractions.

4:10 Top Tree Compression of Tries

Our data structure consists of the vertical top DAG \mathcal{VD} . We augment each internal cluster by the label of the first edge on its spine path and each leaf cluster by the label of the stored edge. This uses $O(n_{\mathcal{VD}})$ space.

Given a cluster C we implement spine path extraction by simulating a depth-first left-to-right order traversal of $\mathcal{V}(C)$ using a recursive traversal of \mathcal{VD} . To extract the first character we return the stored label at C . Suppose we have extracted $\ell - 1$ characters, $1 < \ell \leq |\text{spine}(C)|$. To extract the next character continue the simulated depth-first search until we reach a cluster D in $\mathcal{V}(C)$ whose leftmost leaf is the ℓ th leaf of $\mathcal{V}(C)$. Return the character stored at D and the parent of D in $\mathcal{V}(C)$ as $\text{vexit}(C, \ell)$. (Note the parent of D is the cluster visited right before D in the simulated depth-first search.)

By Lemma 8, the algorithm correctly solves spine path extraction and the total time to extract ℓ characters is $O(\ell + \text{height}(C) - \text{height}(\text{vexit}(C, \ell)))$. We need a stack to keep track of the current search path in the traversal using $O(\text{height}(\mathcal{V}(C))) = O(\log n_{\mathcal{T}}) = O(n_{\mathcal{TD}})$ space. In summary, we have the following lemma.

► **Lemma 9.** *Let \mathcal{VD} be the vertical top DAG. We can represent \mathcal{VD} in $O(n_{\mathcal{VD}})$ space such that given a vertical cluster C , we can support spine path extraction on C in $O(\ell + \text{height}(C) - \text{height}(\text{vexit}(C, \ell)))$ time, where ℓ is the length of the extracted prefix of $\text{spine}(C)$.*

Note that we can use Lemma 9 to compute the longest common prefix of $\text{spine}(C)$ and any string by reporting the characters on the spine path from top-to-bottom and comparing them with the string until we get a mismatch. This uses $O(\ell + 1 + \text{height}(C) - \text{height}(\text{vexit}(C, \ell + 1)))$ time, where ℓ is the length of the longest common prefix.

5 An $O(m + \log n)$ Time Solution

We now plug in our spine path extraction algorithm from Section 4 into the simple algorithm from Section 3.

Define the *horizontal entry cluster* for a vertical cluster C , denoted $\text{hentry}(C)$, to be the highest horizontal cluster or leaf cluster in $\mathcal{T}(C)$ that contains all edges from $\text{top}(C)$ to children within C . For a horizontal cluster or a leaf the horizontal exit cluster is the cluster itself. Note $\text{hentry}(C)$ is the highest horizontal cluster or leaf cluster on the path from C to the leftmost leaf of C .

Our data structure consists of the data structures from Section 3 without fingerprints and Section 4. This uses $O(n_{\mathcal{TD}})$ space. To search for a string P of length m , we use the same algorithm as in Section 3, but with the following new implementation of the vertical merges.

Case 3: C is vertical cluster. Recall we have reached a vertical cluster C and have matched prefix $P[1, i]$. We check if the first character on $\text{spine}(C)$ matches $P[i + 1]$. If it does not, we continue the algorithm from $\text{hentry}(C)$. If it does, we extract characters from $\text{spine}(C)$ in order to compute the length ℓ of the longest common prefix of $\text{spine}(C)$ and $P[i + 1, m]$ and the corresponding vertical exit cluster $E = \text{vexit}(C, \ell + 1)$. Let B be the right child of E in \mathcal{TD} . We traverse the leftmost path from B to find $\text{hentry}(B)$ and continue the search for $P[i + \ell + 1, m]$ from there.

► **Lemma 10.** *The algorithm correctly computes the longest matching prefix of P in T .*

Consider the time used in a vertical step from a cluster C . The time to compute the longest common prefix computation extracting ℓ characters and walking to the corresponding horizontal entry cluster $\text{hentry}(\text{vexit}(C, \ell))$ is $O(\ell + h(C) - h(\text{vexit}(C, \ell)) + h(\text{vexit}(C, \ell)) - h(\text{hentry}(\text{vexit}(C, \ell)))) = O(\ell + h(C) - h(\text{hentry}(\text{vexit}(C, \ell))))$. Hence, if we have z vertical steps from clusters C_1, \dots, C_z extracting ℓ_1, \dots, ℓ_z characters ending in $E_i = \text{hentry}(\text{vexit}(C_i, \ell_i))$, respectively, we use time

$$\sum_{i=1}^z O(\ell_i + h(C_i) - h(E_i)) = O\left(\sum_{i=1}^z \ell_i + h(C_1) - h(E_z)\right) = O(m + \log n_{\mathcal{T}}).$$

This follows from the fact that C_1, \dots, C_z and E_1, \dots, E_z all lie on the same root-to-leaf path in \mathcal{T} and that $h(E_i) \geq h(C_{i+1})$. As in Section 3, the total time used at horizontal merges is $O(\log n_{\mathcal{T}})$, as E_1, \dots, E_z all lie on the same root-to-leaf path in \mathcal{T} and we only walk down in the tree during the horizontal merges. This concludes the proof of the $O(m + \log n)$ query time in Theorem 1.

6 Spine Path Extraction with Constant Overhead

Next, we show how to achieve the $O(m \log \sigma)$ query time in Theorem 1. Our current solutions for horizontal merges (Case 2) from Section 3 and vertical merges (Case 3) from Section 5 both require $\Omega(m + \log n)$ and hence we need new techniques for both cases to achieve the $O(m \log \sigma)$ time bound. We consider vertical merges in this section and horizontal merges in the next section.

In this section, we improve the total time used on spine extraction to optimal $O(m)$ time. To do so we first introduce and present a novel solution to a new path extraction problem on trees in Section 6.1 and then show how to use this to extract characters from the spine in Section 6.2.

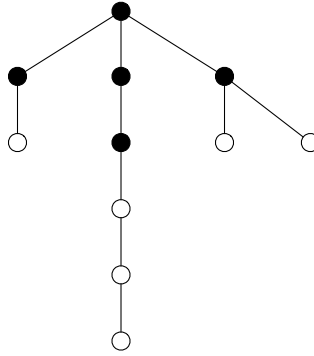
6.1 Path Extraction in Trees

Given a tree T with n nodes, the *path extraction problem* is to compactly represent T such that given a node v we can return the nodes on the path from the root of T to v in constant time per node. We require that the nodes are reported online and from top-to-bottom, that is, the nodes must be reported in sequence and we can stop the extraction at any point in time. The ordering of the nodes from top to bottom is essential. The other direction (from v to the root) is trivial since we can simply store parent pointers and traverse that path using linear space and constant time per node. If we allow word RAM tricks then we can easily solve the problem in the same bounds by using an existing *level ancestor data structure* [2, 12, 26]. We present an optimal solution that does not use word RAM tricks and works on a pointer machine. As mentioned in the introduction, an optimal solution can be also obtained by plugging in known tools, but we believe that our method is simpler and elegant.

Let $\text{depth}(v)$ and $\text{height}(v)$ be the distance from v to the root and to deepest leaf in v 's subtree, respectively. Decompose T into a top part T_{top} consisting of nodes v , such that $\text{depth}(v) \leq \text{height}(v)$, and a bottom part T_{bot} consisting of the remaining nodes. For each leaf u in T_{top} we store the path from the root of T_{top} to u explicitly in a linked list sorted by increasing depth. (see Figure 4). Note that multiple copies of the same node may be stored across different lists. Each such path to a leaf u uses $O(\text{depth}(u))$ space, and hence the total space for all paths in T_{top} is

$$\sum_{u \text{ a leaf in } T_{\text{top}}} \text{depth}(u) \leq \sum_{u \text{ a leaf in } T_{\text{top}}} \text{height}(u) = O(n),$$

where the first equality follows by definition of the decomposition and the second follows since the longest paths from a descendant leaf in $T(u)$ to a leaf u in T_{top} are disjoint for all the leaves u in T_{top} . For all internal nodes in T_{top} we store a pointer to a leaf below it. For



■ **Figure 4** The tree decomposition for path extraction. The black nodes are the nodes in T_{top} and the white nodes are the nodes in T_{bot} . The three root-to-leaf paths are stored as three linked lists sorted by increasing depth. The total size of the lists is $2 + 3 + 2 = 7$.

all nodes v in T_{bot} we store a pointer to the unique ancestor u that is a leaf in T_{top} . We answer a path extraction query for a node v as follows. If v is in T_{top} we follow the leaf pointer and output the path stored in this leaf from the root until we reach v . If v is in T_{bot} we jump to the unique ancestor leaf u of v in T_{top} . We extract the path from the root to u , while simultaneously following parent pointers from v until we reach u storing these nodes on a stack. That is, each time we extract a node from the root-to- u path we follow a parent pointer and put the next node on the stack. We stop pushing nodes to the stack when we reach u . When we have output all nodes from the root to the leaf in T_{top} we output the nodes from the stack. Since $\text{depth}(u) \leq \text{height}(u)$ the path from the root to u is at least as long as the path from v to u plus 1. Therefore, the whole path is extracted. We spend $O(1)$ time per node and hence we have the following result.

► **Lemma 11.** *Given a tree T with n nodes, we can solve the path extraction problem in linear space and preprocessing and constant time per reported node.*

6.2 Optimal Spine Path Extraction

We plug the path extraction solution into our depth-first search traversal of the vertical top DAG \mathcal{VD} to speed up spine extraction and longest common prefix computation. Recall that given a vertical cluster C , our goal is to simulate a depth-first left-to-right order traversal of the subtree $\mathcal{V}(C)$ using the vertical top DAG \mathcal{VD} .

We construct the *left-path suffix forest* L of \mathcal{VD} as follows. The nodes of L are the nodes of \mathcal{VD} . If C has a left child A in \mathcal{VD} then A is the parent of C in L . Hence, any leftmost path in \mathcal{VD} corresponds to a path from a node to an ancestor of the node in L . We now store L with the path extraction data structure from Lemma 11. We implement the depth-first traversal as before except that whenever the traversal reaches an unexplored cluster C' in $\mathcal{V}(C)$ we begin path extraction for that cluster corresponding to the path from C' to the leftmost descendant leaf \hat{C} . We extract the leaf \hat{C} and then continue the depth-first traversal from there. Hence, the current search path of the depth-first traversal is partitioned into an alternating sequence of leftmost paths and right edges. Whenever we need to go up on a left edge in the traversal we extract the next node for the corresponding path extraction instance.

To extract the topmost ℓ characters of $\text{spine}(C)$ we now use constant time to find the leftmost descendant leaf of $\mathcal{V}(C)$ and then $O(\ell)$ time to traverse the first ℓ leaves. Hence, we improve the time from $O(\text{height}(\mathcal{V}(C)) + \ell)$ to $O(\ell)$. At any point during the traversal we

maintain ongoing path extraction instances along the current search path. The stacks each of these need are of size at most linear in the length of their corresponding subpath of the search path and hence this requires at most $O(\log n_{\mathcal{VD}})$ extra space.

► **Lemma 12.** *We can represent the vertical top DAG \mathcal{VD} in $O(n_{\mathcal{VD}})$ space such that given a vertical cluster C , we can support spine path extraction on C in $O(\ell)$ time, where ℓ is the length of the extracted prefix of spine(C).*

7 Horizontal Access

We now show how to efficiently handle horizontal merges (Case 2). In the simple algorithm from Section 3 we use constant time at each horizontal merge leading to an $O(\log n_{\mathcal{T}})$ total time solution. Since we cannot afford $O(\log n_{\mathcal{T}})$ time we instead show how to handle all horizontal merges in $O(m \log \sigma)$ time. The key idea is to convert the problem into a variant of the *random access problem* for grammar compressed strings, and then design a linear-space logarithmic-query solution to the random access problem. We describe the random access problem in Section 7.1 and present our solution to it in Section 7.2, we introduce the horizontal top DAG in Section 7.3, and define and solve the *horizontal access problem* in Section 7.4.

7.1 Grammars and Random Access

Grammar compression replaces a long string S by a small context-free grammar (CFG) \mathcal{G} . We view a grammar \mathcal{G} as a DAG, where each node is a grammar symbol and each rule defines directed ordered edges from the righthand side to the lefthand side. Given a node C in \mathcal{G} , we define $T(C)$ to be the parse tree rooted at C and $S(C)$ to be the string consisting of the leaves of $T(C)$ in left-to-right order. Note that given a rule $C \rightarrow C_1 C_2 \dots C_k$ we have that $S(C) = S(C_1) \cdot S(C_2) \cdot \dots \cdot S(C_k)$, where \cdot denotes concatenation. Given a grammar \mathcal{G} representing a string S , the *random access problem* is to compactly represent \mathcal{G} while supporting fast *access queries*, that is, given an index i in S report $S[i]$. Bille et al. [19] showed how to do random access in $O(\log |S|)$ time using $O(n_{\mathcal{G}} \cdot \alpha_k(n_{\mathcal{G}}))$ space² on a pointer machine model. Furthermore, given a node C in \mathcal{G} , access queries can be supported on the string $S(C)$ in time $O(\log |S(C)|)$.

For our purposes, we need to slightly extend this result to *gapped grammars*. A gapped grammar is a grammar except that each internal rule is now of the form $C \rightarrow C_1 g_1 C_2 \dots g_{k-1} C_k$, where g_i is a non-negative integer called the *gap*. The string generated by \mathcal{G} is now $S(C) = S(C_1) 0^{g_1} S(C_2) \dots S(C_{k-1}) 0^{g_{k-1}} S(C_k)$ and hence the resulting string generated is as before except for the inserted gaps of runs of 0's. Note that $|S(C)| = |S(C_1)| + g_1 + |S(C_2)| + \dots + g_{k-1} + |S(C_k)|$. The above random access result is straightforward to generalize to gapped grammars:

► **Lemma 13** (Bille et al. [19]). *Let S be a string compressed into a gapped grammar \mathcal{S} of size $n_{\mathcal{S}}$. Given a node v in \mathcal{S} , we can support random access queries in $S(v)$ in $O(\log(|S(v)|))$ time using $O(n_{\mathcal{S}} \cdot \alpha_k(n_{\mathcal{S}}))$ space. The solution works on a pointer machine model of computation.*

² Here $\alpha_k(n)$ for any constant k denotes the inverse of the k^{th} row of Ackermann's function, defined as $\alpha_k(n) = 1 + \alpha_k(\alpha_{k-1}(n))$ so that $\alpha_1(n) = n/2$, $\alpha_2(n) = \log n$, $\alpha_3(n) = \log^* n$, and so on.

7.2 Horizontal Access in Linear Space

Bille et al. [19] further showed that the inverse-Ackermann factor in the space complexity of Lemma 13 can be removed if we assume a word RAM model of computation. In this section we show that this can also be achieved on a pointer machine. To this end, we need to replace a single component in the solution of Bille et al., their *weighted level ancestor* structure. In the weighted level ancestor problem, we are given a tree T on n nodes with positive weights on the edges. For every node $u \in T$, let $d(u)$ be its distance to the root, and let $\text{parent}(u)$ be its parent. Then, the goal is to preprocess T to answer the following *weighted level ancestor* queries: given a non-root node $u \in T$ and a positive number $x \leq d(u)$, find an ancestor v such that $d(v) \geq x$ but $d(\text{parent}(v)) < x$.

Without getting into the proof of Lemma 13, it suffices to say that (1) performing a random access query boils down to performing $O(\log(|S(v)|))$ weighted level ancestor queries, and (2) in order for all these $O(\log(|S(v)|))$ queries to be done in total $O(\log(|S(v)|))$ time, the time for each weighted level ancestor query should be proportional to $\log \frac{d(u)}{d(v) - d(\text{parent}(v))}$. Intuitively, we seek a position on an edge at distance x from the root, and the longer the found edge is the smaller the query time should be. We next show how to achieve such query time using linear space on a pointer machine, implying an inverse-Ackermann factor improvement to Lemma 13.

► **Lemma 14.** *A tree T on n nodes can be preprocessed in $O(n)$ space to answer a weighted level ancestor query for a node $u \in T$ and a number x in $O(1 + \log \frac{d(u)}{d(v) - d(\text{parent}(v))})$ time, where v is the found ancestor of u .*

► **Corollary 15.** *Let S be a string compressed into a gapped grammar \mathcal{S} of size $n_{\mathcal{S}}$. Given a node v in \mathcal{S} , we can support random access queries in $S(v)$ in $\log(|S(v)|)$ time using $O(n_{\mathcal{S}})$ space. The solution works on a pointer machine model of computation.*

7.3 Horizontal Top Tree and Horizontal Top DAGs

Similar to the vertical top forest we define the *horizontal top forest* \mathcal{H} of \mathcal{T} as a forest of ordered and rooted trees that consists of all horizontal clusters of \mathcal{T} and leaves of \mathcal{T} whose top boundary is shared with a horizontal cluster. We define the edges in of C in \mathcal{H} as follows. Let C be a horizontal cluster C with children A and B in \mathcal{T} . If A is a horizontal cluster or a leaf then the left child of C is A , and if A is a vertical cluster then the left child of C is $\text{hentry}(A)$. Similarly, the right child of C is either B or $\text{hentry}(B)$. See Figure 3. We have the following property of \mathcal{H} .

► **Lemma 16.** *Let C be a horizontal merge in \mathcal{H} . Then, the leaves of $\mathcal{H}(C)$ are the edges to children of the top boundary node of C and the left-to-right ordering of the leaves correspond to the left-to-right ordering of the children of C in T . All nodes in $\mathcal{H}(C)$ has $\text{top}(C)$ as top boundary node.*

For instance in Figure 3(c) the descendant leaves of C_7 are a_7 , b_8 , c_9 , and d_{10} in left to right ordering corresponding to the edges to the children of $\text{top}(C_7)$. Given the horizontal top forest we define the *horizontal top DAG* \mathcal{HD} as the DAG obtained by merging the subtrees of \mathcal{H} according to the DAG compression of \mathcal{T} into \mathcal{TD} .

7.4 Gapped Grammars and Horizontal Access

Let C be an internal cluster in \mathcal{H} . The *spine child* of C is the unique child of C that contains the first edge of $\text{spine}(C)$. A descendant cluster D of C is a *spine descendant* of C if all clusters on the path from C to D are spine children of their parent. Define the *horizontal exit cluster* for a horizontal cluster C and character α , denoted $\text{hexit}(C, \alpha)$, to be the highest cluster in $\mathcal{H}(C)$ that has the unique leaf in $\mathcal{H}(C)$ labeled α as a spine descendant.

Given the horizontal top DAG \mathcal{HD} , the *horizontal access problem*, is to compactly represent \mathcal{HD} such that given a horizontal merge C and a character $\alpha \in \Sigma$, we can efficiently determine if $\text{top}(C)$ has an edge to a child labeled α within C and if so return the horizontal exit cluster $\text{hexit}(C, \alpha)$. In this section, we show how to solve the horizontal access problem in $O(n_{\mathcal{HD}})$ space and $O(\log \sigma)$ time.

The *characteristic vector* of a cluster C is a binary string encoding the labels of edges to children of $\text{top}(C)$. More precisely, given a character $\alpha \in \Sigma$ define $\text{rank}(\alpha) \in \{1, \dots, \sigma\}$ as the rank of α in the sorted order of characters of Σ . Also, given a cluster C in \mathcal{H} define $\text{rank}(C)$ to be the set of ranks of leaf labels in $\mathcal{H}(C)$. We define the characteristic vector $S(C)$ recursively as follows. If C is a leaf cluster $S(C) = 1$ and if C is an internal cluster with children C_1, \dots, C_k , then $S(C) = S(C_1)0^{g_1}S(C_2) \cdots S(C_{k-1})0^{g_{k-1}}S(C_k)$, where $g_i = \min(\text{rank}(C_{i+1})) - \max(\text{rank}(C_i)) + 1$. Note that $|S(C)| \leq \sigma$ for any cluster C . From the definition we have the following correspondence between the characteristic vector and the leaf labels of a cluster.

► **Lemma 17.** *Given a cluster C in \mathcal{H} and a character $\alpha \in \Sigma$, α is a leaf label in $\mathcal{H}(C)$ iff $S(C)[\text{rank}(\alpha) - \min(\text{rank}(C))] = 1$.*

Let R_1, \dots, R_z be the root clusters of the trees in \mathcal{H} and note that if we add a virtual root cluster R as the parent of R_1, \dots, R_z , \mathcal{H} is a gapped parse tree for the string $S = S(R_1) \cdots S(R_z)$. Hence, the horizontal top DAG \mathcal{HD} is a gapped grammar for the same string. By Lemma 17 we can determine if there is an edge labeled α out of $\text{top}(C)$ in C using a random access query on the corresponding gapped grammar using time $O(\log |S(C)|) = O(\log \sigma)$. If this edge exists, we can also find $\text{hexit}(C, \alpha)$ in the same time using similar ideas. More precisely, we have the following result.

► **Lemma 18.** *Given a cluster C in \mathcal{H} and a character $\alpha \in \Sigma$ we can solve the horizontal access problem in $O(n_{\mathcal{HD}})$ space and $O(\log \sigma)$ time.*

8 An $O(m \log \sigma)$ Solution

We can now plug in the spine extraction from Section 6.2 and the horizontal access from Section 7 into the simple algorithm from Section 3. Define the *vertical entry cluster* for a horizontal cluster C , denoted $\text{ventry}(C)$, to be the highest vertical cluster or leaf cluster in $\mathcal{T}(C)$ that contains the first edge on $\text{spine}(C)$.

Our data structure consists of the data structure from Section 6.2 for spine path extraction and the data structure from Section 7.3 for horizontal access. Furthermore, we store for each vertical cluster in \mathcal{TD} a pointer to its horizontal entry cluster and for each horizontal cluster a pointer to its vertical entry cluster. In total this uses $O(n_{\mathcal{TD}})$ space.

To search we alternate between horizontal accesses using Lemma 18 and spine path extractions using Lemma 12. Instead of traversals to find entry clusters we jump directly using the new pointers. Specifically, we have the following modified algorithm:

Initially, we search for $P[1, m]$ starting at the root of \mathcal{TD} . Suppose we have reached cluster C and have matched $P[1, i]$. If $i = m$ we return m . Otherwise ($i < m$) there are three cases:

Case 1: C is a leaf cluster. Let e be the edge stored in C . We compare $P[i + 1]$ with the label of e . We return $i + 1$ if they match and otherwise i .

Case 2: C is a horizontal cluster. Compute $E = \text{hexit}(C, P[i + 1])$. If $P[i + 1]$ does not match return i . Otherwise, continue the search for $P[i + 1, m]$ from $\text{ventry}(E)$.

Case 3: C is vertical cluster. We check if the first character on $\text{spine}(C)$ matches $P[i + 1]$. If it does not we continue the algorithm from $\text{hentry}(C)$. Otherwise, we extract characters from $\text{spine}(C)$ in order to compute the length ℓ of the longest common prefix of $\text{spine}(C)$ and $P[i + 1, m]$ and the corresponding vertical exit cluster $E = \text{vexit}(C, \ell + 1)$. Continue the search for $P[\ell + 1, m]$ from $\text{hentry}(E)$.

► **Lemma 19.** *The algorithm correctly computes the longest matching prefix of P in T .*

Consider the alternating sequence of horizontal accesses and spine extractions. Each time we go from a horizontal access to a spine extraction the current character of P must match the first character on the spine. Hence, each horizontal access is on a distinct character of P and the total number of horizontal accesses is at most m . By Lemma 18 it follows that the total time for horizontal accesses is $O(m \log \sigma)$. Since the sequence is alternating the number of spine extractions is at most $m + 1$. Hence, by Lemma 12 the total time for spine extractions is at most $O(m)$. This concludes the proof of the $O(m \log \sigma)$ query time in Theorem 1.

References

- 1 Peyman Afshani, Lars Arge, and Kasper Green Larsen. Higher-dimensional orthogonal range reporting and rectangle stabbing in the pointer machine model. In *Proc. 28th SoCG*, pages 323–332, 2012.
- 2 Stephen Alstrup and Jacob Holm. Improved algorithms for finding level ancestors in dynamic trees. In *Proc. 27th ICALP*, pages 73–84, 2000.
- 3 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, 2005.
- 4 J-I Aoe. An efficient digital search algorithm by using a double-array structure. *IEEE Trans. Soft. Eng.*, 15(9):1066–1077, 1989.
- 5 Julian Arz and Johannes Fischer. LZ-compressed string dictionaries. In *Proc. 24th DCC*, pages 322–331, 2014.
- 6 Julian Arz and Johannes Fischer. Lempel–Ziv-78 Compressed String Dictionaries. *Algorithmica*, pages 1–36, 2018.
- 7 Nikolas Askitis and Ranjan Sinha. Engineering scalable, cache and space efficient tries for strings. *The VLDB Journal*, 19(5):633–660, 2010.
- 8 Djamel Belazzougui, Paolo Boldi, and Sebastiano Vigna. Dynamic z-fast tries. In *Proc. 17th SPIRE*, pages 159–172, 2010.
- 9 Djamel Belazzougui, Fabio Cunial, Travis Gagie, Nicola Prezza, and Mathieu Raffinot. Composite repetition-aware data structures. In *Proc. 26th CPM*, pages 26–39, 2015.
- 10 Djamel Belazzougui, Travis Gagie, Pawel Gawrychowski, Juha Kärkkäinen, Alberto Ordóñez, Simon J Puglisi, and Yasuo Tabei. Queries on LZ-bounded encodings. In *Proc. 25th DCC*, pages 83–92, 2015.
- 11 Djamel Belazzougui, Travis Gagie, Simon Gog, Giovanni Manzini, and Jouni Sirén. Relative FM-indexes. In *Proc. 21st SPIRE*, pages 52–64, 2014.

- 12 Michael A. Bender and Martin Farach-Colton. The Level Ancestor Problem simplified. *Theoret. Comput. Sci.*, 321(1):5–12, 2004.
- 13 David Benoit, Erik D Demaine, J Ian Munro, Rajeev Raman, Venkatesh Raman, and S Srinivasa Rao. Representing trees of higher degree. *Algorithmica*, 43(4):275–292, 2005.
- 14 Samuel W. Bent, Daniel D. Sleator, and Robert E. Tarjan. Biased Search Trees. *SIAM J. Comput.*, 14(3):545–568, 1985.
- 15 Philip Bille, Mikko B. Ettiienne, Inge Li Gørtz, and Hjalte W. Vildhøj. Time-space trade-offs for Lempel-Ziv compressed indexing. *Theor. Comput. Sci.*, 713:66–77, 2018.
- 16 Philip Bille, Finn Fernstrøm, and Inge Li Gørtz. Tight Bounds for Top Tree Compression. In *Proc. 24th SPIRE*, pages 97–102, 2017.
- 17 Philip Bille, Inge Li Gørtz, and Frederik Rye Skjoldjensen. Deterministic Indexing for Packed Strings. In *Proc. 28th CPM*, 2017.
- 18 Philip Bille, Inge Li Gørtz, Oren Weimann, and Gad M. Landau. Tree compression with top trees. *Inf. Comput.*, 243:166–177, 2015. Announced at ICALP 2013.
- 19 Philip Bille, Gad M. Landau, Rajeev Raman, Kunihiko Sadakane, Srinivasa Rao Satti, and Oren Weimann. Random Access to Grammar-Compressed Strings and Trees. *SIAM J. Comput.*, 44(3):513–539, 2015. Announced at SODA 2011.
- 20 Barnard Chazelle. Lower bounds for orthogonal range searching: I. The reporting case. *J. ACM*, 37(2):200–212, 1990.
- 21 Bernard Chazelle and Burton Rosenberg. Simplex range reporting on a pointer machine. *Comput. Geom.*, 5(5):237–247, 1996.
- 22 Anders R. Christiansen and Mikko B. Ettiienne. Compressed Indexing with Signature Grammars. In *Proc. 13th LATIN*, pages 331–345, 2018.
- 23 Francisco Claude and Gonzalo Navarro. Self-indexed grammar-based compression. *Fundamenta Informaticae*, 111(3):313–337, 2011.
- 24 Francisco Claude and Gonzalo Navarro. Improved grammar-based compressed indexes. In *Proc. 19th SPIRE*, pages 180–192, 2012.
- 25 John J Darragh, John G Cleary, and Ian H Witten. Bonsai: a compact representation of trees. *Softw. Pract. Exper.*, 23(3):277–291, 1993.
- 26 Paul F. Dietz. Finding level-ancestors in dynamic trees. In *Proc. 2nd WADS*, pages 32–40, 1991.
- 27 Peter J. Downey, Ravi Sethi, and Robert E. Tarjan. Variations on the common subexpression problem. *J. ACM*, 27(4):758–771, 1980.
- 28 Bartłomiej Dudek and Paweł Gawrychowski. Slowing Down Top Trees for Better Worst-Case Compression. In *Proc. 29th CPM*, pages 16:1–16:8, 2018.
- 29 Andrea Farruggia, Travis Gagie, Gonzalo Navarro, Simon J Puglisi, and Jouni Sirén. Relative suffix trees. *Comput. J.*, 61(5):773–788, 2017.
- 30 Edward Fredkin. Trie Memory. *Commun. ACM*, 3(9):490–499, 1960.
- 31 Travis Gagie, Paweł Gawrychowski, Juha Kärkkäinen, Yakov Nekrich, and Simon J. Puglisi. A Faster Grammar-Based Self-index. In *Proc. 6th LATA*, pages 240–251, 2012.
- 32 Travis Gagie, Paweł Gawrychowski, Juha Kärkkäinen, Yakov Nekrich, and Simon J. Puglisi. LZ77-based self-indexing with faster pattern matching. In *Proc. 11th LATIN*, pages 731–742, 2014.
- 33 Roberto Grossi and Giuseppe Ottaviano. Fast compressed tries through path decompositions. *ACM J. Exp. Alg.*, 19:3–4, 2015.
- 34 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.
- 35 Torben Hagerup. Sorting and Searching on the Word RAM. In *Proc. 15th STACS*, pages 366–398, 1998.
- 36 Meng He, J. Ian Munro, and Gelin Zhou. Data Structures for Path Queries. *ACM Trans. Algorithms*, 12(4):53:1–53:32, 2016.
- 37 Robert Hood and Robert Melville. Real-Time Queue Operation in Pure LISP. *Inf. Process. Lett.*, 13(2):50–54, 1981.

- 38 Lorenz Hübschle-Schneider and Rajeev Raman. Tree compression with top trees revisited. In *Proc. 14th SEA*, pages 15–27, 2015.
- 39 Shunsuke Kanda, Kazuhiro Morita, and Masao Fuketa. Compressed double-array tries for string dictionaries supporting fast lookup. *Knowl. Inf. Syst.*, 51(3):1023–1042, 2017.
- 40 Shunsuke Kanda, Kazuhiro Morita, and Masao Fuketa. Practical implementation of space-efficient dynamic keyword dictionaries. In *Proc. 24th SPIRE*, pages 221–233, 2017.
- 41 Juha Kärkkäinen and Esko Ukkonen. Lempel-Ziv parsing and sublinear-size index structures for string matching. In *Proc. 3rd WSP*, pages 141–155, 1996.
- 42 Richard M. Karp and Michael O. Rabin. Efficient Randomized Pattern-Matching Algorithms. *IBM Journal of Research and Development*, 31(2):249–260, 1987.
- 43 Donald E. Knuth, Jr. James H. Morris, and Vaughan R. Pratt. Fast Pattern Matching in Strings. *SIAM J. Comput.*, 6(2):323–350, 1977.
- 44 Donald Erwin Knuth. *The Art of Computer Programming, Volume 1*. Addison Wesley, 1969.
- 45 Veli Mäkinen. Compact suffix array—a space-efficient full-text index. *Fundamenta Informaticae*, 56(1-2):191–210, 2003.
- 46 Veli Mäkinen and Gonzalo Navarro. Succinct suffix arrays based on run-length encoding. *Nordic J. Comput.*, 12(1):40–66, 2005.
- 47 Veli Mäkinen, Gonzalo Navarro, Jouni Sirén, and Niko Välimäki. Storage and retrieval of individual genomes. In *Proc. 13th RECOMB*, pages 121–137, 2009.
- 48 Veli Mäkinen, Gonzalo Navarro, Jouni Sirén, and Niko Välimäki. Storage and retrieval of highly repetitive sequence collections. *J. Comp. Bio.*, 17(3):281–308, 2010.
- 49 Gonzalo Navarro and Veli Mäkinen. Compressed Full-text Indexes. *ACM Comput. Surv.*, 39(1), 2007.
- 50 Gonzalo Navarro and Nicola Prezza. Universal compressed text indexing. *Theor. Comput. Sci.*, 762:41–50, 2019.
- 51 Takaaki Nishimoto, I Tomohiro, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Dynamic index and LZ factorization in compressed space. *Disc. App. Math.*, 2019.
- 52 Andreas Poyias and Rajeev Raman. Improved practical compact dynamic tries. In *Proc. 22nd SPIRE*, pages 324–336, 2015.
- 53 Franco P Preparata and Se June Hong. Convex hulls of finite sets of points in two and three dimensions. *Communications of the ACM*, 20(2):87–93, 1977.
- 54 Rajeev Raman, Venkatesh Raman, and Srinivasa Rao Satti. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. *ACM Trans. Algorithms*, 3(4):43, 2007.
- 55 Kunihiko Sadakane. Compressed text databases with efficient query algorithms based on the compressed suffix array. In *Proc. 11th ISAAC*, pages 410–421, 2000.
- 56 Jouni Sirén, Niko Välimäki, Veli Mäkinen, and Gonzalo Navarro. Run-length compressed indexes are superior for highly repetitive sequence collections. In *Proc. 15th SPIRE*, pages 164–175, 2008.
- 57 Takuya Takagi, Keisuke Goto, Yuta Fujishige, Shunsuke Inenaga, and Hiroki Arimura. Linear-size CDAWG: new repetition-aware indexing and grammar compression. In *Proc. 24th SPIRE*, pages 304–316, 2017.
- 58 Takuya Takagi, Shunsuke Inenaga, Kunihiko Sadakane, and Hiroki Arimura. Packed compact tries: A fast and efficient data structure for online string processing. *IEICE Trans. on Fund. Elect., Comm. and Comp. Sci.*, 100(9):1785–1793, 2017.
- 59 Robert Endre Tarjan. A class of algorithms which require nonlinear time to maintain disjoint sets. *Journal of Computer and System Sciences*, 18(2):110–127, April 1979.
- 60 Kazuya Tsuruta, Dominik Köppl, Shunsuke Kanda, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Dynamic Packed Compact Tries Revisited. arXiv preprint, 2019. [arXiv:1904.07467](https://arxiv.org/abs/1904.07467).
- 61 Susumu Yata. Dictionary compression by nesting prefix/patricia tries. In *Proc. 17th Meeting of the Association for Natural Language*, 2011.
- 62 Naoki Yoshinaga and Masaru Kitsuregawa. A self-adaptive classifier for efficient text-stream processing. In *Proc. 25th COLING*, pages 1091–1102, 2014.

Two Phase Transitions in Two-Way Bootstrap Percolation

Ahad N. Zehmakan

ETH Zurich, Switzerland

abdolahad.noori@inf.ethz.ch

Abstract

Consider a graph G and an initial random configuration, where each node is black with probability p and white otherwise, independently. In discrete-time rounds, each node becomes black if it has at least r black neighbors and white otherwise. We prove that this basic process exhibits a threshold behavior with two phase transitions when the underlying graph is a d -dimensional torus and identify the threshold values.

2012 ACM Subject Classification Theory of computation

Keywords and phrases bootstrap percolation, cellular automata, phase transition, d -dimensional torus, r -threshold model, biased majority

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.5

Acknowledgements The author likes to thank Raphael Cerf, Bernd Gärtner, and Roberto H. Schonmann for several stimulating discussions.

1 Introduction

Consider a graph $G = (V, E)$ and an initial *random configuration*, where each node is independently black with some probability p and white otherwise. In *r -bootstrap percolation* (or shortly *r -BP*) for some positive integer r , in each discrete-time round white nodes with at least r black neighbors become black and black nodes stay unchanged. This basic process is meant to model different progressive dynamics such as rumor spreading in a society, fire propagation in a forest, and infection spreading among cells, where a black/white node corresponds to an individual who is informed/uninformed of a rumor, a tree which is or not on fire, or an infected/uninfected cell.

In the above examples if a node becomes black, it remains black forever. For instance, an individual who is informed of a rumor remains informed. However, there exist many real-world examples where nodes might keep switching between black and white. For example, two service providers might be competing to get people adopting their services, and thus users may switch among two services back and forth. Another example is opinion forming regarding an election in a community, where an individual might adopt the positive opinion if a certain number/fraction of its connections are positive, and become negative otherwise. To study this kind of non-progressive processes the following model has been introduced. For a graph G and an initial random configuration, in *two-way r -bootstrap percolation* in each round a node becomes black if it has at least r black neighbors and white otherwise.

The behavior of these two basic models have been extensively studied by researchers from a wide spectrum of fields, like statistical physics [2, 43], distributed computing [41, 19, 21], mathematics [3, 33], and even sociology [26] due to their various applications, such as distributed fault-local mending [41], modeling biological interactions [35], viral marketing [34], and modeling disordered magnetic systems [6].

The first natural question arises: How long does it take for the above processes to stabilize? Both of these processes are induced by deterministic updating rules and for a graph $G = (V, E)$ there are $2^{|V|}$ possible colorings (configurations). Therefore, by starting



© Ahad N. Zehmakan;

licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 5; pp. 5:1–5:21

Leibniz International Proceedings in Informatics



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

from any initial configuration the process must reach a cycle of configurations eventually. The length of this cycle and the number of rounds the process needs to reach the cycle are respectively called the *period* and the *consensus time* of the process. In r -BP, the periodicity is always one and the consensus time is bounded by $|V| - 1$, which is tight (for instance consider a path P_n and 1-BP, where initially all nodes are white except one of the leaves). In two-way r -BP, $2^{|V|}$ is a trivial upper bound on both period and consensus time. However, interestingly Goles and Olivos [25] proved that the period is always one or two and Fogelman, Goles, and Weisbuch [20] showed that the consensus time is bounded by $\mathcal{O}(|E|)$, which is tight (consider a cycle C_n which is fully white except two adjacent black nodes for $r = 1$).

Arguably, the most well-studied question concerning the behavior of these models is: What is the minimum p for which black color takes over with probability approaching one? We say black/white color *takes over* if the whole graph becomes black/white. This question has been investigated on different classes of graphs such as hypercube [5], the binomial random graph [31, 15, 38], random regular graphs [9, 24, 37], infinite trees [33], and many others. A substantial amount of attention has been devoted to address this question on the d -dimensional torus, due to the study of certain interacting particle systems like fluid flow in rocks [1], dynamics of glasses [22], and biological interactions [35]. The d -dimensional torus \mathbb{T}_L^d is the graph with node set $[L]^d := \{1, \dots, L\}^d$, where two nodes are adjacent if and only if they differ by 1 or $L - 1$ in exactly one coordinate. Notice we always assume that d is a constant while let L tend to infinity.

The aforementioned question regarding r -BP on the d -dimensional torus \mathbb{T}_L^d first was considered by Aizenman and Lebowitz [2], who proved that for $r = d = 2$ the process exhibits a *weak threshold behavior* at $\mathcal{P}_1 := (\log_{(r-1)} L)^{-(d-r+1)}$ where $\log_{(r)} L := \log \log_{(r-1)} L$ for $r \geq 1$ and $\log_0 L = L$. That is, black color takes over for $p = \omega(\mathcal{P}_1)$ and it does not for $p = o(\mathcal{P}_1)$ asymptotically almost surely¹. Cerf and Cirillo [13] made one step further by proving that the process exhibits a similar weak threshold behavior at \mathcal{P}_1 for $d = r = 3$. Finally, Cerf and Manzo [14] extended this result to all values of $1 \leq r \leq d$, building on the work by Schonmann [43]. Later on, it was proven, by Holroyd [29], that for $d = r = 2$ the process actually exhibits a *sharp threshold behavior*; that is, a.a.s. black color takes over if $p \geq (1 + \epsilon) \lambda \mathcal{P}_1$ and it does not if $p \leq (1 - \epsilon) \lambda \mathcal{P}_1$ for any constant $\epsilon > 0$, where $\lambda(d, r) > 0$ is a constant. This sharp threshold behavior was proven for the case of $d = r = 3$ by Balogh, Bollobas, and Morris [7] and finally for all values of $1 < r \leq d$ by Balogh, Bollobas, Duminil-Copin, and Morris [6]. Along the way, as an intermediate step the behavior of a similar process was also studied. In *modified r -bootstrap percolation* on \mathbb{T}_L^d , by starting from a random initial configuration in every round each white node becomes black if it has black neighbor(s) in at least r distinct dimensions and black nodes remain unchanged. See [28, 29] by Holroyd regarding the sharp threshold behavior of modified r -BP for $r = d$.

For two-way r -BP on \mathbb{T}_L^d , Schonmann [42], by applying the results from [2], proved that for $d = r = 2$ black color takes over if $p = \omega(1/\sqrt{\log L})$ and it does not if $p = o(1/\sqrt{\log L})$ a.a.s., i.e., it exhibits a weak threshold behavior. What about the higher dimensions? Despite several attempts [16, 35, 3, 42] over the last three decades, this question has remained open. Intuitively speaking, the inherent difficulty of analyzing two-way r -BP comes from the fact that unlike r -BP, in two-way r -BP a node may switch between two colors back and forth.

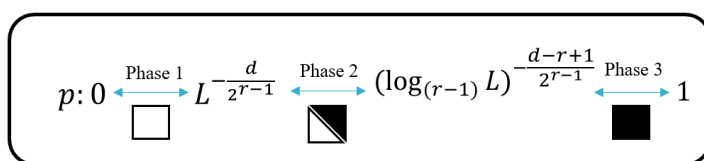
By providing several new techniques, using some ideas inspired from [42, 23] (where the special case of $r = d = 2$ is handled), and applying some prior results regarding (modified) r -BP [14, 6, 28], we extend the above threshold behavior in two-way r -BP to all dimensions.

¹ For a graph $G = (V, E)$ we say an event happens asymptotically almost surely (a.a.s.) if it happens with probability $1 - o(1)$ as $|V|$ tends to infinity.

One might relax the above question and ask what is the minimum p for which black color survives (but does not make the whole graph black necessarily). In r -BP on \mathbb{T}_L^d as will be discussed, it is straightforward to show that black color survives forever if $p = \omega(\mathcal{P}_2)$ and it does not if $p = o(\mathcal{P}_2)$ a.a.s. for $\mathcal{P}_2 := L^{-d}$. The answer to this question for two-way r -BP is somewhat more involved. We prove a similar threshold behavior in the two-way setting, which leads into some interesting insights regarding the behavior of the process.

All in all, we prove two-way r -BP on the d -dimensional torus \mathbb{T}_L^d exhibits two phase transitions. More precisely, asymptotically almost surely

- white color takes over if $p = o(\mathcal{P}_2^{1/2^{r-1}})$: Phase 1
- both colors survive if $p = \omega(\mathcal{P}_2^{1/2^{r-1}})$ and $p = o(\mathcal{P}_1^{1/2^{r-1}})$: Phase 2
- black color takes over if $p = \omega(\mathcal{P}_1^{1/2^{r-1}})$: Phase 3.



■ **Figure 1** Two phase transitions in two-way r -BP on \mathbb{T}_L^d .

Note that for $r = 1$, we have $L^{-\frac{d}{2^{r-1}}} = (\log_{(r-1)} L)^{-\frac{d-r+1}{2^{r-1}}} = L^{-d}$, which implies that the process basically goes through one phase transition. In general, it is an easy exercise to prove that for both 1-BP and two-way 1-BP on an n -node graph, black color (resp. white color) takes over a.a.s. if $p = \omega(n^{-1})$ (resp. $p = o(n^{-1})$). Therefore, in the rest of the paper we assume that $r \geq 2$, otherwise we point out explicitly.

It is worth to mention that the threshold behavior of many other similar models have been extensively studied, cf. [18, 11, 40, 12, 30, 10, 39]. The main difference between these models and ours is that they assume that the nodes update their color sequentially, in a deterministic or random fashion.

Outline. After setting up some basic definitions in Section 1.1, we provide some insights and the main ideas behind our proof techniques in Section 1.2. Finally in Section 2, our main results regarding two phase transitions of two-way r -BP on the d -dimensional torus are provided.

1.1 Definitions and Preliminaries

Let for a graph $G = (V, E)$ and a node $v \in V$, the *neighborhood* of v be $N(v) := \{u \in V : \{v, u\} \in E\}$. For a set $S \subseteq V$ we have $N_S(v) := N(v) \cap S$ and $N(S) := \bigcup_{v \in S} N(v)$. Furthermore, for two nodes $v, u \in V$ we define the *distance* $d(v, u)$ to be the length of a shortest path between v and u , in terms of the number of edges. Let G^2 be the second power of graph G , where two nodes are adjacent if their distance in G is at most 2. Then, we say there is a *semi-connected* path between v and u in G if there is a path between them in G^2 .

Formally, a *configuration* is a function $\mathcal{C} : V \rightarrow \{b, w\}$, where b, w stand for black and white. For a configuration \mathcal{C} , node $v \in V$, and color $c \in \{b, w\}$, we define $N_c^{\mathcal{C}}(v) := \{u \in N(v) : \mathcal{C}(u) = c\}$ which is the set of neighbors of v having color c in configuration \mathcal{C} . Finally, for a color $c \in \{b, w\}$ and a set $S \subseteq V$, we write $\mathcal{C}|_S = c$ if $\forall v \in S, \mathcal{C}(v) = c$.

Let us define (two-way) r -BP formally. Consider a graph $G = (V, E)$ and an initial random configuration \mathcal{C}_0 . In two-way r -BP, $\mathcal{C}_t(v) = b$ if $|N_b^{\mathcal{C}_{t-1}}(v)| \geq r$ and $\mathcal{C}_t(v) = w$ otherwise for $t \geq 1$, where \mathcal{C}_t is the t -th configuration. In r -BP, $\mathcal{C}_t(v) = b$ if $|N_b^{\mathcal{C}_{t-1}}(v)| \geq r$ or $\mathcal{C}_{t-1}(v) = b$ and $\mathcal{C}_t(v) = w$ otherwise.

For a graph G and two configurations \mathcal{C} and \mathcal{C}' we write $\mathcal{C} \leq \mathcal{C}'$ if all black nodes in \mathcal{C} are also black in \mathcal{C}' . A model M_1 is *stronger* than model M_2 if for any graph G and any configuration \mathcal{C} , we have $M_2(\mathcal{C}) \leq M_1(\mathcal{C})$ where $M_1(\mathcal{C})$ and $M_2(\mathcal{C})$ denote the configuration obtained from \mathcal{C} after one round of M_1 and M_2 . For instance, r -BP is stronger than two-way r -BP. Furthermore, M is a *monotone* model if for any graph G and any two configurations $\mathcal{C}_1 \leq \mathcal{C}_2$, we have $\mathcal{C}'_1 \leq \mathcal{C}'_2$ where \mathcal{C}'_1 and \mathcal{C}'_2 are the configurations obtained respectively from \mathcal{C}_1 and \mathcal{C}_2 after one round of M . All models introduced in this paper are monotone.

For any model M and a graph $G = (V, E)$, a set $S \subseteq V$ is called a c -robust set for $c \in \{b, w\}$ whenever the following holds: if all nodes in S share color c in some configuration during the process, then they will all keep it in all upcoming configurations. Furthermore, a set $S \subseteq V$ is c -eternal for $c \in \{b, w\}$ means if all nodes in S have color c in some configuration, then color c survives, that is for any upcoming configuration there is a node which has color c . Clearly, a c -robust set is also a c -eternal set, but not necessarily the other way around. Furthermore, we say a node set D is a c -dynamo for $c \in \{b, w\}$ if color c takes over once all nodes in D have color c . For example in any connected graph and two-way 1-BP, any two adjacent nodes are a b -dynamo. Notice one node might not suffice, for instance in an even cycle.

For some graph G and an integer $r \geq 1$, we say a node set S is (r, c) -robust (analogously, (r, c) -eternal, (r, c) -dynamo) if it is c -robust (resp. c -eternal, c -dynamo) in two-way r -BP on G . Observe that in an (r, b) -robust set (analogously (r, w) -robust set) for each node $v \in S$, $|N_S(v)| \geq r$ (resp. $|N_{V \setminus S}(v)| < r$).

The d -dimensional torus \mathbb{T}_L^d is the graph with the node set $V = \{(i_1, \dots, i_d) : 1 \leq i_1, \dots, i_d \leq L\}$ and the edge set $E = \{\{i, i'\} : |i_j - i'_j| = 1, L - 1 \text{ for some } j \text{ and } i_k = i'_k \forall k \neq j\}$. Notice each node in \mathbb{T}_L^d has $2d$ neighbors, two neighbors in each dimension. For a node $v = (i_1, \dots, i_d)$ and $1 \leq j \leq d$, we call $(i_1, \dots, i_j + 1, \dots, i_d)$ and $(i_1, \dots, i_j - 1, \dots, i_d)$ the neighbors of v in the j -th dimension. (For the above definition to make sense when i_j is equal to 1 or L , we need to apply the modulo L operation. However to lighten the notation, we skip that whenever it is clear from the context.)

The *hyper-rectangle* of size $l_1 \times \dots \times l_d$ starting from node (i_1, \dots, i_d) is the node set $\{(i'_1, \dots, i'_d) : i_j \leq i'_j \leq i_j + l_j \forall 1 \leq j \leq d\}$. An r -dimensional *hyper-square* HS starting at node i is a hyper-rectangle starting at i with exactly r of l_j s being equal to 1 and the rest 0, where we define $J_{HS} := \{j : l_j \neq 0\}$. We denote the *odd-part* (analogously *even-part*) of HS by $HS^{(1)}$ (resp. $HS^{(2)}$), which are the nodes that differ in odd (resp. even) number of coordinates with i . As a warm-up let us prove the following simple, however crucial, lemma.

► **Lemma 1.** *For an r -dimensional hyper-square HS in $\mathbb{T}_L^d = (V, E)$, $HS^{(1)}$ and $HS^{(2)}$ are (r, b) -eternal sets.*

Proof. It suffices to show each node in $HS^{(1)}$ has exactly r neighbors in $HS^{(2)}$ and vice versa because it implies that if $\mathcal{C}_t|_{HS^{(1)}} = b$, we will have $\mathcal{C}_{t+2t'+1}|_{HS^{(2)}} = b$ and $\mathcal{C}_{t+2t'}|_{HS^{(1)}} = b$ for any $t' \geq 0$ (a similar argument for $\mathcal{C}_t|_{HS^{(2)}} = b$). Let i' be a node in $HS^{(1)}$ and assume HS starts at node i . There is an odd-size subset $J \subseteq J_{HS}$ of coordinates in which i' is larger than i by one. Now, by decrementing any coordinate in J or incrementing any coordinate in $J_{HS} \setminus J$, we reach a node in $HS^{(2)}$ which is a neighbor of i' . Thus, i' has r neighbors in $HS^{(2)}$. The proof of the other direction is analogous. See Figure 2 for an example. ◀

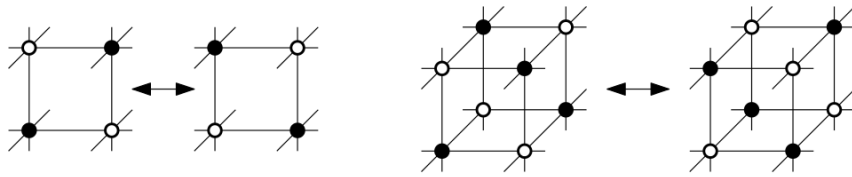


Figure 2 (left) A $(2, b)$ -eternal set (right) a $(3, b)$ -eternal set.

Notice the even/odd-part of an r -dimensional hyper-square is of size 2^{r-1} which implies that there exists an (r, b) -eternal set of size 2^{r-1} . Furthermore, we always assume $1 \leq r \leq d$. The setting of $d + 1 \leq r \leq 2d$ is the same as $1 \leq r \leq d$ if we swap black and white.

In the r -dimensional torus \mathbb{T}_L^r , the set of r -dimensional hyper-squares whose starting node is in $\{(2i_1 - 1, \dots, 2i_r - 1) : 1 \leq i_1, \dots, i_r \leq \lfloor L/2 \rfloor\}$ divide the node set (except the nodes with value L in one of their coordinates if L is odd) into $\lfloor L/2 \rfloor^r$ pair-wise disjoint hyper-squares. Furthermore, if we divide the nodes in the d -dimensional torus \mathbb{T}_L^d into L^{d-r} pair-wise disjoint subsets according to their last $d - r$ coordinates, the induced subgraph by each of these subsets is an r -dimensional torus. Now, if we partition the node set of each of these r -dimensional tori into $\lfloor L/2 \rfloor^r$ hyper-squares as above, we will have $L^{d-r} \lfloor L/2 \rfloor^r = \Theta(L^d)$ pair-wise disjoint r -dimensional hyper-squares. We call this procedure the *tiling* of \mathbb{T}_L^d into r -dimensional hyper-squares.

1.2 Proof Techniques and Some Insights

Phase transition. Intuitively speaking, one might expect any monotone model to exhibit some sort of threshold behavior with two phase transitions on any graph. Assume that the initial probability p is very close to zero, then black nodes probably disappear in a few number of rounds. However, if we gradually increase the initial probability, at some point it would suffice to guarantee the survival of black color, however perhaps it is not high enough to result in a fully black configuration. Finally, if we keep increasing the initial probability, suddenly it should be sufficient to guarantee not only the survival of black color but also the disappearance of white color. Another way of seeing these two phase transitions is in terms of b -eternal set and b -dynamo. One might think of the first threshold as the threshold value for having a fully black b -eternal set and the second one as the threshold for having a fully black b -dynamo since black color survives if and only if there is a black b -eternal set initially and it will take over if and only if there is a black b -dynamo in the initial configuration. Notice the first and the second threshold values might match, which means the process goes actually through one phase transition; for instance, in 1-BP the existence of a black node is the necessary condition for survival of black color and at the same time sufficient condition to take over; i.e., any b -eternal set is also a b -dynamo. Although this threshold behavior might seem conceptually simple, identifying the exact threshold values is usually a very non-trivial task. As we discussed even for the very special case of r -BP on \mathbb{T}_L^d the answer was known after a large series of papers over more than three decades.

As discussed, r -BP on the d -dimensional torus \mathbb{T}_L^d goes through two phase transitions, which matches our intuitive argument from above. More precisely, the torus becomes fully white if $p = o(\mathcal{P}_2)$, both color coexist if $p = \omega(\mathcal{P}_2)$ and $p = o(\mathcal{P}_1)$, and it will become fully black if $p = \omega(\mathcal{P}_1)$ a.a.s. where $\mathcal{P}_1 = (\log_{(r-1)} L)^{-(d-r+1)}$ and $\mathcal{P}_2 = L^{-d}$. The first transition has not been considered before, but it is very easy to handle. For $p = o(\mathcal{P}_2)$, by a simple union bound the probability that there exists a black node in the initial configuration is upper-bounded by $L^d \cdot o(\mathcal{P}_2) = o(1)$, which implies a.a.s. the initial configuration is fully

white. For $p = \omega(\mathcal{P}_2)$, the expected number of black nodes in the initial configuration is equal to $L^d \cdot \omega(\mathcal{P}_2) = \omega(1)$; applying the Chernoff bound [17] yields that a.a.s. there exists a black node initially, which guarantees the survival of black color.

We prove that two-way r -BP on \mathbb{T}_L^d exhibits a similar threshold behavior at threshold values $\mathcal{P}_1^{1/2^{r-1}}$ and $\mathcal{P}_2^{1/2^{r-1}}$. As mentioned, Balogh, Bollobas, Duminil-Copin, and Morris [6] proved that r -BP actually exhibits a sharp threshold behavior in the second transition. Can we expect a sharp threshold behavior in the first transition of r -BP or any of the two transitions of two-way r -BP? We believe that it might be the case in the second phase transition of two-way r -BP, but proving such a statement probably requires novel ideas beyond the known techniques in the literature. On the other hand, the claimed weak threshold behavior is the best possible for the first phase transition in both r -BP and two-way r -BP. (See the appendix, Section A.1, for a simple proof of this claim.)

A more general statement. Recall in Lemma 1 we proved that in two-way r -BP on \mathbb{T}_L^d there is an (r, b) -eternal set of size 2^{r-1} . In Lemma 3 we will show that actually there is no smaller (r, b) -eternal set. Therefore, by switching from r -BP to two-way r -BP, the minimum size of a b -eternal set increases from 1 to 2^{r-1} . Thus, the threshold values in both r -BP and two-way r -BP are equal to $\mathcal{P}_1^{1/s}$ and $\mathcal{P}_2^{1/s}$, where s is the minimum size of a b -eternal set. We believe that there exists a large class of monotone models \mathcal{M} such that each model $M \in \mathcal{M}$ on \mathbb{T}_L^d goes through two phase transitions at threshold values $\mathcal{P}_1^{1/s}$ and $\mathcal{P}_2^{1/s}$, where s is the minimum size of a b -eternal set. We do not prove such a statement, but in Section 3 we illustrate how our proof techniques can possibly be applied to provide such results. For now, let us discuss an interesting example which falls under the umbrella of the above argument. As an intermediate step from the analysis of r -BP to the analysis of two-way r -BP, Coker and Gunderson [16] studied the following variant of bootstrap percolation on \mathbb{T}_L^2 , which is called *2-BP with recovery*. In this model, a white node becomes black if it has at least two black neighbors, and a black node remains unchanged, except if all its four neighbors are white. Coker and Gunderson [16] proved that this process exhibits two phase transitions at $\mathcal{P}_1^{1/2}$ and $\mathcal{P}_2^{1/2}$. Notice this is consistent with the above claim because in 2-BP with recovery the minimum size of a b -eternal set is 2. Clearly, one black node disappears in one round but two adjacent black nodes survive forever.

Proof techniques. Now, we discuss the high-level ideas of the proof techniques applied. In phase one, we want to show that if $p = o(\mathcal{P}_2^{1/2^{r-1}})$ then black color disappears a.a.s. We exploit a technique which we call *clustering*; roughly speaking, we show p is so small that a.a.s. one can partition all black nodes in small clusters which are far from each other. This distance lets us treat each cluster independently since there is no interaction among them. Furthermore, the number of black nodes in each cluster is less than 2^{r-1} , that is the minimum size of an (r, b) -eternal set, which then results in the disappearance of black color.

For the second phase, we must show both colors survive a.a.s. For black color, since there are $\Theta(L^d)$ pair-wise disjoint (r, b) -eternal sets of size 2^{r-1} (namely the even-part of $\Theta(L^d)$ pair-wise disjoint r -dimensional hyper-squares), applying the Chernoff bound implies that there is a black (r, b) -eternal set initially a.a.s. This guarantees the survival of black color. We also need to show for $p = o(\mathcal{P}_1^{1/2^{r-1}})$ white color survives a.a.s. For that, we rely on the threshold behavior of r -BP. More precisely, applying the fact that the minimum size of a b -eternal set is equal to 2^{r-1} , we show the probability that an arbitrary node is black after T rounds, for some constant $T(d, r)$, is $o(\mathcal{P}_1)$. Since the stronger model of r -BP results in the survival of white color a.a.s. in this case, so does two-way r -BP. (Some details are omitted.)

In phase 3, our goal is to prove if $p = \omega(\mathcal{P}_1^{1/2^{r-1}})$, a.a.s. black color takes over. We utilize a method, which we call *scaling*. The idea is to tile the torus \mathbb{T}_L^d into r -dimensional hyper-squares and treat each of the hyper-squares as a single node. We say two hyper-squares are neighbors if there is at least one edge between them; then, each hyper-square has $2d$ neighbors, two in each dimension. Furthermore, we say a hyper-square is *occupied* in configuration \mathcal{C}_t for even t (analogously odd t) if its even-part (resp. odd-part) is black. We prove if in some configuration, a hyper-square has occupied neighbors in r distinct dimensions, then it becomes occupied in constantly many rounds. Furthermore, each hyper-square is occupied initially with probability $p^{2^{r-1}} = \omega(\mathcal{P}_1)$. Hence, the process scaled to the hyper-squares is at least as strong as modified r -BP, where initially each hyper-square is occupied with probability $\omega(\mathcal{P}_1)$. We know modified r -BP with initial probability $\omega(\mathcal{P}_1)$ results in a fully black configuration a.a.s. This implies that two-way r -BP on \mathbb{T}_L^d reaches a configuration where the even-part of each of the hyper-squares is black a.a.s. We can do the same argument by switching the terms of odd and even in the definition of occupation. Then, by a union bound, a.a.s. black color takes over.

Tie-breaking rule. Let us finish this section, by mentioning an interesting observation. Another well-studied model in this literature is the *majority model*, where by starting from an initial random configuration in each round all nodes update their color to the most frequent color in their neighborhood, and in case of a tie, a node keeps its current color. Two-way r -BP on \mathbb{T}_L^d for $r = d$ is sometimes called the *biased majority model* because each node selects the most frequent color in its neighborhood and in case of a tie, it chooses black (notice each node has degree $2d$). Therefore, the majority model on \mathbb{T}_L^d is the same as the biased variant except in tie-breaking rule. We claim in the majority model if $p \leq 1 - \delta$ for any arbitrary constant $\delta > 0$, then black color does not take over a.a.s. For a simple proof see the appendix, Section A.2. On the other hand as we discussed, in the biased model $p = \omega(\mathcal{P}_1^{1/2^{d-1}})$, consequently $p \geq \delta$ for an arbitrarily small constant $\delta > 0$, results in a fully black configuration a.a.s. Putting these two propositions in parallel, we observe that in the majority model p should be very close to 1 to have a high chance of final complete occupancy by black, but by just changing the tie-breaking rule in favor of black, the process ends up in a fully black configuration a.a.s. even for initial probability very close to 0. This comparison illustrates how small alternations in local behavior can result in considerable changes in the global behavior.

2 Two Phase Transitions

2.1 Phase 1

The idea of the proof is to show that if $p = o(\mathcal{P}_2^{1/2^{r-1}})$, then a.a.s. black nodes in \mathcal{C}_0 are contained in a group of hyper-rectangles which are sufficiently far from each other and each hyper-rectangle includes less than 2^{r-1} black nodes. Since the hyper-rectangles are far from each other, the nodes out of the hyper-rectangles, which are all white initially, stay white forever (i.e., create a white (r, w) -robust set). Furthermore, the black nodes inside each hyper-rectangle die out after some rounds because they are less than the minimum size of an (r, b) -eternal set. We prove our claim in Theorem 4, building on Lemma 3. To prove Lemma 3, we first need to provide Lemma 2.

► **Lemma 2.** *In $\mathbb{T}_L^d = (V, E)$, a non-empty (r, b) -robust set intersects at least 2^{r-1} pair-wise disjoint (r, w) -robust sets.*

Proof. We do induction on r . As the base case we show that a $(2, b)$ -robust set S in \mathbb{T}_L^d intersects at least $2^{2-1} = 2$ disjoint $(2, w)$ -robust sets W_1 and W_2 . There exists some coordinate j so that there are two nodes $i^{(1)} = (i_1^{(1)}, \dots, i_d^{(1)})$ and $i^{(2)} = (i_1^{(2)}, \dots, i_d^{(2)})$ in S with $i_j^{(1)} < i_j^{(2)}$ (otherwise S includes only one node, which cannot be a $(2, b)$ -robust set). Let $W_1 = \{(i_1, \dots, i_d) \in V : i_j = i_j^{(1)} \vee i_j = i_j^{(1)} - 1\}$ and $W_2 = V \setminus W_1$. Notice that W_1 includes $i^{(1)}$ and W_2 includes $i^{(2)}$, except if $i_j^{(1)} = 1$ and $i_j^{(2)} = L$, but in this case we simply use $i_j^{(1)} + 1$ instead of $i_j^{(1)} - 1$ in W_1 . For any node $i \in W_1$ (similarly in W_2), $2d - 2$ of neighbors which differ with i only in some coordinate $j' \neq j$ are all in W_1 (resp. W_2) and among the two neighbors which differ in the j -th coordinate at least one of them is in W_1 (resp. W_2) by construction. Thus, each node has at least $2d - 1$ of its $2d$ neighbors in W_1 (resp. W_2), which implies it is a $(2, w)$ -robust set.

Now, as the induction hypothesis assume that the statement is true for some $r \geq 2$, we show it holds also for $r + 1$. Let set S be an $(r + 1, b)$ -robust set. There exists some coordinate j so that there are two nodes in S which differ in the j -th coordinate, otherwise it includes only one node. Let level L_k be the nodes whose j -th coordinate is k for $1 \leq k \leq L$. In other words, level L_k is the node set of the $(d - 1)$ -dimensional torus attained by fixing the j -th coordinate to be k . Based on above, we know there are at least two levels which intersect S . Assume there are $1 \leq k_1, k_2 \leq L$ such that L_{k_1} and L_{k_2} intersect S but L_{k_1+1} and L_{k_2-1} do not and $(L_{k_1} \cup L_{k_1+1}) \cap (L_{k_2-1} \cup L_{k_2}) = \emptyset$. In other words, there are two disjoint pairs and each pair includes two adjacent levels, where one level intersects S and the other one does not. If such pairs do not exist then there are $\Theta(L)$ levels which intersect S . Furthermore, each set $L_{2k-1} \cup L_{2k}$ for $1 \leq k \leq \lfloor L/2 \rfloor$ is an $(r + 1, w)$ -robust set because each node in $L_{2k-1} \cup L_{2k}$ has exactly $2d - 1$ neighbors in it. Based on the last two statements if there do not exist such disjoint pairs of levels, we have $\Theta(L) \geq 2^{(r+1)-1} = 2^r$ pair-wise disjoint $(r + 1, w)$ -robust sets which intersect S , which then we are done. Therefore, assume such disjoint pairs of levels $L_{k_1} \cup L_{k_1+1}$ and $L_{k_2-1} \cup L_{k_2}$ exist.

We define $S_1 := S \cap L_{k_1}$ and $S_2 := S \cap L_{k_2}$; let $(d - 1)$ -dimensional torus \mathbb{T}_1 (similarly \mathbb{T}_2) be the induced subgraph on node set L_{k_1} (resp. L_{k_2}). We claim each node in S_1 (similarly S_2) has at least r neighbors in S_1 (resp. S_2), which means S_1 (resp. S_2) is an (r, b) -robust set with respect to \mathbb{T}_1 (resp. \mathbb{T}_2). We prove the claim for S_1 , and the proof for S_2 is analogous. Each node in L_{k_1} has all its neighbors in L_{k_1} except one in L_{k_1-1} and one in L_{k_1+1} , but the one in L_{k_1+1} is not in S because $L_{k_1+1} \cap S = \emptyset$ by our assumption. Furthermore, each node in S has at least $r + 1$ neighbors in S , which implies that each node in S_1 has at least r neighbors in S_1 . Since S_1 is an (r, b) -robust set with respect to the $(d - 1)$ -dimensional torus \mathbb{T}_1 , it intersects at least 2^{r-1} pair-wise disjoint (r, w) -robust sets in \mathbb{T}_1 by the induction hypothesis. The same argument applies to S_2 with respect to \mathbb{T}_2 . Therefore, there are pair-wise disjoint sets $W_1^{(1)}, \dots, W_{2^{r-1}}^{(1)} \subset L_{k_1}$ (analogously $W_{2^{r-1}+1}^{(1)}, \dots, W_{2^r}^{(1)} \subset L_{k_2}$) such that a node v in $W_\ell^{(1)}$ for $1 \leq \ell \leq 2^{r-1}$ (resp. $2^{r-1} + 1 \leq \ell \leq 2^r$) has at least $2(d - 1) - r + 1$ neighbors in $W_\ell^{(1)}$, based on the definition of an (r, w) -robust set in a $(d - 1)$ -dimensional torus. Now, let set $W_\ell^{(2)}$ for $1 \leq \ell \leq 2^{r-1}$ (similarly $2^{r-1} + 1 \leq \ell \leq 2^r$) be the mapping of set $W_\ell^{(1)}$ into L_{k_1+1} (resp. L_{k_2-1}); that is, we change the j -th coordinate from k_1 (resp. k_2) to $k_1 + 1$ (resp. $k_2 - 1$) for each node in $W_\ell^{(1)}$ to obtain $W_\ell^{(2)}$. Now, we claim $W_\ell := W_\ell^{(1)} \cup W_\ell^{(2)}$ for $1 \leq \ell \leq 2^r$ are 2^r pair-wise disjoint $(r + 1, w)$ -robust sets in \mathbb{T}_L^d which all intersect S . Firstly, each node in $W_\ell^{(1)}$ (similarly $W_\ell^{(2)}$) has at least $2(d - 1) - r + 1 = 2d - r - 1$ neighbors in $W_\ell^{(1)}$ (resp. $W_\ell^{(2)}$) and one neighbor in $W_\ell^{(2)}$ (resp. $W_\ell^{(1)}$), which is overall $2d - r = 2d - (r + 1) + 1$ neighbors in W_ℓ ; this implies that it is an $(r + 1, w)$ -robust set in \mathbb{T}_L^d . Furthermore, they are all disjoint because based on our construction $(L_{k_1} \cup L_{k_1+1}) \cap (L_{k_2-1} \cup L_{k_2}) = \emptyset$. Finally, S intersects each W_ℓ for $1 \leq \ell \leq 2^r$ because it intersects $W_\ell^{(1)}$ based on the induction hypothesis. \blacktriangleleft

► **Lemma 3.** *In two-way r -BP on \mathbb{T}_L^d , a configuration with less than 2^{r-1} black nodes becomes fully white in T rounds for some constant $T(d, r)$.*

Proof. Consider an initial configuration \mathcal{C}_0 which includes less than 2^{r-1} black nodes and denote the set of black nodes in \mathcal{C}_0 with B . Define the distance between two hyper-rectangles HR and HR' to be $d(HR, HR') = \min_{v \in HR, u \in HR'} d(v, u)$. Let HR_1, \dots, HR_k be constant-size hyper-rectangles whose pair-wise distance is at least three and include all black nodes. Notice such a set of hyper-rectangles exists since $|B|$ is a constant. All nodes which are not in the hyper-rectangles are white and remain white forever; that is, they are a white (r, w) -robust set. This is true because a node which is not in the hyper-rectangles is adjacent to at most one hyper-rectangle (otherwise it violates the aforementioned distance property), which implies at most one of its $2d$ neighbors is black. Therefore, only nodes in the hyper-rectangles can switch their color. Since the hyper-rectangles are of constant size, the number of configurations that the process can possibly reach from \mathcal{C}_0 is upper-bounded by some constant $T(d, r)$. That is, the process reaches a cycle of configurations in at most T rounds. Based on the results by Goles and Olivos [25], we know the length of the cycle is one or two.

Let B' be the union of black nodes in the configuration(s) in the cycle; we claim B' is an (r, b) -robust set. Therefore, if B' is non-empty it must intersect at least 2^{r-1} pairwise disjoint (r, w) -robust sets based on Lemma 2. However, initially there are at most $2^{r-1} - 1$ black nodes, which can intersect at most $2^{r-1} - 1$ pairwise disjoint (r, w) -robust sets. Thus, there is at least one (r, w) -robust set which is initially fully white, but at the end includes a black node, which is a contradiction with its (r, w) -robustness. Thus, B' is actually empty.

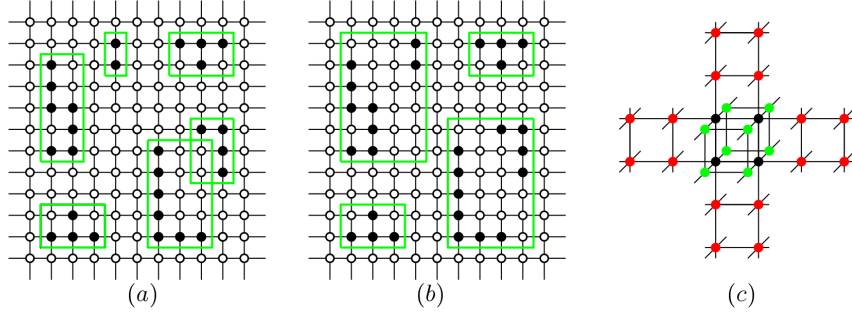
It only remains to show that B' is (r, b) -robust. If the process reaches a cycle of length one, a fixed configuration, trivially the set of black nodes is an (r, b) -robust set. If it reaches a cycle of length two and switches between two configurations \mathcal{C}_1 and \mathcal{C}_2 , we define B'_1 and B'_2 to be the set of black nodes in \mathcal{C}_1 and \mathcal{C}_2 , respectively. The set $B' = B'_1 \cup B'_2$ is (r, b) -robust since each node in B'_1 (similarly B'_2) has at least r neighbors in B'_2 (resp. B'_1), otherwise it cannot be black in \mathcal{C}_1 (resp. \mathcal{C}_2). Therefore, each node in B' has at least r neighbors in B' , which implies that it is an (r, b) -robust set. ◀

► **Theorem 4.** *In two-way r -BP on \mathbb{T}_L^d , white color takes over a.a.s. if $p = o(L^{-d/2^{r-1}})$.*

Proof. Recall that the distance between two hyper-rectangles HR and HR' is equal to $d(HR, HR') = \min_{v \in HR, u \in HR'} d(v, u)$. We show that for the initial configuration a.a.s. there is a set of hyper-rectangles which are pair-wise in distance at least three from each other and any black node belongs to one of these hyper-rectangles and the number of black nodes in each hyper-rectangle is less than 2^{r-1} . Each node which is not in any of the hyper-rectangles is adjacent to at most one of them (otherwise, there are two hyper-rectangles whose distance is less than three). Thus, each of these nodes has at least $2d - 1$ white neighbors which implies they all stay white forever. Furthermore, in each of these “isolated” hyper-rectangles there are less than 2^{r-1} black nodes which disappear after at most T rounds by Lemma 3.

It remains to prove that a.a.s. such a set of hyper-rectangles exist. For each black connected component in \mathcal{C}_0 , consider the smallest hyper-rectangle which includes all its node. Let \mathcal{A}_0 be the set of these (not necessarily disjoint) hyper-rectangles. There is no black connected component of size 2^{r-1} or larger in \mathcal{C}_0 a.a.s. Let X denote the number of black connected subgraphs of size 2^{r-1} in \mathcal{C}_0 . The number of connected subgraphs of size 2^{r-1} which include an arbitrary node v is a constant (notice d , thus also r , is fixed); then, the number of connected subgraphs of size 2^{r-1} is of order $\Theta(L^d)$. Thus, $\mathbb{E}[X] = \Theta(L^d) p^{2^{r-1}} = \Theta(L^d) o(L^{-d}) = o(1)$. By Markov’s inequality [17] a.a.s. there is no black connected subgraph of size 2^{r-1} , which implies that there is no black connected component of this size or larger. Therefore, for any hyper-rectangle of size $l_1 \times \dots \times l_d$ in \mathcal{A}_0 , $l_j < 2^{r-1}$ for all $1 \leq j \leq d$ a.a.s.

Consider the following procedure. By starting from $\mathcal{A} = \mathcal{A}_0$, in each iteration if all hyper-rectangles in \mathcal{A} are pair-wise in distance at least three from each other, the procedure is over, otherwise there are two hyper-rectangles $HR_1, HR_2 \in \mathcal{A}$ such that $d(HR_1, HR_2) \leq 2$. In this case, we set $\mathcal{A} = \mathcal{A} \setminus \{HR_1, HR_2\} \cup \{HR\}$, where HR is the smallest hyper-rectangle which includes all black nodes in both HR_1 and HR_2 . See Figure 3 (a) and (b) for an example, where the boundaries of the smallest hyper-rectangles are distinguished by green. The process definitely terminates, because in each round $|\mathcal{A}|$ decreases. Moreover, when the process is over, the hyper-rectangles in \mathcal{A} satisfy our desired distance property. We still have to show that each of them contains less than 2^{r-1} black nodes. Let us make the three following observations.



■ **Figure 3** (a) the smallest hyper-rectangles (b) after two iterations (c) the inner and outer neighbors.

- (a) Let HR of size $l_1 \times \dots \times l_d$ be the smallest hyper-rectangle which contains all black nodes in both HR_1 and HR_2 respectively of size $l_1^{(1)} \times \dots \times l_d^{(1)}$ and $l_1^{(2)} \times \dots \times l_d^{(2)}$ in the above procedure, we have $l_j \leq 3 \max_{i \in \{1,2\}} l_j^{(i)}$ for all $1 \leq j \leq d$ because $d(HR_1, HR_2) \leq 2$.
- (b) Assume that a hyper-rectangle HR of size $l_1 \times \dots \times l_d$ starting in (i_1, \dots, i_d) is in \mathcal{A} at some iteration in the above procedure, then it contains at least $\max_{1 \leq j \leq d} l_j/2$ black nodes. Intuitively, this should be obvious since in each iteration we combine two hyper-rectangles whose distance is at most two. For a formal proof, let us first show that for any two black nodes v, u in a hyper-rectangle HR in \mathcal{A} , there is a semi-connected path between v and u along the black nodes in HR . We apply proof by induction; initially, this is trivially true since each hyper-rectangle includes a black connected component. Assume in k -th iteration we combine HR_1 and HR_2 because there is node v' in HR_1 and node u' in HR_2 such that $d(v', u') \leq 2$. In the new hyper-rectangle HR , every two black nodes originally from HR_1 (similarly from HR_2) are semi-connected by the induction hypothesis. Two black nodes v and u respectively from HR_1 and HR_2 are also semi-connected along a semi-connected path from v to v' , from v' to u' , and finally from u' to u . Assume $l_{j'} = \max_{1 \leq j \leq d} l_j$; since HR is the smallest hyper-rectangle, there is a black node whose j' -th coordinate is $i_{j'}$ and a black node whose j' -th coordinate is $i_{j'} + l_{j'}$. Consider the semi-connected path between these two nodes which clearly includes at least $l_{j'}/2$ black nodes.
- (c) In \mathcal{C}_0 a.a.s. there is no hyper-rectangle HR of size $l_1 \times \dots \times l_d$ which includes at least 2^{r-1} black nodes and $l_j < 6 \cdot 2^{r-1}$ for all $1 \leq j \leq d$. Let random variable Y denote the number of such hyper-rectangles. The number of hyper-rectangles of the aforementioned sizes starting from a fixed node i is bounded by constant $K = (6 \cdot 2^{r-1})^d$, which implies there are at most KL^d hyper-rectangles of such sizes. Thus, $\mathbb{E}[Y] \leq KL^d \binom{K}{2^{r-1}} p^{2^{r-1}} = o(1)$ for $p = o(L^{-\frac{d}{2^{r-1}}})$, which implies $Y = 0$ a.a.s. by applying Markov's inequality.

At the beginning of the proof we showed that all the sides of any hyper-rectangle in \mathcal{A}_0 are smaller than 2^{r-1} a.a.s. Putting this fact in parallel with (a), we conclude if the process does not terminate while all the sides of any hyper-rectangle in \mathcal{A} are smaller than or equal to $2 \cdot 2^{r-1}$, then it has to generate a hyper-rectangle HR' of size $l'_1 \times \dots \times l'_d$ such that $\forall 1 \leq j \leq d, l'_j \leq 6 \cdot 2^{r-1}$ and there exists $1 \leq j' \leq d$ such that $2 \cdot 2^{r-1} < l'_{j'}$. Based on (b), HR' must include at least $l'_{j'}/2 \geq 2^{r-1}$ black nodes; however, based on (c) such an HR' does not exist a.a.s. Therefore, a.a.s. the process terminates while all the sides of any hyper-rectangle in \mathcal{A} are upper-bounded by $2 \cdot 2^{r-1}$. By applying (c) another time, none of the hyper-rectangles includes 2^{r-1} or more black nodes a.a.s. \blacktriangleleft

2.2 Phase 2

In this section, we prove that in two-way r -BP on $\mathbb{T}_L^d = (V, E)$, if $p = \omega(L^{-\frac{d}{2^{r-1}}})$ and $p = o((\log_{(r-1)} L)^{-\frac{d-r+1}{2^{r-1}}})$, then both colors coexist a.a.s.

Black Color Survives. Let us first show that black color a.a.s. will survive for $p = \omega(L^{-\frac{d}{2^{r-1}}})$. As discussed, in \mathbb{T}_L^d there are L^d/γ pair-wise disjoint r -dimensional hyper-squares, for a constant $\gamma \simeq 2^r$. Consider an arbitrary labeling from 1 to L^d/γ on these hyper-squares and define Bernoulli random variable x_k for $1 \leq k \leq L^d/\gamma$ to be 1 if the even-part of k -th hyper-square is fully black in \mathcal{C}_0 and let $X := \sum_{k=1}^{L^d/\gamma} x_k$. We show $X \neq 0$ a.a.s., which implies that there is a hyper-square whose even-part is fully black initially. Since the even-part of an r -dimensional hyper-square is an (r, b) -eternal set (see Lemma 1), it guarantees the survival of black color. We have $\mathbb{E}[X] = (L^d/\gamma)p^{2^{r-1}} = (L^d/\gamma)\omega(1/L^d) = \omega(1)$, where we used that the even-part of an r -dimensional hyper-square is of size 2^{r-1} . Since X is the sum of independent Bernoulli random variables, we have $Pr[X = 0] \leq \exp(-\omega(1)) = o(1)$ by the Chernoff bound.

White Color Survives. It remains to prove that white color survives a.a.s. if $p = o(\mathcal{P}_1^{1/2^{r-1}})$. Based on Lemma 3, there is a constant T so that by starting from an initial configuration with less than 2^{r-1} black nodes, we have no black nodes after T rounds. We claim this implies that for an arbitrary node v to be black in round T , it needs at least 2^{r-1} black nodes in its T -neighborhood (i.e., nodes in distance at most T from v) in the initial configuration. For the sake of contradiction, assume that there is an initial configuration \mathcal{C}_0 in which v has less than 2^{r-1} black nodes in its T -neighborhood and it is black in the T -th round. Then, we consider the initial configuration \mathcal{C}'_0 in which all nodes in v 's T -neighborhood have the same color as \mathcal{C}_0 and all others are white. Configuration \mathcal{C}'_0 has less than 2^{r-1} black nodes. Furthermore, v must be black after T rounds by starting from \mathcal{C}'_0 because the color of v in round T is only a function of the initial color of nodes in its T -neighborhood (this is easy to see; however, for a formal proof one can simply apply induction) and the color of all nodes in the T -neighborhood of v is the same as \mathcal{C}_0 . However, this is in contradiction with Lemma 3.

So far, we know that for a node v to be black in round T , it needs at least 2^{r-1} black nodes in its T -neighborhood initially. This immediately implies that for an arbitrary node, the probability of being black in round T is upper-bounded by $Kp^{2^{r-1}} = o(\mathcal{P}_1)$, where constant K is an upper-bound on the number of possibilities of choosing 2^{r-1} nodes in the T -neighborhood of an arbitrary node in \mathbb{T}_L^d ; notice that since d and T both are constant, the number of nodes in T -neighborhood of a node is bounded by a constant. Therefore, in round T each node is black with probability $o(\mathcal{P}_1)$. It is known (see Theorem 5) that the stronger model of r -BP results in the survival of white color from such a configuration a.a.s., so does

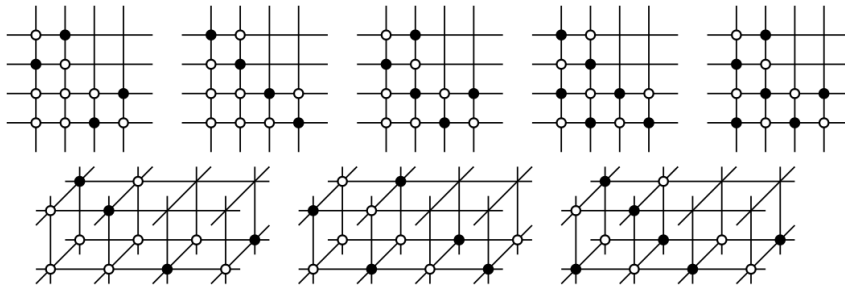
two-way r -BP. The first part of the last statement is not fully correct since in r -BP each node is black independently, but here clearly the color of a node is not independent from the color of nodes in its $2T$ -neighborhood. In the appendix, Section A.3, we show that the proof of Theorem 5 is robust enough to tolerate this level of local dependency.

► **Theorem 5** (Theorem 3.1 in [14]). *In r -BP on \mathbb{T}_L^d if $p = o(\mathcal{P}_1)$, white color survives a.a.s.*

2.3 Phase 3

In this section, we prove that in two-way r -BP on \mathbb{T}_L^d , if $p = \omega(\mathcal{P}_1^{1/2^{r-1}})$ then a.a.s. black color takes over, where $\mathcal{P}_1 = (\log_{(r-1)} L)^{-(d-r+1)}$. For the sake of simplicity, assume L is even (we discuss at the end, how our argument easily carries on the odd case). Recall that the tiling procedure (from Section 1.1) partitions the node set of \mathbb{T}_L^d into $L^d/2^r$ pair-wise disjoint r -dimensional hyper-squares. We say two hyper-squares are neighbors if their distance is equal to one, i.e., there is an edge between them. More precisely, the neighbors of an r -dimensional hyper-square HS starting from $i = (i_1, \dots, i_d)$ are divided into two groups. First, $2r$ hyper-squares whose starting nodes differ with i only in one of the first r coordinates and exactly by two, which are called the *inner* neighbors. The second group are $2(d-r)$ hyper-squares whose starting nodes differ with i in only one of the last $d-r$ coordinates and exactly by one, which are called the *outer* neighbors. The two hyper-squares whose starting nodes differ with i in the j -th coordinate are called the neighbors in the j -th dimension. See Figure 3 (c) for an example of the inner (red) and outer (green) neighbors of a 2-dimensional hyper-square in \mathbb{T}_L^3 . Furthermore, let us define the *parity* of HS to be the parity of the sum of the last $d-r$ coordinates of i . Clearly, the inner neighbors have the same parity as HS but the outer neighbors have different parity.

From now on, we only look at the even rounds; i.e., we only consider \mathcal{C}_t for even t . For an r -dimensional hyper-square of even parity (similarly odd parity), we say it is *occupied* in \mathcal{C}_t if its even-part (resp. odd-part) is black. Based on Lemma 1, an occupied hyper-square remains occupied forever. In Lemma 6, we state that if in some configuration in two-way r -BP on \mathbb{T}_L^d , an r -dimensional hyper-square has occupied neighbor in at least r distinct dimensions then it becomes occupied in constantly many rounds. The proof is technical and is presented in the appendix, Section A.4. The idea is to apply induction on r . See Figure 4, for two examples on how a 2-dimensional hyper-square becomes occupied with occupied neighbors in two distinct dimensions (regarding the selection of black nodes, recall that the parity of a hyper-square is the same as its inner neighbors but different with outer ones).



■ **Figure 4** (top) two inner occupied neighbors (bottom) one inner and one outer occupied neighbor.

► **Lemma 6.** *In two-way r -BP on \mathbb{T}_L^d , if an r -dimensional hyper-square has occupied neighbor in at least r distinct dimensions, it becomes occupied in t' rounds for some even constant t' .*

For our proof, we also need that modified r -BP on \mathbb{T}_L^d with initial probability $\omega(\mathcal{P}_1)$ results in a fully black configuration a.a.s. However, this is known only for $d = r$ by Holroyd [28]. He showed that the process exhibits a sharp threshold behavior at $\lambda' \mathcal{P}_1$ for some constant $\lambda'(d) > 0$. We require a much weaker statement; that is, the initial probability $\omega(\mathcal{P}_1)$ a.a.s. results in a fully black configuration, but for all values of $r \leq d$. The good news is that the upper bound proof by Cerf and Manzo [14] regarding r -BP can be easily adapted to prove our desired upper bound for modified r -BP. Actually, exactly the same proof works because wherever they apply r -BP rule, modified r -BP suffices. However, it is interesting by its own sake to study the sharp threshold behavior of modified r -BP also for $r \neq d$, in future work.

► **Theorem 7** (derived from Theorem 3.1 in [14]). *In modified r -BP on \mathbb{T}_L^d for $r \leq d$, if $p = \omega(\mathcal{P}_1)$, then black color takes over.*

Now, it is time to put the aforementioned claims together to finish the proof. If we tile \mathbb{T}_L^d into hyper-squares as above, in two-way r -BP with $p = \omega(\mathcal{P}_1^{1/2^{r-1}})$ each hyper-square is occupied initially with probability $\omega(\mathcal{P}_1)$. Furthermore, based on Lemma 6 if a hyper-square has occupied neighbor in at least r distinct dimensions, it becomes occupied, which implies the occupation process among the hyper-squares is at least as strong as modified r -BP. Based on Theorem 7, we know that modified r -BP with initial probability $\omega(\mathcal{P}_1)$ becomes fully black a.a.s. Thus, all the hyper-squares become occupied in our process a.a.s. We can do the same argument by just switching the terms of even and odd in the definition of occupation. Then, by a union bound, a.a.s. for two-way r -BP on \mathbb{T}_L^d with $p = \omega(\mathcal{P}_1^{1/2^{r-1}})$ eventually both the even-part and odd-part of all the hyper-squares are black, which implies that black color takes over.

We assumed at the beginning that L is even. Theorem 7 also works for the d -dimensional lattice $[L]^d$. Therefore, for odd L we can do the same argument for the lattice, attained by skipping the nodes with at least one coordinate equal to L , and also the lattice, attained by skipping the nodes with at least one coordinate equal to one. Then, again a union bound finishes the proof.

3 Future Work

We proved that two-way r -BP on \mathbb{T}_L^d exhibits a threshold behavior with two phase transitions at $\mathcal{P}_1^{1/s}$ and $\mathcal{P}_2^{1/s}$ where s is the minimum size of a b -eternal set. The question, then, arises: Can one prove such results for a larger class of models? We introduce a sub-class of monotone models on the d -dimensional torus and then explain how one can possibly employ our proof techniques to prove the desired threshold behavior in this more general framework.

In (r, r') -BP on \mathbb{T}_L^d and for $0 \leq r' \leq r \leq d$, by starting from an initial random configuration in discrete-time rounds each white node becomes black if and only if it has at least r black neighbors and each black node remains black if and only if it has at least r' black neighbors. This includes $\sum_{r=0}^d \sum_{r'=0}^r 1 = (d+1)(d+2)/2$ different models on \mathbb{T}_L^d . For instance, $(r, 0)$ -BP, (r, r) -BP, and $(r, 1)$ -BP are respectively the same as r -BP, two-way r -BP, and r -BP with recovery. It is an interesting exercise to check that (r, r') -BP for $0 \leq r' \leq r$ includes all monotone models where each node updates its color in each round based on its own color and the cardinality of black/white nodes in its neighborhood. We add the constraint $r \leq d$ to make sure that the model includes a constant-size b -eternal set (and no constant-size w -eternal set). Note that monotonicity of the model and constant-size b -eternal set are inseparable parts of our proof techniques.

Now, we illustrate by applying our proof techniques, some prior results, and some novel ideas one can possibly prove that (r, r') -BP on \mathbb{T}_L^d goes through two phase transitions at $\mathcal{P}_1^{1/s}$ and $\mathcal{P}_2^{1/s}$, for s being the minimum size of a b -eternal set. Notice that for $0 \leq r' \leq r \leq d$, an r' -dimensional hyper-square is a b -eternal set, which implies that s is a constant smaller than $2^{r'}$. We assume that $r \geq 2$.

- Phase 1: We can show that white color takes over if $p = o(\mathcal{P}_2^{1/s}) = o(L^{-d/s})$, by replacing 2^{r-1} with s in the proof of Theorem 4, where the clustering technique is applied.
- Phase 2: There are $\Theta(L^d)$ pair-wise disjoint r' -dimensional hyper-squares and each of them includes a b -eternal set of size s . For $p = \omega(L^{-d/s}) = \omega(\mathcal{P}_2^{1/s})$, in expectation $\Theta(L^d)p^s = \omega(1)$ of these b -eternal sets are fully black in the initial configuration. Therefore, by applying the Chernoff bound a.a.s. there is a fully black b -eternal set in the initial configuration. For $p = o(\mathcal{P}_1^{1/s})$, employing our argument from Section 2.2 implies that after a constant number of rounds, each node is black with probability $\Theta(p^s) = o(\mathcal{P}_1)$. We know that the stronger model of r -BP results in the survival of white color from such a configuration a.a.s., so does (r, r') -BP. (Again, we clearly have the dependency issue, which can be handled similarly.)
- Phase 3: We can apply the scaling technique by tiling the torus into r' -dimensional hyper-squares. However, to “reduce” this scaled process to modified r -BP, we need some knowledge about the structure of the b -eternal sets in addition to the value of s . We believe that one can extract sufficient structural properties such as symmetry from the definition of (r, r') -BP, but this is left for future work.

In the present paper, we studied the random setting, but from an extremal point of view it is natural to ask: What is the minimum number of nodes which must be black initially to make the whole graph black? This question has been studied extensively for both r -BP and two-way r -BP on \mathbb{T}_L^d , see e.g. [8, 19, 3, 36, 27, 32], and some lower and upper bounds are known. Can our proof techniques be used to improve on these bounds?

It is also interesting to study the *expected consensus time* of the process, which is the expected number of rounds the process needs to reach a cycle of configurations for an initial random configuration. We are not aware of any result for two-way r -BP on \mathbb{T}_L^d , and for r -BP, the answer is known only for $d = 2$, by Balister, Bollobas, and Smith [4].

References

- 1 Joan Adler and Amnon Aharony. Diffusion percolation. I. Infinite time limit and bootstrap percolation. *Journal of Physics A: Mathematical and General*, 21(6):1387, 1988.
- 2 Michael Aizenman and Joel L Lebowitz. Metastability effects in bootstrap percolation. *Journal of Physics A: Mathematical and General*, 21(19):3801, 1988.
- 3 Paul Balister, Béla Bollobás, J Robert Johnson, and Mark Walters. Random majority percolation. *Random Structures & Algorithms*, 36(3):315–340, 2010.
- 4 Paul Balister, Béla Bollobás, and Paul Smith. The time of bootstrap percolation in two dimensions. *Probability Theory and Related Fields*, 166(1-2):321–364, 2016.
- 5 József Balogh and Béla Bollobás. Bootstrap percolation on the hypercube. *Probability Theory and Related Fields*, 134(4):624–648, 2006.
- 6 József Balogh, Béla Bollobás, Hugo Duminil-Copin, and Robert Morris. The sharp threshold for bootstrap percolation in all dimensions. *Transactions of the American Mathematical Society*, 364(5):2667–2701, 2012.
- 7 József Balogh, Béla Bollobás, and Robert Morris. Bootstrap percolation in three dimensions. *The Annals of Probability*, pages 1329–1380, 2009.
- 8 József Balogh and Gábor Pete. Random disease on the square grid. *Random Structures and Algorithms*, 13(3-4):409–422, 1998.

- 9 József Balogh and Boris G Pittel. Bootstrap percolation on the random regular graph. *Random Structures & Algorithms*, 30(1-2):257–286, 2007.
- 10 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.
- 11 George Barmpalias, Richard Elwes, and Andy Lewis-Pye. Tipping points in 1-dimensional Schelling models with switching agents. *Journal of Statistical Physics*, 158(4):806–852, 2015.
- 12 Christina Brandt, Nicole Immorlica, Gautam Kamath, and Robert Kleinberg. An analysis of one-dimensional Schelling segregation. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 789–804. ACM, 2012.
- 13 Raphaël Cerf and Emilio NM Cirillo. Finite size scaling in three-dimensional bootstrap percolation. *Annals of probability*, pages 1837–1850, 1999.
- 14 Raphaël Cerf and Francesco Manzo. The threshold regime of finite volume bootstrap percolation. *Stochastic Processes and their Applications*, 101(1):69–82, 2002.
- 15 Ching-Lueh Chang and Yuh-Dauh Lyuu. Bounding the sizes of dynamic monopolies and convergent sets for threshold-based cascades. *Theoretical Computer Science*, 468:37–49, 2013.
- 16 Tom Coker and Karen Gunderson. A sharp threshold for a modified bootstrap percolation with recovery. *Journal of Statistical Physics*, 157(3):531–570, 2014.
- 17 Devdatt P Dubhashi and Alessandro Panconesi. *Concentration of measure for the analysis of randomized algorithms*. Cambridge University Press, 2009.
- 18 Richard Durrett, Jeffrey E Steif, et al. Fixation results for threshold voter systems. *The Annals of Probability*, 21(1):232–247, 1993.
- 19 Paola Flocchini, Elena Lodi, Fabrizio Luccio, Linda Pagli, and Nicola ro. Dynamic monopolies in tori. *Discrete applied mathematics*, 137(2):197–212, 2004.
- 20 F Fogelman, Eric Goles, and Gérard Weisbuch. Transient length in sequential iteration of threshold functions. *Discrete Applied Mathematics*, 6(1):95–98, 1983.
- 21 Silvio Frischknecht, Barbara Keller, and Roger Wattenhofer. Convergence in (social) influence networks. In *International Symposium on Distributed Computing*, pages 433–446. Springer, 2013.
- 22 Juan P Garrahan, Peter Sollich, and Cristina Toninelli. Kinetically constrained models. *Dynamical heterogeneities in glasses, colloids, and granular media*, 150:111–137, 2011.
- 23 Bernd Gärtner and Ahad N Zehmakan. (Biased) Majority Rule Cellular Automata. *arXiv preprint*, 2017. [arXiv:1711.10920](https://arxiv.org/abs/1711.10920).
- 24 Bernd Gärtner and Ahad N Zehmakan. Majority Model on Random Regular Graphs. *Latin American Symposium on Theoretical Informatics*, pages 572–583, 2018.
- 25 Eric Goles and Jorge Olivos. Periodic behaviour of generalized threshold functions. *Discrete mathematics*, 30(2):187–189, 1980.
- 26 Mark Granovetter. Threshold models of collective behavior. *American journal of sociology*, 83(6):1420–1443, 1978.
- 27 Lianna Hambardzumyan, Hamed Hatami, and Yingjie Qian. Polynomial method and graph bootstrap percolation. *arXiv preprint*, 2017. [arXiv:1708.04640](https://arxiv.org/abs/1708.04640).
- 28 Alexander Holroyd et al. The metastability threshold for modified bootstrap percolation in d dimensions. *Electronic Journal of Probability*, 11:418–433, 2006.
- 29 Alexander E Holroyd. Sharp metastability threshold for two-dimensional bootstrap percolation. *Probability Theory and Related Fields*, 125(2):195–224, 2003.
- 30 Nicole Immorlica, Robert Kleinberg, Brendan Lucier, and Morteza Zadomighaddam. Exponential segregation in a two-dimensional schelling model with tolerant individuals. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 984–993. SIAM, 2017.
- 31 Svante Janson, Tomasz Łuczak, Tatyana Turova, Thomas Vallier, et al. Bootstrap percolation on the random graph $G_{\{n,p\}}$. *The Annals of Applied Probability*, 22(5):1989–2047, 2012.
- 32 Clemens Jeger and Ahad N Zehmakan. Dynamic monopolies in two-way bootstrap percolation. *Discrete Applied Mathematics*, 2019.

- 33 Yashodhan Kanoria, Andrea Montanari, et al. Majority dynamics on trees and the dynamic cavity method. *The Annals of Applied Probability*, 21(5):1694–1748, 2011.
- 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*, pages 137–146. ACM, 2003.
- 35 Jane Molofsky, Richard Durrett, Jonathan Dushoff, David Griffeath, and Simon Levin. Local frequency dependence and global coexistence. *Theoretical population biology*, 55(3):270–282, 1999.
- 36 Natasha Morrison and Jonathan A Noel. Extremal bounds for bootstrap percolation in the hypercube. *Journal of Combinatorial Theory, Series A*, 156:61–84, 2018.
- 37 Elchanan Mossel, Joe Neeman, and Omer Tamuz. Majority dynamics and aggregation of information in social networks. *Autonomous Agents and Multi-Agent Systems*, 28(3):408–429, 2014.
- 38 Ahad N Zehmakan. Opinion Forming in Erdős-Rényi Random Graph and Expanders. In *29th International Symposium on Algorithms and Computation (ISAAC 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.
- 39 Hamed Omidvar and Massimo Franceschetti. Self-organized Segregation on the Grid. *Journal of Statistical Physics*, 170(4):748–783, 2018.
- 40 Hamed Omidvar and Massimo Franceschetti. Shape of diffusion and size of monochromatic region of a two-dimensional spin system. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 100–113. ACM, 2018.
- 41 David Peleg. Local majority voting, small coalitions and controlling monopolies in graphs: A review. In *Proc. of 3rd Colloquium on Structural Information and Communication Complexity*, pages 152–169, 1997.
- 42 Roberto H Schonmann. Finite size scaling behavior of a biased majority rule cellular automaton. *Physica A: Statistical Mechanics and its Applications*, 167(3):619–627, 1990.
- 43 Roberto H Schonmann. On the behavior of some cellular automata related to bootstrap percolation. *The Annals of Probability*, pages 174–193, 1992.

A Appendix

A.1 No Sharp Threshold in First Transition

In r -BP on \mathbb{T}_L^d , the process exhibits a sharp threshold behavior in the second phase transition; that is, if $p \geq (1 + \epsilon) \lambda \mathcal{P}_1$ (analogously $p \leq (1 - \epsilon) \lambda \mathcal{P}_1$) for some fixed constant $\lambda(d, r) > 0$ black color takes over (resp. does not) a.a.s. for any constant $\epsilon > 0$. We claim that one cannot expect such a behavior in the first transition. We show that if $p = \mu \mathcal{P}_2$ for any constant μ , then black color survives with some non-zero constant probability (which implies that there is no constant μ' such that $p \leq (1 - \epsilon) \mu' \mathcal{P}_2$ results in a fully white configuration a.a.s. for any constant $\epsilon > 0$). The probability that all nodes are white initially is equal to $(1 - p)^{L^d}$ which is smaller than $\exp(-pL^d) = \exp(-\mu)$ for $p = \mu \mathcal{P}_2$, where we used the estimate $1 - x \leq \exp(-x)$. Thus, there is a black node initially with a non-zero constant probability.

Now by applying a similar argument, we show that one cannot also expect a sharp threshold behavior in the first phase transition of two-way r -BP on \mathbb{T}_L^d . We prove that if $p = \mu \mathcal{P}_2^{1/2^{r-1}}$ for any constant $\mu > 0$, then black color has a constant non-zero probability to survive. There are L^d/γ pair-wise-disjoint r -dimensional hyper-squares in \mathbb{T}_L^d for some constant $\gamma > 0$ and based on Lemma 1 the even-part of an r -dimensional hyper-square is an (r, b) -eternal set. The probability that the even-part of at least one of these disjoint hyper-square is black initially is equal to $1 - (1 - p^{2^{r-1}})^{L^d/\gamma}$. Again by applying the estimate $1 - x \leq \exp(-x)$, this probability is lower-bounded by $1 - \exp(-\frac{\mu^{2^{r-1}}}{\gamma})$, which is a non-zero constant.

A.2 Majority Model

► **Theorem 8.** *In the majority model on \mathbb{T}_L^d if $p \leq 1 - \delta$ for any constant $\delta > 0$, then white color survives a.a.s.*

Proof. There are L^d/γ pair-wise disjoint d -dimensional hyper-squares in \mathbb{T}_L^d for some constant $\gamma \simeq 2^d$ as discussed at the end of Section 1.1. Let us label them from 1 to L^d/γ and define Bernoulli random variable x_k to be one if the k -th hyper-square is fully white in the initial configuration for $1 \leq k \leq L^d/\gamma$. Let $X := \sum_{k=1}^{L^d/\gamma} x_k$. Since each hyper-square has 2^d nodes, $\mathbb{E}[X] = \frac{L^d}{\gamma} (1-p)^{2^d} = \Omega(L^d)$ for $p \leq 1 - \delta$. Since x_k s are independent, by applying the Chernoff bound a.a.s. there is a fully white d -dimensional hyper-square initially, which is a w -eternal set in the majority model. This is true because as we argued in Lemma 1 each node in a d -dimensional hyper-square HS has exactly d neighbors in HS . ◀

A.3 Locally Dependent r -BP

We want to prove that in two-way r -BP on \mathbb{T}_L^d if $p = o(\mathcal{P}_1^{1/2^{r-1}})$ then a.a.s. white color survives forever. In Section 2.2, using Lemma 3 we showed that there is a constant $T(d, r)$ such that for each node to be black in the T -th round, it needs to have at least 2^{r-1} black nodes in its T -neighborhood initially. Let us introduce a new process on \mathbb{T}_L^d . Assume that the initial configuration is obtained in the following way: first we make each node black independently with probability $p^{1/2^{r-1}}$, and then each node will be assigned black color if it has at least 2^{r-1} black nodes in its T -neighborhood and white otherwise. Starting from such initial configuration, in each round a white node becomes black if it has at least r black neighbors and black nodes remain unchanged. We call this process *locally dependent r -BP*. Clearly, if we prove that in locally dependent r -BP on \mathbb{T}_L^d for $p = o(\mathcal{P}_1)$, a.a.s. white color survives, then we are done due to the monotonicity of two-way r -BP. To prove this statement, we rely on the results by Cerf and Manzo [14] who showed that in r -BP on \mathbb{T}_L^d , white color will survive forever a.a.s. if $p = o(\mathcal{P}_1)$. We show that a careful treatment of their proof results in the same statement for locally dependent r -BP. Note that the setting of r -BP with $p = o(\mathcal{P}_1)$ is the same as locally dependent r -BP with $p = o(\mathcal{P}_1)$. Firstly, they follow the same updating rule. Secondly, each node is black initially with probability $o(\mathcal{P}_1)$. This is trivial in r -BP and it is true in locally dependent r -BP since the number of possibilities of choosing 2^{r-1} nodes in the T -neighborhood of an arbitrary node in \mathbb{T}_L^d is bounded by a constant, where we use that r, d , and T are constants. The only difference is that in r -BP each node is black independently from all other nodes, but in locally dependent r -BP each node is black independently from all nodes which are not in its $2T$ -neighborhood. (Two nodes which are in distance $2T$ or smaller are not independent since their T -neighborhood overlaps.)

► **Theorem 9** (derived from Theorem 3.1 in [14]). *In locally dependent r -BP on \mathbb{T}_L^d , white color will survive forever a.a.s. if $p = o(\mathcal{P}_1)$.*

Cerf and Manzo [14] proved the statement of Theorem 9 for r -BP. However, basically the same proof with some small changes can be applied to prove Theorem 9. The main ingredients of their proof are two lemmata, Lemma 5.1 and Lemma 5.2 in their paper. The proof of Lemma 5.2 (originally proved by Aizenman and Lebowitz [2]) and how to combine these two lemmata to prove the final statement is quite straightforward and does not use the independence in the initial configuration. Thus it remains to show that the statement of Lemma 5.2 is also true for locally dependent r -BP, which we present in Lemma 10.

For (locally dependent) r -BP on the d -dimensional torus $\mathbb{T}_L^d = (V, E)$ and two nodes $v, u \in V$, let $Pr[v \xrightarrow{p,r} u \text{ in } \mathbb{T}_L^d]$ be the probability that there is a path between v and u along the black nodes in the final configuration. We define $m_-(d, r, p) := \exp^{(r-2)} \left(\beta(d, r) p^{-\frac{1}{d-r+1}} \right)$, where $\exp^{(r)}(x) = \exp(\exp^{(r-1)}(x))$ and $\exp^{(0)}(x) = x$.

► **Lemma 10** (derived from Lemma 5.2 in [14]). *In locally dependent r -BP for $3 \leq r \leq d$: there exist $\beta(d, r) > 0$, $\gamma(d, r) > 0$, $p(d, r) > 0$ such that $\forall p < p(d, r)$ and $\forall m \leq m_-(d, r, p)$, we have $Pr[v \xrightarrow{p,r} u \text{ in } \mathbb{T}_m^d] \leq p^{\gamma \|v-u\|_\infty}$, where $\|\cdot\|_\infty$ denotes the infinity norm and v, u are two nodes in \mathbb{T}_m^d .*

Since the proof of Lemma 5.2 in [14] is quite long, we do not reproduce the whole proof here. Instead, we point out how it should be changed in certain parts, where the independence in the initial configuration is used.

Consider (locally dependent) r -BP on $\mathbb{T}_m^d = (V, E)$, where $V = \{v_1, \dots, v_{m^d}\}$. Define Bernoulli random variable x_k for $1 \leq k \leq m^d$ to be 1 if and only if node v_k is white in the initial configuration. Since a single black node in the initial configuration suffices to make the whole torus black for $r = 1$, we have

$$Pr[v \xrightarrow{p,1} u \text{ in } \mathbb{T}_m^d] = 1 - Pr\left[\bigwedge_{k=1}^{m^d} x_k = 1\right].$$

In r -BP, this probability is equal to $1 - (1-p)^{m^d}$ because each node is white independently with probability $1-p$. Since locally dependent r -BP does not enjoy the independence in the initial configuration, we cannot apply the same argument. However, we have

$$Pr[v \xrightarrow{p,1} u \text{ in } \mathbb{T}_m^d] = 1 - Pr\left[\bigwedge_{k=1}^{m^d} x_k = 1\right] = 1 - \prod_{k=1}^{m^d} Pr[x_k = 1 \mid \bigwedge_{k'=1}^{k-1} x_{k'} = 1].$$

We know that the whiteness of different nodes are positively correlated; that is, the probability of a node v being white in the initial configuration does not decrease if we know that some other nodes are white in the initial configuration. Therefore, $Pr[x_k = 1 \mid \bigwedge_{k'=1}^{k-1} x_{k'} = 1] \geq Pr[x_k = 1]$. Since each node is black initially with probability p , we get $Pr[v \xrightarrow{p,1} u \text{ in } \mathbb{T}_m^d] \leq 1 - (1-p)^{m^d}$. As we will see later, this upper bound is all we need.

Then, they consider the case of $r = 2$. They prove that in r -BP there exist $\beta(d, 2) > 0$, $C > 0$ and $p(d, 2) > 0$ such that $\forall p < p(d, 2)$ and $\forall m < m_-(d, 2, p)$ the probability $Pr[v \xrightarrow{p,2} u \text{ in } \mathbb{T}_m^d]$ is at most $(C \|v-u\|_\infty^{d-1} p)^{\|v-u\|_\infty/2}$. The idea of the proof is as follows. Consider an integer $m \leq m_-(d, 2, p) = \beta(d, 2) p^{-\frac{1}{d-1}}$. Let v be a d -dimensional vector; we denote by \underline{v} its first $d-1$ coordinates and by \bar{v} the last one and write $v = (\underline{v}, \bar{v})$. By symmetry, one can assume that (\underline{v}, \bar{v}) and (\underline{u}, \bar{u}) are such that $\bar{u} - \bar{v} = \|(\underline{v}, \bar{v}) - (\underline{u}, \bar{u})\|_\infty$. Consider the *slices*

$$T_i := \{(\underline{v}, \bar{v}) \in \mathbb{T}_m^d : \bar{v} \in \{2i, 2i+1\}\} \text{ for } i \in \mathbb{Z}.$$

Suppose that there is a path along black nodes from v to u in the final configuration. Let \mathcal{C} be the maximal connected set of black nodes in the final configuration which include v and u . Let a and b be the first and the last indices of the slices intersecting \mathcal{C} . In all the slices T_i for $i \in [a, b]$ there exists at least one black node (\underline{w}, \bar{w}) such that $\|\underline{v} - \underline{w}\|_\infty \leq \|v - u\|_\infty$. The probability of this to happen in one fixed slice in r -BP is less than $1 - (1-q)^{(2\|v-u\|_\infty+1)^{d-1}}$ where $q = 2p - p^2$. (Here, the estimate is similar to the $r = 1$ estimate from above.) Furthermore, the slices being independent, one gets

$$Pr[v \xrightarrow{p,1} u \text{ in } \mathbb{T}_m^d] \leq d \left(1 - (1-p)^{2(2\|v-u\|_\infty+1)^{d-1}}\right)^{\|v-u\|_\infty/2} \quad (1)$$

where the factor d comes from the possible directions where $\|v - u\|_\infty$ is realized and we used $1 - q = 1 - 2p + p^2 = (1 - p)^2$. (Let us mention that the probability q is not necessarily equal to $2p - p^2$ in locally dependent r -BP, but it is bounded by p and $2p$; we will use this fact later.)

In locally dependent r -BP by applying our argument from above for $r = 1$, we have that the probability that there exists at least one black node (\underline{w}, \bar{w}) such that $\|\underline{v} - \underline{w}\|_\infty \leq \|v - u\|_\infty$ in one fixed slice is less than $1 - (1 - p)^{2(2\|v - u\|_\infty + 1)^{d-1}}$. In contrast to r -BP, locally independent r -BP does not enjoy the independence of the slices, but we can consider $\|v - u\|_\infty / \alpha_1$ slices which are independent for some constant $\alpha_1 > 0$. Therefore, in locally dependent r -BP, we get

$$Pr[v \xrightarrow{p,1} u \text{ in } \mathbb{T}_m^d] \leq d \left(1 - (1 - p)^{2(2\|v - u\|_\infty + 1)^{d-1}} \right)^{\|v - u\|_\infty / \alpha_1}. \quad (2)$$

Cerf and Manzo show that the right hand side of Equation (1) can be upper-bounded by $(C\|v - u\|_\infty^{d-1} p)^{\|v - u\|_\infty / 2}$ for some constant $C > 0$. Applying basically the same calculations on the right hand side of Equation (2) yields a similar upper bound in locally dependent r -BP. Using the estimate $1 - \exp(x) \leq -x$ implies that

$$Pr[v \xrightarrow{p,1} u \text{ in } \mathbb{T}_m^d] \leq d \left(-2(2\|v - u\|_\infty + 1)^{d-1} \ln(1 - p) \right)^{\|v - u\|_\infty / \alpha_1}.$$

For p small, $\ln(1 - p) \geq -2p$ and hence we have

$$Pr[v \xrightarrow{p,2} u \text{ in } \mathbb{T}_m^d] \leq (C\|v - u\|_\infty^{d-1} p)^{\|v - u\|_\infty / \alpha_1}.$$

Therefore, in locally dependent r -BP we get the same upper bound as r -BP except that 2 is replaced by α_1 . We will see that this is all we need to prove our statement.

For $r \geq 3$, they apply an induction on the dimension d and on the parameter r . First, they modify the initial configuration by adding some black nodes and they assume that some nodes become black if they have at least $r - 1$ black neighbors instead of r . Such assumptions can be made due to the monotonicity of the process. They decompose the event $\{v \xrightarrow{p,r} u \text{ in } \mathbb{T}_m^d\}$. This upper-bounds the probability $Pr[v \xrightarrow{p,r} u \text{ in } \mathbb{T}_m^d]$ by the sum of the probability of some events of the following form: a particular set of slices must be fully black but not the slices in between, and the fully black slices are connected by some paths along black nodes. To compute the probability of such events, they utilize the independence of the slices. In locally dependent r -BP the slices are not independent, but we can consider a constant fraction of the slices which must be fully black such that they are independent, i.e., they are in distance at least $2T$ from each other. This is still problematic since for three selected slices T_i , $T_{i'}$, and $T_{i''}$ the event that there is a black path connecting T_i to $T_{i'}$ and the event that there is a black path connecting $T_{i'}$ and $T_{i''}$ are not independent. To deal with this issue, we only consider the events for the connecting black paths one by one. This changes the exponent of our desired probability by a constant factor, similar to the case of $r = 2$. By following their calculations, one can see that the effect of these constant factors appears in the choice of constant γ in the statement of Lemma 10. That is, the inequality $Pr[v \xrightarrow{p,r} u \text{ in } \mathbb{T}_m^d] \leq p^{\gamma\|v - u\|_\infty}$ holds for a smaller value of γ .

Due to several lengthy calculations, we do not reproduce the whole proof. Basically, there are two small changes which one has to do to make the proof work for the locally dependent variant.

- Firstly, at the end of page 80 they apply the result for $r = 2$ which we discussed above, see Equation (1). By some simplifications, they reach an upper bound of form C_2q , where $C_2 > 0$ is a constant and $q = p^2 - 2p$. By applying Equation (2) instead of Equation (1) and using the fact that q is in the same magnitude as p , we get an upper bound of form C'_2p . (One needs to split the sum at the end of page 80 from an integer larger than 9.)
- As we discussed above, to get rid of the dependency among the slices, we choose a constant fraction of them. Therefore, in the calculations at the end of page 81, instead of k we have k/α_2 for some constant $\alpha_2 > 0$.

Both aforementioned constant factors can be hidden in constant γ in the last line of calculations in page 81.

A.4 Proof of Lemma 6

Let us first set up some definitions. For an r -dimensional hyper-square HS starting from $i = (i_1, \dots, i_d)$, the set of nodes in HS whose j -th coordinate is equal to i_j (similarly $i_j + 1$) induce an $(r - 1)$ -dimensional hyper-square, which is called a *face* of HS ; specifically the $(r - 1)$ -dimensional hyper-squares attained by fixing the r -th coordinate to be i_r and $i_r + 1$ are respectively called the *upper face* and *lower face*. Furthermore, the two outer neighbors of HS in the j -th coordinate, where by definition j is among the last $d - r$ coordinates, are simply attained by increasing or decreasing the j -th coordinate of all nodes in HS by one. This implies that if the even-part (odd-part) of one of the outer neighbors of HS , say HS' , is black in some configuration, then each node in the even-part (resp. odd-part) of HS has a black neighbor in HS' in that configuration. In other words, if HS' is occupied, then the upper face (similarly lower face) of HS has an occupied neighbor in the j -th dimension. Similarly, if an inner neighbor of HS in the j -th dimension for $1 \leq j < r$, say HS'' , is occupied, then the upper face of HS (similarly lower face) has one occupied neighbor, namely the upper face (resp. lower face) of HS'' . However, this is not the case for $j = r$. One of the inner neighbors in the r -th dimension has its upper face adjacent to the lower face of HS (which implies if this neighbor is occupied, then the lower face of HS has an occupied neighbor in the r -th dimension) and the other one has its lower face adjacent to the upper face of HS (which provides an occupied neighbor in the r -th dimension for the upper face of HS if it is occupied).

We prove our claim by induction on r . As the base case, we prove that for $r = 2$ the statement is correct. Recall that we look at only the even rounds, otherwise we mention explicitly. Now, let HS be a two-dimensional hyper-square starting from node i with even parity (the odd case is analogous), then to become occupied it needs its even-part to become fully black. We want to show if HS has occupied neighbors in two distinct dimensions in some configuration \mathcal{C}_t , it will be occupied in $\mathcal{C}_{t+t'}$ for some even constant t' . It has 4 inner neighbors and $2d - 4$ outer neighbors. If two of the outer neighbors are occupied in \mathcal{C}_t , their odd-part must be black because their parity is different with HS . Thus, each node in the odd-part of HS has two black neighbors, which implies that the odd-part becomes fully black in \mathcal{C}_{t+1} and thus the even-part becomes black in \mathcal{C}_{t+2} , i.e., HS is occupied. For the case that HS has two occupied inner neighbors or one inner and one outer neighbor, see Figure 4.

Assume as the induction hypothesis that the claim is correct for $r - 1 \geq 2$, we prove it is true also for r . Suppose that the r -dimensional hyper-square HS starting from i has occupied neighbors in r distinct dimensions in some configuration \mathcal{C}_t . Furthermore, assume the parity of HS is even (the odd case is handled analogously). The lower face or upper face of HS must have r occupied neighbors as we discussed above; let it be the lower face.

We claim one of the neighbors provides for each node in the even-part (similarly each node in the odd-part) of the lower face in every odd round (resp. even round) a black neighbor. To show that let us distinguish two cases. If one of the neighbors of HS is outer, say the hyper-square HS' , then the lower face of HS' , which is a neighbor of the lower face of HS , satisfies our requirement. If there is no outer neighbor, then HS has at least one occupied neighbor in each of the first r dimensions. The neighbor in the r -th dimension which has its upper face adjacent to the lower face of HS must be occupied (we assumed the lower face has r occupied neighbors), which is then our required neighbor. Note that to occupy the lower face only making even nodes (the nodes in the even-part) black in even rounds or odd nodes (the nodes in the odd-part) black in odd rounds help because if for example the odd-part of the lower face is fully black in an even round, it does not have any impact on its occupation. By applying the induction hypothesis and the fact that one of the neighbors provides for each even node (similarly odd node) in each odd round (resp. even round) a black neighbor, we can conclude that the lower face must become occupied in a constant and even number of rounds. This is true because the remaining $r - 1$ neighbors must make the lower face occupied under two-way $(r - 1)$ -BP and the extra neighbor needed by r -BP is always provided. Now, we can apply the same argument on the upper face by setting the lower face as the neighbor which provides for each even node (similarly odd node) in the upper face in each even round (resp. odd round) a black neighbor.

Sliding Window Property Testing for Regular Languages

Moses Ganardi

Universität Siegen, Germany
ganardi@eti.uni-siegen.de

Danny Huc

Universität Siegen, Germany
huc@eti.uni-siegen.de

Markus Lohrey

Universität Siegen, Germany
lohrey@eti.uni-siegen.de

Tatiana Starikovskaya

DI/ENS, PSL Research University, Paris, France
tat.starikovskaya@gmail.com

Abstract

We study the problem of recognizing regular languages in a variant of the streaming model of computation, called the sliding window model. In this model, we are given a size of the sliding window n and a stream of symbols. At each time instant, we must decide whether the suffix of length n of the current stream (“the active window”) belongs to a given regular language.

Recent works [14, 15] showed that the space complexity of an optimal deterministic sliding window algorithm for this problem is either constant, logarithmic or linear in the window size n and provided natural language theoretic characterizations of the space complexity classes. Subsequently, [16] extended this result to randomized algorithms to show that any such algorithm admits either constant, double logarithmic, logarithmic or linear space complexity.

In this work, we make an important step forward and combine the sliding window model with the property testing setting, which results in ultra-efficient algorithms for all regular languages. Informally, a sliding window property tester must accept the active window if it belongs to the language and reject it if it is far from the language. We show that for every regular language, there is a deterministic sliding window property tester that uses logarithmic space and a randomized sliding window property tester with two-sided error that uses constant space.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Streaming algorithms, approximation algorithms, regular languages

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.6

Related Version A full version of this paper is available at <https://arxiv.org/abs/1909.10261>.

Funding *Moses Ganardi*: Funded by DFG project LO 748/13-1.

Markus Lohrey: Funded by DFG project LO 748/13-1.

1 Introduction

Regular expression search constitutes an important part of many search engines for biological data or code, such as, for example, Elasticsearch Service¹. In this paper, we consider the following formalization of this problem. We assume to be given an integer n , a regular

¹ <https://www.elastic.co>



language L , and a stream of symbols that we receive one symbol at a time. At each time instant, we have direct access only to the last arrived symbol, and must decide whether the suffix of length n of the current stream (“the active window”) belongs to L .

The model described above is a variant of the streaming model and was introduced by Datar et al. [10], where the authors proved that the number of 1’s in a 0/1-sliding window of size n can be maintained in space $\mathcal{O}(\frac{1}{\epsilon} \cdot \log^2 n)$ if one allows a multiplicative error of $1 \pm \epsilon$. The motivation for this model of computation is that in many streaming applications, data items are outdated after a certain time, and the sliding window setting is a simple way to model this. In general, we aim to avoid storing the window content explicitly, and, instead, to work in considerably smaller space, e.g. polylogarithmic space with respect to the window length. For more details on the sliding window model see [1, Chapter 8].

The study of recognizing regular languages in the sliding window model was commenced in [14, 15]. In [15], Ganardi et al. showed that for every regular language L the optimal space bound for a deterministic sliding window algorithm is either constant, logarithmic or linear in the window size n . In [14], Ganardi et al. gave characterizations for these space classes. More formally, they showed that a regular language has a deterministic sliding window algorithm with space $\mathcal{O}(\log n)$ (resp., $\mathcal{O}(1)$) if and only if it is a Boolean combination of so-called regular left-ideals and regular length languages (resp., suffix-testable languages and regular length languages). A subsequent work [16] studied the space complexity of randomized sliding window algorithms for regular languages. It was shown that for every regular language L the optimal space bound of randomized sliding window algorithm is $\mathcal{O}(1)$, $\mathcal{O}(\log \log n)$, $\mathcal{O}(\log n)$, or $\mathcal{O}(n)$. Moreover, complete characterizations of these space classes were provided.

1.1 Our results

Previous study implies that even simple languages require linear space in the sliding window model, which gives the motivation to seek for novel approaches in order to achieve efficient algorithms for all regular languages. We take our inspiration from the property testing model introduced by Goldreich et. al [22]. In this model, the task is to decide whether the input has a particular property P , or is “far” from any input satisfying it. For a function $\gamma : \mathbb{N} \rightarrow \mathbb{R}_{\geq 0}$, we say that a word w of length n is γ -far from satisfying P , if the Hamming distance between w and any word w' satisfying P is at least $\gamma(n)$. We will call the function $\gamma(n)$ the Hamming gap of the tester. We must make the decision by inspecting as few symbols of the input as possible, and the time complexity of the algorithm is defined to be equal to the number of inspected symbols. The motivation is that when working with large-scale data, accessing a data item is a very time-expensive operation. The membership problem for a regular language in the property testing model was studied by Alon et al. [2] who showed that for every regular language L and every constant $\epsilon > 0$, there is a property tester with Hamming gap $\gamma(n) = \epsilon n$ for deciding membership in L that can make the decision by inspecting a random constant-size sample of symbols of the input word.

In this work, we introduce a class of algorithms called *sliding window property testers*. Informally, at each time moment, a sliding window property tester must accept if the active window has the property P and reject if it is far from satisfying P . The space complexity of a sliding window property tester is defined to be all the space used, including the space we need to store information about the input. We consider deterministic sliding window property testers and randomized sliding window property testers with one-sided and two-sided errors (for a formal definition, see Section 2). A similar but simpler model of streaming property testers, where the whole stream is considered, was introduced by Feigenbaum et al. [11].

François et al. [12] continued the study of this model in the context of language membership problems and came up with a streaming property tester for visibly pushdown languages that uses polylogarithmic space. Note that deciding membership in a regular languages becomes trivial in this model (where the active window is the whole stream): one can simply simulate a deterministic finite automaton on the stream. What makes the sliding window model more difficult is the fact that the oldest symbol in the active window expires in the next step.

While at first sight the only connection between property testers and sliding window property testers is that we must accept the input if it satisfies P and reject if it is far from satisfying P , there is, in fact, a deeper link. In particular, the above mentioned result of Alon et al. [2] combined with an optimal sampling algorithm for sliding windows [4], immediately yields a $\mathcal{O}(\log n)$ -space, two-sided error sliding window property tester with Hamming gap $\gamma(n) = \epsilon n$ for every regular language. We will improve on this observation. Our main contribution are tight complexity bounds for each of the following classes of sliding window property testers for regular languages: deterministic sliding window property testers and randomized sliding window property testers with one-sided and two-sided error.

Deterministic sliding window property testers. We call a language L *trivial*, if for some constant $c > 0$ the following holds: For every word $w \in \Sigma^*$ such that L contains a word of length $|w|$, the Hamming distance from w to L is at most c . Every trivial regular language has a constant-space deterministic sliding window property tester with constant Hamming gap (Theorem 4). For generic regular languages, we show a deterministic sliding window property tester with constant Hamming gap that uses $\mathcal{O}(\log n)$ space. This is particularly surprising, because for Hamming gap zero (i.e., the exact case) [16] showed a space lower bound of $\Omega(n)$ for generic regular languages. In other words, a constant Hamming gap allows an exponential space improvement. We also show that for *non-trivial* regular languages, $\mathcal{O}(\log n)$ space is the best one can hope to achieve, even for Hamming gap $\gamma(n) = \epsilon n$ (Theorem 6).

Randomized sliding window property testers with two-sided error. Next, we show that for every regular language, there is a randomized sliding window property tester with Hamming gap $\gamma(n) = \epsilon n$ and two-sided error that uses constant space (Theorem 7). This is an optimal bound and a considerable improvement compared to the tester that can be obtained by combining the property tester of Alon et al. [2] and an optimal sampling algorithm for sliding windows [4]. Our constant space tester makes use of a probabilistic counter from [16].

Randomized sliding window property testers with one-sided error. While our randomized sliding window property tester with two-sided error is optimal, we believe that a two-sided error is a very strong relaxation that has to be avoided in some applications. To this end, we study the one-sided error randomized setting. The general landscape for this setting is the most complex: In Theorems 8 and 9, we show that for every regular language L , the space complexity of an optimal randomized sliding window property tester with one-sided error is either $\mathcal{O}(1)$, $\mathcal{O}(\log \log n)$, or $\mathcal{O}(\log n)$, and we provide language theoretic characterizations of these space classes.

In order to show our upper bound results, we demonstrate novel combinatorial properties of automata and regular languages and develop new streaming techniques, such as probabilistic counters, which can be of interest on their own. To show the lower bound results, we introduce a new methodology, which could potentially simplify further establishments of lower bounds in string processing tasks in the streaming setting: namely, we view the testers as nondeterministic automata, and study their behaviour.

1.2 Related work

The results above assume that the regular language admits a constant-space description and we will follow the same assumption in this work. Currently, there are few studies on the dependency of the complexity of sliding window algorithms on the size of the language description. On the negative side, Ganardi et al. [14] showed that there are regular languages such that any sliding window algorithm that achieves logarithmic space (in the window size) depends exponentially on the automata size. On the positive side, there is an extensive study of the pattern matching problem and its variants that gives sub-exponential upper bounds for a class of (very simple) regular languages. In this problem, we are given a pattern and a streaming text T , and at each moment we must decide if the active window is equal to the pattern. This problem and its generalisations have been studied in [5, 6, 7, 8, 9, 19, 20, 21, 28, 30].

Similar to regular languages, we can ask whether the current active window belongs to a given context-free language. This question was studied in [3, 24, 25, 26] for the model where the active window is the complete stream and in [13, 18] for the sliding-window model.

2 Sliding window property tester

We fix a finite alphabet Σ for the rest of the paper. We denote by Σ^* the set of all words over Σ and by Σ^n the set of words over Σ of length n . The empty word is denoted by λ . Let w be a word. We say that v is a *prefix (suffix)* of w if $w = xv$ ($w = vx$) for some word x . We say that v is a *factor* of w if $w = xvy$ for some words x, y . The *Hamming distance* between two words $u = a_1 \cdots a_n$ and $v = b_1 \cdots b_n$ of equal length is the number of positions where u and v differ, i.e. $\text{dist}(u, v) = |\{i : a_i \neq b_i\}|$. The distance of a word u to a language L is defined as $\text{dist}(u, L) = \inf\{\text{dist}(u, v) : v \in L\} \in \mathbb{N} \cup \{\infty\}$.

A *deterministic finite automaton* (DFA) is a tuple $A = (Q, \Sigma, q_0, \delta, F)$ where Q is a finite set of states, Σ is the input alphabet, q_0 is the initial state, $\delta : Q \times \Sigma \rightarrow Q$ is the transition mapping and $F \subseteq Q$ is the set of final states. We extend δ to a mapping $\delta : Q \times \Sigma^* \rightarrow Q$ inductively in the usual way: $\delta(q, \lambda) = q$ and $\delta(q, aw) = \delta(\delta(q, a), w)$. The language accepted by A is $L(A) = \{w \in \Sigma^* : \delta(q_0, w) \in F\}$. A language is *regular* if it is accepted by a DFA. For more background in automata theory see [23].

A *stream* is a word $a_1 a_2 \cdots a_m$ over Σ . A *sliding window algorithm* is a family $\mathcal{A} = (A_n)_{n \geq 0}$ of streaming algorithms. Given a window size $n \in \mathbb{N}$ and an input stream $a_1 a_2 \cdots a_m \in \Sigma^*$ the algorithm A_n reads the stream symbol by symbol from left to right and thereby updates its memory content. After reading a prefix $a_1 \cdots a_t$ ($0 \leq t \leq m$) the algorithm is required to compute an output value that depends on the *active window* $\text{last}_n(a_1 \cdots a_t) = a_{t-n+1} \cdots a_t$ at time t . For convenience, for $i < 0$ we define $a_i = \square$ where $\square \in \Sigma$ is an arbitrary fixed symbol. In other words, we assume an initial window \square^n that is active at time $t = 0$. We consider *deterministic sliding window algorithms* (where every A_n can be viewed as a DFA) and *randomized sliding window algorithms* (where every A_n can be viewed as a probabilistic finite automaton in the sense of Rabin [29]). In the latter case, A_n updates in each step its memory content according to a probability distribution that depends on the current memory content and the current input symbol. Let $\gamma : \mathbb{N} \rightarrow \mathbb{R}_{\geq 0}$ be a function such that $\gamma(n) \leq n$ for all $n \in \mathbb{N}$, let α, β be probabilities, and let $L \subseteq \Sigma^*$ be a language.

► **Definition 1.** A deterministic sliding window (property) tester for L with Hamming gap $\gamma(n)$ is a deterministic sliding window algorithm $\mathcal{A} = (A_n)_{n \geq 0}$ such that for every input stream $w \in \Sigma^*$ and every window size n the following properties hold:

- if $\text{last}_n(w) \in L$, then A_n accepts;
- if $\text{dist}(\text{last}_n(w), L) > \gamma(n)$, then A_n rejects.

► **Definition 2.** A randomized sliding window (property) tester for L with Hamming gap $\gamma(n)$ and error (α, β) is a randomized sliding window algorithm $\mathcal{A} = (A_n)_{n \geq 0}$ such that for every input stream $w \in \Sigma^*$ and every window size n the following properties hold:

- if $\text{last}_n(w) \in L$, then A_n accepts with probability at least $1 - \alpha$;
- if $\text{dist}(\text{last}_n(w), L) > \gamma(n)$, then A_n rejects with probability at least $1 - \beta$.

We say that \mathcal{A} has one-sided error if \mathcal{A} has error $(0, 1/2)$ and two-sided error if \mathcal{A} has error $(1/3, 1/3)$.

Notice that our definition is non-uniform since we allow an arbitrary algorithm A_n for each window size n . If the window size is not specified, then it is implicitly universally quantified. The space consumption of \mathcal{A} is the mapping $s(n)$, where $s(n)$ is the space consumption of A_n , i.e., the maximal number of bits stored by A_n while reading any input stream. We can assume that $s(n) \in \mathcal{O}(n)$ since A_n can store the active window in $\mathcal{O}(n)$ bits. The goal is to devise algorithms which only use $o(n)$ space. Using probability amplification (similar to [16]) one can replace the error probability $1/3$ in the two-sided error setting (resp. $1/2$ in the one-sided error setting) by any probability $p < 1/2$ (resp. $p < 1$). This influences the space complexity only by a constant factor. The case of Hamming gap $\gamma(n) = 0$ corresponds to exact membership testing to L which was studied in [14, 15, 16]. In this paper, we focus on the two cases $\gamma(n) = c$ for some constant $c > 0$ and $\gamma(n) = \epsilon n$ for some $\epsilon > 0$.

Before we come to the main results of the paper we state two simple facts about the sliding window testers.

► **Lemma 3.** Assume that $L = \bigcup_{i=1}^k L_i$ and that for every $1 \leq i \leq k$ there exists a randomized sliding window tester for L_i with Hamming gap $\gamma(n)$ and error (α, β) that uses space $s_i(n)$. Then there exists a sliding window tester for L with Hamming gap $\gamma(n)$ and error (α, β) that uses space $\mathcal{O}(\sum_{i=1}^k s_i(n))$.

The second fact concerns so-called trivial languages. Let $\gamma : \mathbb{N} \rightarrow \mathbb{R}_{\geq 0}$ be a mapping with $\gamma(n) \leq n$ for all $n \geq 0$. A language is $L \subseteq \Sigma^*$ is γ -trivial if there exists $n_0 \in \mathbb{N}$ such that for all $n \geq n_0$ with $L \cap \Sigma^n \neq \emptyset$ and all $w \in \Sigma^n$ we have $\text{dist}(w, L) \leq \gamma(n)$. If $\gamma(n) \in \mathcal{O}(1)$, we say that L is trivial. Note that Alon et al. [2] call a language L trivial if L is (ϵn) -trivial for all $\epsilon > 0$ according to our definition. In the long version [17] we show that both definitions coincide for regular languages, but we will not make use of this fact.

► **Theorem 4.** For every trivial (but not necessarily regular) language there is a deterministic sliding window tester with constant Hamming gap that uses constant space. The converse is also true: If for a language L there is a deterministic constant-space sliding window tester with Hamming gap $\gamma(n)$, then there exists a constant c such that L is $(\gamma + c)$ -trivial.

3 Main results

Our first main contribution is a deterministic logspace sliding window tester for every regular language, together with a matching lower bound for so-called *nontrivial* regular languages (defined above).

► **Theorem 5.** For every regular language L , there exists a deterministic sliding window tester for L with constant Hamming gap which uses $\mathcal{O}(\log n)$ space.

► **Theorem 6.** For every non-trivial regular language L , there exist $\epsilon > 0$ and infinitely many window sizes $n \in \mathbb{N}$ on which every deterministic sliding window tester for L with Hamming gap ϵn uses space $\Omega(\log n)$.

Our second main contribution is a constant-space randomized sliding window property tester with two-sided error for any regular language:

► **Theorem 7.** *For every regular language L and every $\epsilon > 0$, there exists a randomized sliding window tester for L with two-sided error and Hamming gap $\gamma(n) = \epsilon n$ that uses space $\mathcal{O}(1/\epsilon)$.*

While the randomized setting with two-sided error allows ultra-efficient testers, we find that allowing a two-sided error is a very strong relaxation. To this end, we study the randomized setting with one-sided error. In this setting, only a small class of regular languages admits sliding window testers working in space $o(\log n)$. A language $L \subseteq \Sigma^*$ is *suffix-free* if $xy \in L$ and $x \neq \lambda$ imply $y \notin L$.

► **Theorem 8.** *If L is a finite union of trivial regular languages and suffix-free regular languages, then there exists a randomized sliding window tester for L with one-sided error and constant Hamming gap which uses $\mathcal{O}(\log \log n)$ space.*

► **Theorem 9.** *Let L be a regular language.*

- *If L is not a finite union of trivial regular languages and suffix-free regular languages, there exist $\epsilon > 0$ and infinitely many window sizes n on which every randomized sliding window tester for L with one-sided error and Hamming gap ϵn uses space $\Omega(\log n)$.*
- *If L is non-trivial, then there exist $\epsilon > 0$ and infinitely many window sizes n on which every sliding window tester for L with one-sided error and Hamming gap ϵn uses space $\Omega(\log \log n)$.*

We sketch the proofs of Theorem 5, 7, and 8 in Sections 4.1, 4.2, and 4.3, respectively. The proofs of the lower bounds (Theorems 6 and 9) can be found in the long version [17]. We would like to emphasize that the lower bounds shown in [17] are stronger than those stated in Theorems 6 and 9. More precisely, we show space lower bounds for nondeterministic and co-nondeterministic sliding window testers; see [17] for definitions.

4 Proofs of the upper bounds

In this section we sketch proofs of Theorems 5, 7, and 8 that give upper bounds for deterministic and (one-sided and two-sided error) randomized sliding window testers. All algorithms in this section satisfy the stronger property that words with large prefix distance are rejected by the algorithm with high probability (probability one in the deterministic setting). The *prefix distance* between words $u = a_1 \cdots a_n$ and $v = b_1 \cdots b_n$ is $\text{pdist}(u, v) = \min\{i \in \{0, \dots, n\} : a_{i+1} \cdots a_n = b_{i+1} \cdots b_n\}$. Clearly, we have $\text{dist}(u, v) \leq \text{pdist}(u, v)$. We extend the definition to languages: for a language L , let $\text{pdist}(u, L) = \min\{\text{pdist}(u, v) : v \in L\}$. The prefix distance between two runs $\pi = (q_0, a_1, \dots, q_{n-1}, a_n, q_n)$ and $\rho = (p_0, b_1, \dots, p_{n-1}, b_n, p_n)$ is defined as $\text{pdist}(\pi, \rho) = \min\{i \in \{0, \dots, n\} : (q_i, a_{i+1}, \dots, q_{n-1}, a_n, q_n) = (p_i, b_{i+1}, \dots, p_{n-1}, b_n, p_n)\}$.

For our upper bound proofs it is convenient to work with DFAs which read the input word from right to left. A *right-deterministic finite automaton (rDFA)* is a tuple $B = (Q, \Sigma, F, \delta, q_0)$, where Q , Σ , q_0 and F are as in a DFA, and $\delta: \Sigma \times Q \rightarrow Q$ is the transition function. We extend δ to a mapping $\delta: Q \times \Sigma^* \rightarrow Q$ analogously to DFAs: $\delta(q, \lambda) = q$ and $\delta(q, wa) = \delta(\delta(q, a), w)$. The regular language recognized by the rDFA B is $L(B) = \{w \in \Sigma^* : \delta(w, q_0) \in F\}$. A run from $p_0 \in Q$ to $p_n \in Q$ on a word $x = a_n \cdots a_2 a_1 \in \Sigma^*$ is a sequence $\pi = (p_n, a_n, p_{n-1}, \dots, p_2, a_2, p_1, a_1, p_0)$ such that $p_i = \delta(a_i, p_{i-1})$ for all $1 \leq i \leq n$. The *length* of π is $|\pi| = n$. We visualize π in the form

$$\pi: p_n \xleftarrow{a_n} p_{n-1} \xleftarrow{a_{n-1}} \cdots \xleftarrow{a_2} p_1 \xleftarrow{a_1} p_0.$$

If $p_n \in F$, then π is an *accepting run*. A run of length 1 is a *transition*. If π is a run from p to q on a word v , and ρ is a run from q to r on a word u , then $\rho\pi$ denotes the unique run from p to r on uv . We denote by $\pi_{w,q}$ the unique run on w from q .

Strongly connected graphs. With a DFA $A = (Q, \Sigma, q_0, \delta, F)$ we associate the directed graph (Q, E) with edge set $E = \{(p, \delta(p, a)) \mid p \in Q, a \in \Sigma\}$. Similarly, with an rDFA $A = (Q, \Sigma, F, \delta, q_0)$ we associate the directed graph (Q, E) with edge set $E = \{(p, \delta(a, p)) \mid p \in Q, a \in \Sigma\}$. Let A be a DFA or an rDFA. Two states p, q in A are *strongly connected* if there exists a path in (Q, E) from p to q , and vice versa. The *strongly connected components* (SCCs) of A with state set Q are the maximal subsets $C \subseteq Q$ in which all states $p, q \in C$ are strongly connected. A state $q \in Q$ is *transient* if there exists no nonempty path from q to q . An SCC C is *transient* if it only contains a single transient state. There is a natural partial order on the SCCs, called the *SCC-ordering*, where the SCC C_1 is smaller than the SCC C_2 if there exists a path in (Q, E) from a state in C_1 to a state in C_2 .

The following combinatorial result from [2] will be used in this paper. Consider a directed graph $G = (V, E)$. The period of G is the greatest common divisor of all cycle lengths in G . If G is acyclic we define the period to be ∞ .

► **Lemma 10** (cf. [2]). *Let $G = (V, E)$ be a strongly connected directed graph with $E \neq \emptyset$ and finite period g . Then there exist a partition $V = \bigcup_{i=0}^{g-1} V_i$ and a constant $m(G) \leq 3|V|^2$ with the following properties:*

- *For every $0 \leq i, j \leq g-1$ and for every $u \in V_i, v \in V_j$ the length of every directed path from u to v in G is congruent to $j-i$ modulo g .*
- *For every $0 \leq i, j \leq g-1$, for every $u \in V_i, v \in V_j$ and every integer $r \geq m(G)$, if r is congruent to $j-i$ modulo g , then there exists a directed path from u to v in G of length r .*

If $G = (V, E)$ is strongly connected with $E \neq \emptyset$ and finite period g , and V_0, \dots, V_{g-1} satisfy the properties from Lemma 10, then we define the *shift* from $u \in V_i$ to $v \in V_j$ by

$$\text{shift}(u, v) = j - i \pmod{g} \in \{0, \dots, g-1\}. \quad (1)$$

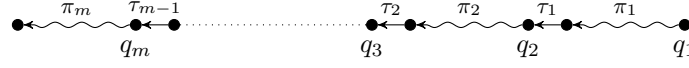
Notice that this definition is independent of the partition $\bigcup_{i=0}^{g-1} V_i$ since any path from u to v has length $\ell \equiv \text{shift}(u, v) \pmod{g}$ by Lemma 10. Also note that $\text{shift}(u, v) + \text{shift}(v, u) \equiv 0 \pmod{g}$. In the following let $g(C)$ denote the period of the SCC C .

► **Lemma 11.** *For every regular language L there exists an rDFA A for L and a number g such that every non-transient SCC C in A has period $g(C) = g$.*

Path summaries. We start by recalling the notion of a path summary from [14], where it was used in order to prove a logspace upper bound for regular left-ideals (in the exact setting where the Hamming gap is zero). For the rest of Section 4 we fix a regular language $L \subseteq \Sigma^*$ and an rDFA $B = (Q, \Sigma, F, \delta, q_0)$ which recognizes L . By Lemma 11, we can assume that every non-transient SCC C of B has period $g(C) = g$. Consider a run $\pi = (p_n, a_n, \dots, a_1, p_0)$ on $x = a_n \cdots a_1$. If all states p_n, \dots, p_0 are contained in a single SCC we call π *internal*. We can decompose $\pi = \pi_m \tau_{m-1} \pi_{m-1} \cdots \tau_1 \pi_1$, where each π_i is a possibly empty internal run and each τ_i is a single transition connecting two distinct SCCs. We call this unique factorization the *SCC-factorization* of π , which is illustrated in Figure 1. The *path summary* of π is

$$\text{ps}(\pi) = (|\pi_m|, q_m)(|\tau_{m-1}\pi_{m-1}|, q_{m-1}) \cdots (|\tau_2\pi_2|, q_2)(|\tau_1\pi_1|, q_1),$$

where q_i is the first state in π_i ($1 \leq i \leq m$). Note that m is bounded by the constant number of states of B . Hence, a path summary can be stored with $\mathcal{O}(\log |\pi|)$ bits.



■ **Figure 1** The SCC-factorization of a run.

Periodic acceptance sets. For $a \in \mathbb{N}$ and $X \subseteq \mathbb{N}$ we use the standard notation $X + a = \{a + x : x \in X\}$. For a state $q \in Q$ we define $\text{Acc}(q) = \{n \in \mathbb{N} : \exists w \in \Sigma^n : \delta(w, q) \in F\}$. A set $X \subseteq \mathbb{N}$ is *eventually d -periodic*, where $d \geq 1$ is an integer, if there exists a *threshold* $t \in \mathbb{N}$ such that for all $x \geq t$ we have $x \in X$ if and only if $x + d \in X$. If X is eventually d -periodic for some $d \geq 1$, then X is *eventually periodic*.

► **Lemma 12.** *For every $q \in Q$ the set $\text{Acc}(q)$ is eventually g -periodic.*

Two sets $X, Y \subseteq \mathbb{N}$ are *equal up to a threshold* $t \in \mathbb{N}$, in symbol $X =_t Y$, if for all $x \geq t$: $x \in X$ iff $x \in Y$. Sets $X, Y \subseteq \mathbb{N}$ are *almost equal* if $X =_t Y$ for some threshold $t \in \mathbb{N}$.

► **Lemma 13.** *Let C be a non-transient SCC in B , $p, q \in C$ and $s = \text{shift}(p, q)$. Then $\text{Acc}(p)$ and $\text{Acc}(q) + s$ are almost equal.*

► **Corollary 14.** *There exists a threshold $t \in \mathbb{N}$ such that*

1. $\text{Acc}(q) =_t \text{Acc}(q) + g$ for all $q \in Q$, and
2. $\text{Acc}(p) =_t \text{Acc}(q) + \text{shift}(p, q)$ for all non-transient SCCs C and all $p, q \in C$.

We fix the threshold t from Corollary 14 for the rest of Section 4. The following lemma is the main tool to prove the correctness of our sliding window testers. It states that if a word of length n is accepted from p and ρ is any internal run from p of length at most n , then, up to a bounded length prefix, ρ can be extended to an accepting run of length n . Formally, a run π k -simulates a run ρ if one can factorize $\rho = \rho_1 \rho_2$ and $\pi = \pi' \rho_2$ where $|\rho_1| \leq k$.

► **Lemma 15.** *If ρ is an internal run starting from p of length at most n and $n \in \text{Acc}(p)$, then there exists an accepting run π from p of length n which t -simulates ρ .*

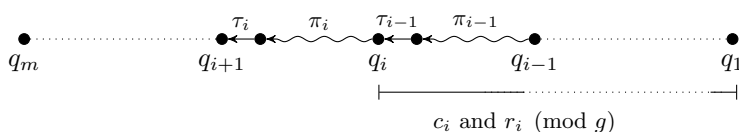
4.1 Deterministic logspace tester

Proof of Theorem 5. Let $n \in \mathbb{N}$ such that $n \geq |Q|$ (for $n < |Q|$ we use a trivial streaming algorithm which stores the window explicitly). The algorithm maintains the set $\{\text{ps}(\pi_w, q) \mid q \in Q\}$ where $w \in \Sigma^n$ is the active window. Initially this set is $\{\text{ps}(\pi_w, q) \mid q \in Q\}$ for $w = \square^n$. Now suppose $w = av$ for some $a \in \Sigma$ and the next symbol of the stream is $b \in \Sigma$, i.e. the new active window is vb . For each transition $q \xleftarrow{b} p$ in B we can compute $\text{ps}(\pi_{vb, p})$ from $\text{ps}(\pi_{av, q})$ as follows. Suppose that $\text{ps}(\pi_{av, q}) = (\ell_m, q_m) \cdots (\ell_1, q_1)$ where $q = q_1$.

- If p and q belong to the same SCC, then we increment ℓ_1 by one, else we append a new pair $(1, p)$.
- If $\ell_m > 0$ we decrement ℓ_m by one. If $\ell_m = 0$ we remove the pair (ℓ_m, q_m) and we decrement ℓ_{m-1} by one (in this case we must have $m > 1$ and $\ell_{m-1} > 0$).

The obtained path summary is $\text{ps}(\pi_{vb, p})$. This data structure can be stored with $\mathcal{O}(\log n)$ bits since it contains $|Q|$ path summaries, each of which can be stored in $\mathcal{O}(\log n)$ bits.

It remains to define a proper acceptance condition. Consider the run $\pi = \pi_w, q_0$, its SCC-factorization $\pi_m \tau_{m-1} \pi_{m-1} \cdots \tau_1 \pi_1$ and its path summary $(\ell_m, q_m) \cdots (\ell_1, q_1)$. The algorithm accepts if and only if $\ell_m = |\pi_m| \in \text{Acc}(q_m)$. If $w \in L$, then clearly $|\pi_m| \in \text{Acc}(q_m)$. If $|\pi_m| \in \text{Acc}(q_m)$, then the internal run π_m can be t -simulated by an accepting run π'_m of equal length by Lemma 15. The run $\pi'_m \tau_{m-1} \pi_{m-1} \cdots \tau_1 \pi_1$ is accepting and witnesses that $\text{pdist}(w, L) \leq t$. ◀



■ **Figure 2** A compact summary of a run π .

4.2 Randomized constant-space tester with two-sided error

Let us first define a probabilistic counter, similar to the approximate counter by Morris [27], which uses $O(\log \log n)$ bits. For our purposes it suffices to distinguish high and low counter states. Consider a probabilistic data structure Z representing a counter. Its operations are incrementing the counter (using random coins) and querying whether the state of the counter is *low* or *high*. Initially Z is in a low state. The random state reached after k increments is denoted by $Z(k)$. Given numbers $0 \leq \ell < h$ (they will depend on our window size n) we say that Z is an (h, ℓ) -counter with error probability $\delta < \frac{1}{2}$ if for all $k \in \mathbb{N}$ we have:

- If $k \leq \ell$, then $\text{Prob}[Z(k) \text{ is high}] \leq \delta$.
- If $k \geq h$, then $\text{Prob}[Z(k) \text{ is low}] \leq \delta$.

► **Lemma 16.** *For all $h, \ell, \xi > 0$ with $\ell \leq (1 - \epsilon)h + \mathcal{O}(1)$ there exists an (h, ℓ) -counter Z with error probability $1/3|Q|$ which internally stores $\mathcal{O}(\log(1/\epsilon))$ bits.*

Fix a parameter $0 < \epsilon < 1$ and a window length $n \in \mathbb{N}$. Based on the previous concepts, we are now able to describe a randomized sliding window tester for a regular language L with Hamming gap ϵn that uses $\mathcal{O}(\log(1/\epsilon))$ bits. Let Z be the (h, ℓ) -counter with error probability $1/(3|Q|)$ from Lemma 16 where $h = n - t$ and $\ell = (1 - \epsilon)n + t + 1$. The counter is used to define so-called compact summaries of runs.

► **Definition 17.** *A compact summary $cs = (q_m, r_m, c_m) \cdots (q_2, r_2, c_2)(q_1, r_1, c_1)$ is a sequence of triples, where each triple (q_i, r_i, c_i) consists of a state $q_i \in Q$, a remainder $0 \leq r_i \leq g - 1$, and a state c_i of the (h, ℓ) -counter Z . The state c_1 must be low and $r_1 = 0$.*

A compact summary $(q_m, r_m, c_m) \cdots (q_1, r_1, c_1)$ represents a run π if the SCC-factorization of π has the form $\pi_m \tau_{m-1} \pi_{m-1} \cdots \tau_1 \pi_1$, and the following properties hold:

1. for all $1 \leq i \leq m$, π_i starts in q_i ;
2. for all $2 \leq i \leq m$, if $|\tau_{i-1} \pi_{i-1} \cdots \tau_1 \pi_1| \leq (1 - \epsilon)n + t + 1$, then c_i is the low state; and if $|\tau_{i-1} \pi_{i-1} \cdots \tau_1 \pi_1| \geq n - t$, then c_i is the high state;
3. for all $2 \leq i \leq m$, $r_i = |\tau_{i-1} \pi_{i-1} \cdots \tau_1 \pi_1| \pmod{g}$.

The idea of a compact summary is visualized in Figure 2. If $m > |Q|$ then the above compact summary cannot represent a run. Therefore, we can assume that $m \leq |Q|$. For every triple (q_i, r_i, c_i) , the entries q_i and r_i only depend on the rDFA B , and hence can be stored with $\mathcal{O}(1)$ bits. Every state c_i of the probabilistic counter needs $\mathcal{O}(\log(1/\epsilon))$ bits. Hence, a compact summary can be stored in $\mathcal{O}(\log(1/\epsilon))$ bits. In contrast to Theorem 5, we maintain a set of compact summaries which represent all runs of B on the *complete* stream read so far (not only on the active window) with high probability.

► **Lemma 18.** *For a given input stream $w \in \Sigma^*$, we can maintain a set of compact summaries S containing for each $q \in Q$ a compact summary $cs_q \in S$ starting in q such that cs_q represents the unique run $\pi_{w,q}$ with probability at least $2/3$.*

It remains to define an acceptance condition on compact summaries. For every $q \in Q$ we define $\text{Acc}_{\text{mod}}(q) = \{\ell \pmod{g} : \ell \in \text{Acc}(q) \text{ and } \ell \geq t\}$, which is intuitively speaking the set of accepting remainders. Let $\text{cs} = (q_m, r_m, c_m) \cdots (q_1, r_1, c_1)$ be a compact summary. Since c_1 is the low initial state of the probabilistic counter, there exists a maximal index $i \in \{1, \dots, m\}$ such that c_i is low. We say that cs is *accepting* if $n - r_i \pmod{g} \in \text{Acc}_{\text{mod}}(q_i)$.

► **Proposition 19.** *Assume that $\epsilon n \geq t$. Let $w \in \Sigma^*$ with $|w| \geq n$ and let cs be a compact summary which represents π_{w, q_0} .*

1. *If $\text{last}_n(w) \in L$, then cs is accepting.*
2. *If cs is accepting, then $\text{pdist}(\text{last}_n(w), L) \leq \epsilon n$.*

Proof of Theorem 7. Assume that $\epsilon n \geq t$, otherwise we use a trivial streaming algorithm that stores the window explicitly with $\mathcal{O}(1/\epsilon)$ bits. We use the algorithm from Proposition 18 for each incoming symbol from the stream. To initialize, we run the algorithm on \square^n . The algorithm accepts if the computed compact summary starting in q_0 is accepting. From Proposition 18 and 19 we get:

- If $\text{pdist}(\text{last}_n(w), L) > \epsilon n$, then the algorithm rejects with probability at least $2/3$.
- If $\text{last}_n(w) \in L$, then the algorithm accepts with probability at least $2/3$.

This concludes the proof of the theorem. ◀

Comparing Theorems 5 and 7 leads to the question whether one can replace the Hamming gap $\gamma(n) = \epsilon n$ in Theorem 7 by $\gamma(n) = o(n)$ while retaining constant space at the same time. We show that this is not the case:

► **Lemma 20.** *Every randomized sliding window tester with two-sided error for $a^* \subseteq \{a, b\}^*$ with Hamming gap $\gamma(n)$ needs space $\Omega(\log n - \log \gamma(n))$ for infinitely many n .*

4.3 Randomized loglogspace tester with one-sided error

Let L be a finite union of trivial regular languages and suffix-free regular languages. In this section, we present a randomized sliding window tester for L with one-sided error and Hamming gap $\gamma(n) = \epsilon n$ that uses space $\mathcal{O}(\log \log n)$. By Lemma 3 and Theorem 4, it suffices to consider the case when L is a suffix-free regular language. As in Section 4 we fix an rDFA $B = (Q, \Sigma, F, \delta, q_0)$ for L such that $g(C) = g$ for all SCCs of A . Since L is suffix-free, B has the property that no final state can be reached from a final state by a non-empty run. We decompose B into a finite union of *partial automata*, similar to [14].

► **Definition 21.** *A sequence $(q_k, a_k, p_{k-1}), C_{k-1}, \dots, (q_2, a_2, p_1), C_1, (q_1, a_1, p_0), C_0, q_0$ is a path description if C_{k-1}, \dots, C_0 is a chain (read from right to left) in the SCC-ordering of B , $p_i, q_i \in C_i$, $q_{i+1} \xleftarrow{a_{i+1}} p_i$ is a transition in B for all $0 \leq i \leq k-1$, and $q_k \in F$.*

Each path description defines a *partial rDFA* $B_P = (Q_P, \Sigma, \{q_k\}, \delta_P, q_0)$ by restricting B to the state set $Q_P = \bigcup_{i=0}^{k-1} C_i \cup \{q_k\}$, restricting the transitions of B to internal transitions from the SCCs C_i and the transitions $q_{i+1} \xleftarrow{a_{i+1}} p_i$, and declaring q_k to be the only final state. The rDFA is partial since for every state p_i and every symbol $a \in \Sigma$ there exists at most one transition $q \xleftarrow{a} p_i$. Since the number of path descriptions P is finite and $L(B) = \bigcup_P L(B_P)$, it suffices to provide a sliding window tester for $L(B_P)$ (we again use Lemma 3 here).

From now on, we fix a path description P from Definition 21 and the partial automaton $B_P = (Q_P, \Sigma, \{q_k\}, \delta_P, q_0)$ corresponding to it. The acceptance sets $\text{Acc}(q)$ are defined with respect to B_P . If all C_i are transient, then $L(B_P)$ is a singleton and we can use a trivial sliding window tester with space complexity $\mathcal{O}(1)$. Now assume the contrary and let $0 \leq e \leq k-1$ be maximal such that C_e is nontransient.

► **Lemma 22.** *There exist $r_0, \dots, r_{k-1}, s_0, \dots, s_e \in \mathbb{N}$ such that the following holds:*

1. *For all $e + 1 \leq i \leq k$, the set $\text{Acc}(q_i)$ is a singleton.*
2. *Every run from q_i to q_{i+1} has length $r_i \pmod{g}$.*
3. *For all $0 \leq i \leq e$, $\text{Acc}(q_i) =_{s_i} \sum_{j=i}^{k-1} r_j + g\mathbb{N}$.*

Let $s = \max\{k, \sum_{j=0}^{k-1} r_j, s_0, \dots, s_e\}$ and for a word $w \in \Sigma^*$ define the function $\ell_w: Q \rightarrow \mathbb{N} \cup \{\infty\}$ where $\ell_w(q) = \inf\{\ell \in \mathbb{N} \mid \delta_P(\text{last}_\ell(w), q) = q_k\}$ (we set $\inf \emptyset = \infty$).

Let p be a random prime with $\Theta(\log \log n)$ bits. We now define an acceptance condition on $\ell_w(q)$. If $n \notin \text{Acc}(q_0)$, we always reject. Otherwise, we accept w iff $\ell_w(q_0) \equiv n$ modulo our randomly chosen prime p .

► **Lemma 23.** *Let $n \in \text{Acc}(q_0)$ be a window size with $n \geq s + |Q_P|$ and $w \in \Sigma^*$ with $|w| \geq n$. There exists a constant $c > 0$ such that:*

1. *if $\text{last}_n(w) \in L(B_P)$, then w is accepted with probability 1;*
2. *if $\text{pdist}(\text{last}_n(w), L(B_P)) > c$, then w is rejected with probability at least $2/3$.*

Proof of Theorem 8. Let $n \in \mathbb{N}$ be the window size. From the discussion above, it suffices to show a tester for a fixed partial automaton B_P . Assume $n \geq s + |Q|$, otherwise a trivial tester can be used. If $n \notin \text{Acc}(q_0)$, the tester always rejects. Otherwise, the tester picks a random prime p with $\Theta(\log \log n)$ bits and maintains $\ell_w(q) \pmod{p}$ for all $q \in Q_P$, where w is the stream read so far, which requires $\mathcal{O}(\log \log n)$ bits. When a symbol $a \in \Sigma$ is read, we can update ℓ_{wa} using ℓ_w : If $q = q_k$, then $\ell_{wa}(q) = 0$, otherwise $\ell_{wa}(q) = 1 + \ell_w(\delta_P(a, q)) \pmod{p}$ where $1 + \infty = \infty$. The tester accepts if $\ell_w(q_0) \equiv n \pmod{p}$. Lemma 23 guarantees correctness of the tester in the one-sided error setting. ◀

5 Further research

We gave a complete characterization of the space complexity of sliding window testers for regular languages. A natural open research problem is, whether similar results can be shown for context-free languages:

- Does every context-free language L have a deterministic sliding window tester with Hamming gap ϵn (or even $\mathcal{O}(1)$) that uses space $\mathcal{O}(\log n)$ (or at least space $o(n)$)?
- Does every context-free language L have a randomized sliding window tester with Hamming gap ϵn (or even $\mathcal{O}(1)$) that uses space $\mathcal{O}(1)$ (or at least space $o(n)$)?

If the answers to these questions turn out to be negative, then one might look at deterministic context-free languages or visibly pushdown languages.

References

- 1 Charu C. Aggarwal. *Data Streams – Models and Algorithms*. Springer, 2007.
- 2 Noga Alon, Michael Krivelevich, Ilan Newman, and Mario Szegedy. Regular Languages are Testable with a Constant Number of Queries. *SIAM Journal on Computing*, 30(6):1842–1862, 2000. doi:10.1137/S0097539700366528.
- 3 Ajesh Babu, Nutan Limaye, Jaikumar Radhakrishnan, and Girish Varma. Streaming algorithms for language recognition problems. *Theoretical Computer Science*, 494:13–23, 2013.
- 4 Vladimir Braverman, Rafail Ostrovsky, and Carlo Zaniolo. Optimal sampling from sliding windows. *Journal of Computer and System Sciences*, 78(1):260–272, 2012.
- 5 Dany Breslauer and Zvi Galil. Real-Time Streaming String-Matching. *ACM Transactions on Algorithms*, 10(4):22:1–22:12, 2014. doi:10.1145/2635814.
- 6 Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana Starikovskaya. Dictionary Matching in a Stream. In *Proceedings of ESA 2015*, volume 9294 of *Lecture Notes in Computer Science*, pages 361–372. Springer, 2015. doi:10.1007/978-3-662-48350-3_31.

- 7 Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana Starikovskaya. The k -mismatch problem revisited. In *Proceedings of SODA 2016*, pages 2039–2052. SIAM, 2016. doi:10.1137/1.9781611974331.ch142.
- 8 Raphaël Clifford, Tomasz Kociumaka, and Ely Porat. The streaming k -mismatch problem. In *Proceedings of SODA 2019*, pages 1106–1125. SIAM, 2019. doi:10.1137/1.9781611975482.68.
- 9 Raphaël Clifford and Tatiana Starikovskaya. Approximate Hamming Distance in a Stream. In *Proceedings of ICALP 2016*, volume 55 of *LIPICs*, pages 20:1–20:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.ICALP.2016.20.
- 10 Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining Stream Statistics over Sliding Windows. *SIAM Journal on Computing*, 31(6):1794–1813, 2002.
- 11 Joan Feigenbaum, Sampath Kannan, Martin Strauss, and Mahesh Viswanathan. Testing and Spot-Checking of Data Streams. *Algorithmica*, 34(1):67–80, 2002. doi:10.1007/s00453-002-0959-4.
- 12 Nathanaël François, Frédéric Magniez, Michel de Rougemont, and Olivier Serre. Streaming Property Testing of Visibly Pushdown Languages. In *Proceedings of ESA 2016*, volume 57 of *LIPICs*, pages 43:1–43:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.
- 13 Moses Ganardi. Visibly Pushdown Languages over Sliding Windows. In *Proceedings of STACS 2019*, volume 126 of *LIPICs*, pages 29:1–29:17. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.STACS.2019.29.
- 14 Moses Ganardi, Danny Hucce, Daniel König, Markus Lohrey, and Konstantinos Mamouras. Automata theory on sliding windows. In *Proceedings of STACS 2018*, volume 96 of *LIPICs*, pages 31:1–31:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.
- 15 Moses Ganardi, Danny Hucce, and Markus Lohrey. Querying Regular Languages over Sliding Windows. In *Proceedings of FSTTCS 2016*, volume 65 of *LIPICs*, pages 18:1–18:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.
- 16 Moses Ganardi, Danny Hucce, and Markus Lohrey. Randomized Sliding Window Algorithms for Regular Languages. In *Proceedings of ICALP 2018*, volume 107 of *LIPICs*, pages 127:1–127:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.
- 17 Moses Ganardi, Danny Hucce, Markus Lohrey, and Tatiana Starikovskaya. Sliding window property testing for regular languages. Technical report, arXiv.org, 2020. arXiv:1909.10261.
- 18 Moses Ganardi, Artur Jež, and Markus Lohrey. Sliding Windows over Context-Free Languages. In *Proceedings of MFCS 2018*, volume 117 of *LIPICs*, pages 15:1–15:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.
- 19 Shay Golan, Tsvi Kopelowitz, and Ely Porat. Streaming Pattern Matching with d Wildcards. In *Proceedings of ESA 2016*, volume 57 of *LIPICs*, pages 44:1–44:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.ESA.2016.44.
- 20 Shay Golan, Tsvi Kopelowitz, and Ely Porat. Towards Optimal Approximate Streaming Pattern Matching by Matching Multiple Patterns in Multiple Streams. In *Proceedings of ICALP 2018*, volume 107 of *LIPICs*, pages 65:1–65:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.ICALP.2018.65.
- 21 Shay Golan and Ely Porat. Real-Time Streaming Multi-Pattern Search for Constant Alphabet. In *Proceedings of ESA 2017*, volume 87 of *LIPICs*, pages 41:1–41:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.ESA.2017.41.
- 22 Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property Testing and its Connection to Learning and Approximation. *Journal of the ACM*, 45(4):653–750, 1998. doi:10.1145/285055.285060.
- 23 John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, Reading, MA, 1979.
- 24 Rahul Jain and Ashwin Nayak. The Space Complexity of Recognizing Well-Parentthesized Expressions in the Streaming Model: The Index Function Revisited. *IEEE Transactions on Information Theory*, 60(10):6646–6668, October 2014. doi:10.1109/TIT.2014.2339859.

- 25 Andreas Krebs, Nutan Limaye, and Srikanth Srinivasan. Streaming Algorithms for Recognizing Nearly Well-Parentesized Expressions. In *Proceedings of MFCS 2011*, volume 6907 of *Lecture Notes in Computer Science*, pages 412–423. Springer, 2011.
- 26 Frédéric Magniez, Claire Mathieu, and Ashwin Nayak. Recognizing Well-Parentesized Expressions in the Streaming Model. *SIAM Journal on Computing*, 43(6):1880–1905, 2014.
- 27 Robert H. Morris. Counting Large Numbers of Events in Small Registers. *Communications of the ACM*, 21(10):840–842, 1978. doi:10.1145/359619.359627.
- 28 Benny Porat and Ely Porat. Exact and Approximate Pattern Matching in the Streaming Model. In *Proceedings of FOCS 2009*, pages 315–323. IEEE Computer Society, 2009. doi:10.1109/FOCS.2009.11.
- 29 Michael O. Rabin. Probabilistic Automata. *Information and Control*, 6(3):230–245, 1963.
- 30 Tatiana Starikovskaya. Communication and Streaming Complexity of Approximate Pattern Matching. In *Proceedings of CPM 2017*, volume 78 of *LIPICs*, pages 13:1–13:11. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.CPM.2017.13.

On the Hardness of Set Disjointness and Set Intersection with Bounded Universe

Isaac Goldstein

Bar-Ilan University, Ramat Gan, Israel
goldshi@cs.biu.ac.il

Moshe Lewenstein

Bar-Ilan University, Ramat Gan, Israel
moshe@cs.biu.ac.il

Ely Porat

Bar-Ilan University, Ramat Gan, Israel
porately@cs.biu.ac.il

Abstract

In the SetDisjointness problem, a collection of m sets S_1, S_2, \dots, S_m from some universe U is preprocessed in order to answer queries on the emptiness of the intersection of some two query sets from the collection. In the SetIntersection variant, all the elements in the intersection of the query sets are required to be reported. These are two fundamental problems that were considered in several papers from both the upper bound and lower bound perspective.

Several conditional lower bounds for these problems were proven for the tradeoff between preprocessing and query time or the tradeoff between space and query time. Moreover, there are several unconditional hardness results for these problems in some specific computational models. The fundamental nature of the SetDisjointness and SetIntersection problems makes them useful for proving the conditional hardness of other problems from various areas. However, the universe of the elements in the sets may be very large, which may cause the reduction to some other problems to be inefficient and therefore it is not useful for proving their conditional hardness.

In this paper, we prove the conditional hardness of SetDisjointness and SetIntersection with bounded universe. This conditional hardness is shown for both the interplay between preprocessing and query time and the interplay between space and query time. Moreover, we present several applications of these new conditional lower bounds. These applications demonstrates the strength of our new conditional lower bounds as they exploit the limited universe size. We believe that this new framework of conditional lower bounds with bounded universe can be useful for further significant applications.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases set disjointness, set intersection, 3SUM, space-time tradeoff, conditional lower bounds

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.7

Related Version A full version of the paper is available at <https://arxiv.org/abs/1910.00831>.

Funding *Isaac Goldstein*: This research is supported by the Adams Foundation of the Israel Academy of Sciences and Humanities.

Moshe Lewenstein: This work was partially supported by ISF grant #1278/16.

Ely Porat: This work was partially supported by ISF grant #1278/16 and ERC grant MPM - 683064.

1 Introduction

The emerging field of fine-grained complexity receives much attention in the last years. One of the most notable pillars of this field is the celebrated 3SUM conjecture. In the 3SUM problem, given a set of n numbers we are required to decide if there are 3 numbers in this set that



© Isaac Goldstein, Moshe Lewenstein, and Ely Porat;
licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 7; pp. 7:1–7:22

Leibniz International Proceedings in Informatics



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

sum up to zero. It is conjectured that no truly subquadratic solution to this problem exists. This conjecture was extensively used to prove the conditional hardness of other problems in a variety of research areas, see e.g. [9, 10, 1, 2, 3, 4, 11, 20, 19, 24, 27, 33]. The 3SUM problem is closely related to the fundamental SetDisjointness problem. In the SetDisjointness problem we are given m sets S_1, S_2, \dots, S_m from some universe U for preprocessing. After the preprocessing phase, given a query pair of indices (i, j) we are required to decide if the intersection $S_i \cap S_j$ is empty or not. In the SetIntersection variant, all the elements within the intersection $S_i \cap S_j$ are required to be reported.

Cohen and Porat [15] investigated the upper bound of both problems. Specifically, they showed that SetDisjointness can be solved almost trivially in linear space and $O(\sqrt{N})$ query time, where N is the total number of elements in all sets. This solution can be generalized to a full tradeoff between the space S and the query time T such that $S \cdot T^2 = O(N^2)$. For the SetIntersection problem, Cohen and Porat demonstrated a linear space solution with $O(N\sqrt{N})$ preprocessing time and $O(\sqrt{N}\sqrt{out} + out)$ query time, where out is the output size. This was further generalized by Cohen [14] to a solution that uses $O(N^{2-2t})$ space with $O(N^{2-t})$ preprocessing time and $O(N^t out^{1-t} + out)$ query time for $0 \leq t \leq 1/2$.

From the lower bound perspective, Pătraşcu [27] proved the conditional *time* hardness of the multiphase problem, which is a dynamic version of the SetDisjointness problem, based on the 3SUM conjecture. He also proved a connection between 3SUM and reporting triangles in a graph which is closely related to the SetIntersection problem. His conditional hardness results were improved by Kopelowitz et al. [24] that considered the *preprocessing and query time* tradeoff of both SetDisjointness and SetIntersection. Specifically, they proved, based on the 3SUM conjecture, that SetDisjointness has the following lower bound on the tradeoff between preprocessing time T_p and query time T_q for any $0 < \gamma < 1$: $T_p + N^{\frac{1+\gamma}{2-\gamma}} T_q = \Omega(N^{\frac{2}{2-\gamma}-o(1)})$. Moreover, based on the 3SUM conjecture they also proved that SetIntersection has the following lower bound on the tradeoff between preprocessing, query and reporting (per output element) time for any $0 \leq \gamma < 1, \delta > 0$: $T_p + N^{\frac{2(1+\gamma)}{3+\delta-\gamma}} T_q + N^{\frac{2(2+\delta)}{3+\delta-\gamma}} T_r = \Omega(N^{\frac{4}{3+\delta-\gamma}-o(1)})$. Kopelowitz et al. [23] also proved the conditional *time* hardness of the *dynamic* versions of SetDisjointness and SetIntersection.

The lower bound on the *space-query time* tradeoff for solving SetDisjointness was considered by Cohen and Porat [16] and Pătraşcu and Roditty [28]. They have the following conjecture regarding the hardness of SetDisjointness (this is the formulation of Cohen and Porat. Pătraşcu and Roditty use slightly different formulation):

► **Conjecture 1 (SetDisjointness Conjecture).** Any data structure for the SetDisjointness problem with constant query time must use $\tilde{\Omega}(N^2)$ space.

Recently, Goldstein et al. [21] considered *space* conditional hardness in a broader sense and demonstrated the conditional hardness of SetDisjointness and SetIntersection with regard to their space-query time tradeoff. They had a generalized form of Conjecture 1 that claims that the whole (simple) space-time tradeoff upper-bound for SetDisjointness is tight:

► **Conjecture 2 (Strong SetDisjointness Conjecture).** Any data structure for the SetDisjointness problem that answers queries in T time must use $S = \tilde{\Omega}(\frac{N^2}{T^2})$ space.

Moreover, they also presented a conjecture regarding the space-time tradeoff for SetIntersection:

► **Conjecture 3 (Strong SetIntersection Conjecture).** Any data structure for the SetIntersection problem that answers queries in $O(T + out)$ time, where out is the size of the output of the query, must use $S = \tilde{\Omega}(\frac{N^2}{T})$ space.

Goldstein et al. [21] showed connections between these conjectures and other problems like 3SUM-Indexing (a data structure variant of 3SUM), k-Reachability and boolean matrix multiplication. Unconditional lower bounds for the *space-time* tradeoff of SetDisjointness and SetIntersection were proven by Dietz et al. [18] and Afshani and Nielsen [5] for specific models of computation. The results of Dietz et al. [18] implies that Conjecture 2 is true in the semi-group model. Afshani and Nielsen [5] proved Conjecture 3 in the pointer-machine model.

The fundamental nature of SetDisjointness and SetIntersection makes them useful for proving conditional lower bounds especially when considering their connection to the 3SUM problem. Indeed, several conditional lower bounds were proven using these problems (see [16, 17, 20, 24, 28, 29]). One major problem with this approach is that the universe of the elements in the sets of the SetDisjointness and SetIntersection problems can be large. This may cause the reduction from these problems to other problems, which we wish to prove their conditional hardness, to be inefficient. Therefore, it is of utmost interest to obtain a conditional lower bound on the hardness of SetDisjointness and SetIntersection with bounded universe, which in turn will be fruitful for achieving conditional lower bounds for other applications.

Our Results. In this paper we prove several conditional lower bounds for SetDisjointness and SetIntersection with bounded universe. We obtain the following results regarding the interplay between space and query time for solving these problems: (1) Based on the Strong SetDisjointness Conjecture, we prove that SetDisjointness with m sets from universe $[u]$ must either use $\Omega(m^{2-o(1)})$ space or have $\Omega(u^{1/2-o(1)})$ query time. (2) Based on the Strong SetDisjointness Conjecture, we prove that SetIntersection with m sets from universe $[u]$ must either use $\Omega(m^{2-o(1)})$ space or have $\tilde{\Omega}(u^{\alpha-o(1)} + out)$ query time, for any $1/2 \leq \alpha \leq 1$ and any output size out such that $out = \Omega(u^{2\alpha-1-\delta})$ and $\delta > 0$ (3) Based on the Strong SetIntersection Conjecture, we prove that SetIntersection with m sets from universe $[u]$ must either use $\Omega((m^2 u^\alpha)^{1-o(1)})$ space or have $\tilde{\Omega}(u^{\alpha-o(1)} + out)$ query time for any $1/2 \leq \alpha \leq 1$ and any output size out such that $out = \Omega(u^{2\alpha-1-\delta})$ and $\delta > 0$.

Regarding the interplay of preprocessing and query time we demonstrate a reduction from 3SUM to SetDisjointness and SetIntersection. Using this reduction we prove the following results based on the 3SUM conjecture: (i) Any solution to SetDisjointness with m sets from universe $[u]$ must either have $\Omega(m^{2-o(1)})$ preprocessing time or have $\Omega(u^{1/2-o(1)})$ query time. (ii) Any solution to SetIntersection with m sets from universe $[u]$ must either have $\Omega(m^{2-o(1)})$ preprocessing time or have $\Omega(u^{1-o(1)})$ query time.

These new conditional lower bounds are useful in proving conditional lower bounds for other problems that exploit the small universe size as explained before. We give some examples of such applications.

- (1) **Range Mode.** The Range Mode problem is a classic problem that was studied in several papers (see e.g. [12, 26]). In this problem, an array A with n elements is given for preprocessing. Then, we are required to answer range mode queries. That is, given a range $[i, j]$ we have to find the mode element (the most frequent element) in the range $[i, j]$ in A . The best known upper bound for the space-query time tradeoff of this problem is $S \cdot T^2 = \tilde{O}(n^2)$, where S is the space usage and T is the query time ([12, 26]). We prove using our new lower bound for SetDisjointness with bounded universe the following lower bound on the tradeoff between space and query time: $S \cdot T^4 = \Omega(n^{2-o(1)})$. We note that if the query time in the lower bound on SetDisjointness (in Theorem 4, see (1) above) was $\tilde{\Omega}(u^{1-o(1)})$ then the lower and upper bounds were tight.
- (2) **Distance oracle** is a data structure for computing the shortest path between any two vertices in a graph. We say that a distance oracle has a stretch t if for any two vertices in the graph the distance it returns is no more than t times the true distance between

these vertices. Approximate distance oracles were investigated in many papers (see for example [7, 6, 13, 28, 29, 30]). Agrawal [6] showed a $(\frac{5}{3})$ -stretch distance oracle for a graph $G = (V, E)$ that uses $\tilde{O}(|E| + \frac{|V|^2}{\alpha})$ space and has $O(\alpha \frac{|E|}{|V|})$ query time for any $1 \leq \alpha \leq (\frac{|V|^2}{|E|})^{\frac{1}{3}}$. We prove that this tradeoff is the best that can be achieved for any stretch-less-than-2 distance oracles based on our new lower bound on SetDisjointness with bounded universe (see a more detailed discussion in Section 4).

(3) **3SUM-Indexing** is a data structure variant of 3SUM. In this problem, two arrays A and B with n numbers in each of them are preprocessed. Then, given a query number z we are required to decide if there are $x \in A$ and $y \in B$ such that $x + y = z$. Goldstein et al. [21] conjecture that there is no $\tilde{O}(1)$ query time solution to 3SUM-Indexing using truly subquadratic space. In a stronger form of this conjecture they claim that there is no truly sublinear query time solution to 3SUM-Indexing using truly subquadratic space. Recently, it was proven that the strong 3SUM-Indexing conjecture is false [22, 25]. However, the exact interplay between time and space for solving 3SUM-Indexing is unclear and it still seems that there might be some strong variant of the 3SUM-Indexing conjecture that is true. Goldstein et al. [21] proved some connections between 3SUM-Indexing, SetDisjointness and SetIntersection. In this paper we strengthen their results using our new lower bounds for SetDisjointness and SetIntersection with bounded universe. Specifically, we prove based on our new lower bound on SetDisjointness with bounded universe that any solution to 3SUM-Indexing where the universe of the numbers within arrays A and B is $[n^{2+\epsilon}]$ for any $\epsilon > 0$ has this lower bound on the tradeoff between space (S) and query time (T): $S \cdot T^2 = \Omega(n^{2-o(1)})$. Moreover, we prove the same lower bound on the tradeoff between preprocessing (T_p) and query time (T_q): $T_p \cdot T_q^2 = \tilde{\Omega}(n^{2-o(1)})$. The latter is proven based on the 3SUM conjecture following our reduction to SetDisjointness with bounded universe.

In the 3SUM conjecture the universe of the numbers in the given instance is assumed to be $[n^3]$ (see [27]) or even $[n^4]$ (see [32]). It is known that 3SUM can be easily solved in $O(u \log u)$ time if the universe is $[u]$ by using FFT. Therefore, 3SUM with numbers from universe $[n^{2-\epsilon}]$ for any $\epsilon > 0$ can be solved in truly subquadratic time. Consequently, assuming that no truly subquadratic solution to 3SUM with universe $[n^2]$ seems to be much stronger conjecture (it was used once in [9]). Solving 3SUM-Indexing can be done easily with $\tilde{O}(n^2)$ preprocessing time, $O(n^2)$ space and $\tilde{O}(1)$ query time. Our results demonstrate that this is tight even if the universe of the numbers in A and B and the query numbers is $[n^{2+\epsilon}]$ for any $\epsilon > 0$. This is a very strong lower bound, as 3SUM-Indexing with numbers from universe $[u]$ can be solved with $\tilde{O}(u)$ preprocessing time, $O(u)$ space and $\tilde{O}(1)$ query time. This is done in a similar way to solving 3SUM with numbers from universe $[u]$. Consequently, for any $\epsilon > 0$, 3SUM-Indexing with numbers from universe $[n^{2-\epsilon}]$ can be solved by a data structure that has constant query, while the preprocessing time and space are subquadratic. Our new conditional lower bound demonstrates that having such a data structure for a slightly larger universe seems to be impossible.

2 Hardness of Space-Time Tradeoff for SD and SI with Bounded Universe

We prove the hardness of SetDisjointness with bounded universe in the following theorem:

► **Theorem 4.** *Any solution to SetDisjointness with sets $S_1, S_2, \dots, S_m \subseteq [u]$ for any value of $u \in [N^\delta, N]$, such that $N = \sum_{i=1}^m |S_i|$ and $\delta > 0$, must either use $\Omega(m^{2-o(1)})$ space or have $\tilde{\Omega}(u^{1/2-o(1)})$ query time, unless the Strong SetDisjointness Conjecture is false.*

Proof. Let us assume to the contradiction that the Strong SetDisjointness Conjecture is true, but there is an algorithm A that solves SetDisjointness on m sets from a universe $[u]$ and creates a data structure D , such that the space complexity of the data structure D is $O(m^{2-\epsilon_1})$ for some $\epsilon_1 > 0$ and the query time of algorithm A is $O(u^{1/2-\epsilon_2})$ for some $0 < \epsilon_2 \leq 1/2$. We define $\epsilon = \min(\epsilon_1, \epsilon_2)$.

Now, given an instance of SetDisjointness with sets $S'_1, S'_2, \dots, S'_{m'}$, we denote by N' the total number of elements in all sets, that is $N' = \sum_{i=1}^{m'} |S'_i|$. We rename the elements of all the sets such that each element e_i is mapped to some integer $x_i \in [N']$.

We distinguish between 3 types of sets:

- (a) **Large sets** are all the sets with more than \sqrt{u} elements. Denote by d the number of large sets. Let $S_{p_1}, S_{p_2}, \dots, S_{p_d}$ be some ordering of the large sets. Let p be a function such that $p(i) = p_j$ if S_i is the set S_{p_j} in the ordering of the large sets.
- (b) **Small sets** are all the sets with $O(u^{1/2-\epsilon})$ elements.
- (c) **Medium sets** are all the sets that are neither large nor small. Denote by e the number of medium sets. Let $S_{q_1}, S_{q_2}, \dots, S_{q_e}$ be some ordering of the medium sets. Let q be a function such that $q(i) = q_j$ if S_i is the set S_{q_j} in the ordering of the medium sets.

Now, we can solve SetDisjointness in the following way.

Preprocessing:

- (1) For any set S_i use static hashing to save all elements of the set in a table T_i , such that we can check if some element exists in the set in $O(1)$ time and the size of T_i is $O(|S_i|)$.
- (2) Maintain a $d \times (d + e)$ matrix M . The ℓ th row in this matrix represents the set S_{p_ℓ} . For $1 \leq \ell \leq d$, the ℓ th column represents S_{p_ℓ} and for $d + 1 \leq \ell \leq d + e$, the ℓ th column represents $S_{q_{\ell-d}}$.
- (3) For all pairs of sets S_i and S_j such that S_i is a large set and S_j is a large or medium set, save an explicit answer to the emptiness of the intersection of S_i and S_j in $M[p(i), p(j)]$ and $M[p(j), p(i)]$ if S_j is a large set and in $M[p(i), d + q(j)]$ if S_j is a medium set.
- (4) Pick $\log n$ hash functions $h_i : N \rightarrow [8u]$, for $1 \leq i \leq \log n$. Apply each h_i to all elements in all medium sets. Denote by $h_i(S_j)$ the set S_j after h_i has been applied to its elements.
- (5) For every $i, j \in [e]$, if $S_{q_i} \cap S_{q_j} = \emptyset$ do the following: Check if for all $k \in [\log n]$ there are $x \in S_{q_i}$ and $x' \in S_{q_j}$ such that $x \neq x'$ but $h_k(x) = h_k(x')$. If so, go back to step (4).
- (6) For every $k \in [\log n]$:
 - (6.1) Apply h_k to all the elements of all the medium sets.
 - (6.2) Use algorithm A to create a data structure D_k that solves the set disjointness problem on the medium sets $S_{q_1}, S_{q_2}, \dots, S_{q_e}$ after h_k has been applied to their elements.

Query: Given a pair of indices i and j , we need to determine if $S_i \cap S_j$ is empty or not. Without loss of generality we assume that $|S_i| < |S_j|$ and do the following:

- (1) If S_i is a small set:
 - (1.1) For each element $x \in S_i$: Check if $x \in S_j$ using table T_j . If so, return 0.
 - (1.2) Return 1.
- (2) If S_j is a large set:
 - (2.1) If S_i is a large set: Return $M[p(i), p(j)]$.
 - (2.2) If S_i is a medium set: Return and $M[p(j), d + q(i)]$.

(3) Else (if both S_i and S_j are medium sets):

For every $k \in [\log n]$, check by using algorithm A and the data structure D_k if S_i and S_j are disjoint.

If there is at least one value of k for which these sets are disjoint, return 1.

Otherwise, return 0.

Correctness. If at least one of the query sets is small then we can check if any of its elements is in the other query set using the hash tables that have been created in step (1) of the preprocessing phase. This is done in step (1) of the query algorithm. If at least one of the sets is large we can find the answer immediately by looking at the right position of matrix M that has been created in steps (2)-(3) of the preprocessing phase. The last option is that both query sets are medium. If this is the case we use the data structures that have been created in step (6) of the preprocessing phase. In steps (4) and (5) of the preprocessing phase we look for $\log n$ hash functions such that if any pair of sets are disjoint then they must be disjoint when applying the hash functions to their elements by at least one of the $\log n$ hash functions. Therefore, if any of the data structures that have been created in step (6) of the preprocessing phase reports that a pair of sets are disjoint they must be disjoint. Moreover, if a pair of sets are disjoint then there must be at least one data structure that reports that they are disjoint. This is checked in the step (3) of the query algorithm.

The last thing that needs to be justified is the existence of $\log n$ hash function such that for every pair of sets S_i and S_j that are disjoint they are also disjoint after applying the hash functions by at least one of the $\log n$ hash functions. The range of the hash function is $[8u]$. The number of elements in the medium sets is no more than \sqrt{u} . Therefore, for any two medium sets S_i and S_j and a hash function $h_k : N \rightarrow [8u]$ we have by the union-bound that $\Pr[\exists x_1 \in S_i, x_2 \in S_j : x_1 \neq x_2 \wedge h_k(x_1) = h_k(x_2)] \leq \frac{\sqrt{u} \cdot \sqrt{u}}{8u} = 1/8$. Consequently, the probability that a pair of disjoint medium sets S_i and S_j are not disjoint when applying h_k for all $k \in [\log n]$ is no more than $(1/8)^{\log n} = 1/n^3$. Therefore, the probability that any pair of disjoint medium sets are not disjoint when applying h_k for all $k \in [\log n]$ is no more than $n^2/n^3 = 1/n$ by the union-bound. Using the probabilistic method we get that there must be $\log n$ hash functions such that for every pair of sets S_i and S_j that are disjoint they are also disjoint after applying the hash functions by at least one of the $\log n$ hash functions.

Complexity analysis

Space complexity. The space for the tables in step (1) of the preprocessing is clearly $O(N)$ - linear in the total number of elements. The total number of large sets d is at most $O(N/u^{1/2})$. The total number of medium sets e is at most $O(N/u^{1/2-\epsilon})$. Therefore, the size of the matrix M is at most $O(N/u^{1/2} \cdot (N/u^{1/2} + N/u^{1/2-\epsilon})) = O(N^2/u^{1-\epsilon})$. There are $\log n$ data structures that are created in step (6). Each data structure uses at most $O((N/u^{1/2-\epsilon})^{2-\epsilon}) = O(N^{2-\epsilon}/u^{1-5\epsilon/2+\epsilon^2})$ space. Consequently, the total space complexity is $S = \tilde{O}(N^2/u^{1-\epsilon} + N^{2-\epsilon}/u^{1-5\epsilon/2+\epsilon^2})$.

Query time complexity. Step (1) of the query algorithm can be done in $O(u^{1/2-\epsilon})$ as this is the size of the largest small set. Step (2) is done in constant time by looking at the right position in M . In step (3) we do $\log n$ queries using algorithm A and the data structures D_k . The query time for each query is $O(u^{1/2-\epsilon})$ as the universe of the sets after applying any hash function h_k is $[8u]$. Therefore, the total query time is $T = O(u^{1/2-\epsilon})$.

Following our analysis we have that $S \cdot T^2 = \tilde{O}((N^2/u^{1-\epsilon} + N^{2-\epsilon}/u^{1-5\epsilon/2+\epsilon^2}) \cdot (u^{1/2-\epsilon})^2) = \tilde{O}(N^2u^{-\epsilon} + N^{2-\epsilon}u^{\epsilon/2-\epsilon^2}) = \tilde{O}(N^2u^{-\epsilon} + N^2u^{-\epsilon^2})$ (the last equality follows from the fact that $u \leq N$). This contradicts the Strong SetDisjointness Conjecture and therefore our assumption is false. \blacktriangleleft

From the proof of the above theorem we get a specific range for the value of m for hard instances of SetDisjointness. Bounding the value of m for hard instances may be useful for some specific applications. Therefore, we state the following corollary of the proof of Theorem 4:

► Corollary 5. *For any $\epsilon > 0$, any solution to set disjointness with sets $S_1, S_2, \dots, S_m \subseteq [u]$ for any value of $u \in [N^\delta, N]$, such that $N = \sum_{i=1}^m |S_i|$, $\delta > 0$ and the solution works for any value of m in the range $[\frac{N}{u^{1/2}}, \frac{N}{u^{1/2-\epsilon}}]$, must either use $\Omega(m^{2-o(1)})$ space or have $\Omega(u^{1/2-o(1)})$ query time, unless the Strong SetDisjointness Conjecture is false.*

We also prove conditional lower bounds on SetIntersection with bounded universe based on the Strong SetDisjointness Conjecture and the Strong SetIntersection Conjecture by generalizing the ideas from the previous proof. These results appear in Appendix A.

3 Hardness of Preprocessing-Query Time Tradeoff for SD and SI with Bounded Universe

We combine the ideas of Goldstein et al. [20] and Kopelowitz et al. [24] to get conditional lower bounds on the complexity of SetDisjointness with bounded universe. To achieve these bounds we prove the following lemma:

► Lemma 6. *Let X be any integer in $[n^\delta, n]$ for any $\delta > 0$. For any $\epsilon > 0$, an instance of 3SUM-Indexing that contains 2 arrays with n integers can be reduced to $2\epsilon \log X$ instances of SetDisjointness $SD_1, SD_2, \dots, SD_{2\epsilon \log X}$. For any $1 \leq i \leq 2\epsilon \log X$, instance SD_i have $N_i = n\sqrt{u_i}$ elements from universe $[u_i]$ and $m = n\sqrt{\frac{X}{u_i}}$ sets that each one of them is of size $O(\sqrt{u_i})$, where $u_i = X^{1+\epsilon}/2^{i-1}$. The time and space complexity of the reduction is truly subquadratic in n . Each query to the 3SUM-Indexing instance can be answered by at most $O(n/\sqrt{X})$ queries to each instance SD_i plus some additional time that is truly sublinear in n .*

Proof. We begin with an instance of 3SUM indexing with arrays A and B and do the following construction in order to reduce this 3SUM indexing instance to $2\epsilon \log n$ instances of SetDisjointness. The construction uses almost-linear and almost-balanced hash functions that serve as a useful tool in many reductions from 3SUM. We briefly define this notion here (see full details in [24, 31]). Let \mathcal{H} be a family of hash functions from $[u] \rightarrow [m]$. \mathcal{H} is called *linear* if for any $h \in \mathcal{H}$ and any $x, x' \in [u]$, we have $h(x) + h(x') \equiv h(x+x') \pmod{m}$. \mathcal{H} is called *almost-linear* if for any $h \in \mathcal{H}$ and any $x, x' \in [u]$, we have either $h(x) + h(x') \equiv h(x+x') + c_h \pmod{m}$, or $h(x) + h(x') \equiv h(x+x') + c_h + 1 \pmod{m}$, where c_h is an integer that depends only on the choice of h . For a function $h : [u] \rightarrow [m]$ and a set $S \subset [u]$ where $|S| = n$, we say that $i \in [m]$ is an overflowed value of h if $|\{x \in S : h(x) = i\}| > 3n/m$. \mathcal{H} is called *almost-balanced* if for a random $h \in \mathcal{H}$ and any set $S \subset [u]$ where $|S| = n$, the expected number of elements from S that are mapped to overflowed values is $O(m)$. For simplicity of presentation, we treat the almost-linear hash functions as linear and this only affects some constant factors in our analysis.

Construction

Initial Construction. We use an almost-linear almost-balanced hash function $h_1 : U \rightarrow [R]$ to map the elements of A to R buckets A_1, A_2, \dots, A_R such that $A_i = \{x \in A : h_1(x) = i\}$ and the elements of B to R buckets B_1, B_2, \dots, B_R such that $B_i = \{x \in B : h_1(x) = i\}$. As h_1 is almost-balanced the expected size of each bucket is $O(n/R)$. Moreover, buckets with more than $3n/R$ elements, called overflowed buckets, have no more than $O(R)$ elements in total. We save these $O(R)$ elements in lists L_A and L_B (we put elements from overflowed buckets of A in L_A and elements from overflowed buckets of B in L_B). We also sort A and B and save lookup tables for both A and B .

We pick another almost-linear almost-balanced hash function $h_2 : U \rightarrow [n]$. For each bucket A_i , we create an n -length characteristic vector v_{A_i} such that $v_{A_i}[j] = 1$ if there is $x \in A_i$ such that $h_2(x) = j$ and $v_{A_i}[j] = 0$ if there is no $x \in A_i$ such that $h_2(x) = j$. In the same way we create an n -length characteristic vector v_{B_j} for each bucket B_j .

Quad Trees Construction. We create a search quad tree for each pair of buckets A_i and B_j following the idea of Goldstein et al. [20]. The construction involves calculating the *convolution* of many pairs of vectors. The *convolution* of two vectors $u, v \in \{\mathbb{R}^+ \cup \{0\}\}^n$ is a vector c , such that $c[k] = \sum_{i=0}^k u[i]v[k-i]$ for $0 \leq k \leq 2n-2$. Constructing the quad tree is done as follows:

Quad-Tree-Construction(v_{A_i}, v_{B_j}, X).

- (1) For the bottom level of the quad tree:
 - (1.1) Partition the characteristic vector v_{A_i} into $\lceil n/X \rceil$ sub-vectors $v_{A_{i_1}}, \dots, v_{A_{i_{\lceil n/X \rceil}}}$ each of them of length X .
 - (1.2) Pad the last sub-vector with zeroes if needed.
 - (1.3) Let i_1, i_2, \dots, i_Y be the indices of the ones in some sub-vector $v_{A_{i_k}}$. If $Y > X/R$
 - (1.3.1) Duplicate $v_{A_{i_k}}$ $t = \lceil Y/(X/R) \rceil$ times.
 - (1.3.2) For every $p \in [t]$: Save in the p th copy of $v_{A_{i_k}}$ just the ones in the indices $i_{(p-1) \cdot (X/R) + 1}, \dots, i_{p \cdot (X/R) - 1}$. Replace all other ones by zeroes.
 - (1.4) Denote the sequence of sub-vectors of v_{A_i} and their duplicates by $P_{A_i} = v_{A_i}^1, v_{A_i}^2, \dots, v_{A_i}^{cn/X}$ for some constant $c \geq 1$. Order the sub-vectors in P_{A_i} by the locations of the ones. That is, sub-vector w occurs before u in P_{A_i} if the ones in w appear before the ones of u in v_{A_i} . A sub-vector w that contains only zeroes and therefore represents a sub-vector $v_{A_{i_k}}$ for some $1 < k \leq \lceil n/X \rceil$ without any duplicates appears before all sub-vectors $v_{A_{i_{k'}}$ for $k' > k$ and their duplicates.
 - (1.5) Repeat steps (1.1)-(1.4) for v_{B_j} and create a sequence of sub-vectors $P_{B_j} = v_{B_j}^1, v_{B_j}^2, \dots, v_{B_j}^{c'n/X}$ for some constant $c' \geq 1$.
 - (1.6) Without loss of generality let us assume that $c \geq c'$. Add to the end of the sequence P_{B_j} the vectors $v_{B_j}^{c'n/X+1}, \dots, v_{B_j}^{cn/X}$, such that each of these vectors contains exactly X zeroes.
 - (1.7) For each pair of sub-vectors $v_{A_i}^k$ and $v_{B_j}^\ell$:
 - (1.7.1) Create a node $c_{i,j}^{k,\ell}$ in the quad tree.
 - (1.7.2) Calculate the convolution of $v_{A_i}^k$ and $v_{B_j}^\ell$ and save the result in $c_{i,j}^{k,\ell}$.
- (2) For the next level of the quad tree upward:
 - (2.1) Create a sequence of sub-vectors $v_{A_i}^1, v_{A_i}^2, \dots, v_{A_i}^{cn/2X}$ such that $v_{A_i}^k$ is the concatenation of $v_{A_i}^{2k-1}$ and $v_{A_i}^{2k}$ from the previous level.

- (2.2) For every $v_{A_i}^k$ if there are overlapping locations in $v_{A_i}^{2k-1}$ and $v_{A_i}^{2k}$ - merge them. That is, if there are elements in both sub-vectors that represent the same interval of v_{A_i} , merge all of them in $v_{A_i}^k$ by setting each overlapping location to 1 if any of the two overlapping elements in this location is 1, and setting each overlapping location to 0 otherwise.
- (2.3) Repeat steps (2.1) and (2.2) for v_{B_j} and create a sequence $v_{B_j}^1, v_{B_j}^2, \dots, v_{B_j}^{cn/2X}$.
- (2.4) For each pair of sub-vectors $v_{A_i}^k$ and $v_{B_j}^\ell$ create a node $c_{i,j}^{k,\ell}$ in the quad tree.
- (2.5) Make the node $c_{i,j}^{k,\ell}$ the parent of 4 nodes from the previous level:
 $c_{i,j}^{2k-1,\ell-1}, c_{i,j}^{2k-1,\ell}, c_{i,j}^{2k,\ell-1}, c_{i,j}^{2k,2\ell}$.
- (2.6) Calculate the convolution of $v_{A_i}^k$ and $v_{B_j}^\ell$ and save the result in $c_{i,j}^{k,\ell}$. The convolution of $v_{A_i}^k$ and $v_{B_j}^\ell$ can be easily calculated using the convolution results that are saved in $c_{i,j}^{2k-1,\ell-1}, c_{i,j}^{2k-1,\ell}, c_{i,j}^{2k,\ell-1}, c_{i,j}^{2k,2\ell}$ from the previous level.
- (3) Repeat step (2) for all the levels up to the root. Notice that in the root we have the complete vectors v_{A_i} and v_{B_j} and we calculate and save their convolution within the root node.

We emphasize that in the bottom level of the quad tree the number of sub-vectors of v_{A_i} including all duplicates is no more than cn/X for some constant $c \geq 1$, as the total number of ones in v_{A_i} is $O(n/R)$. Therefore, the size of the sequence in step (1.4) is cn/X .

We call a quad tree such that the length of the sub-vectors in its bottom level is X X -quad-tree. We denote the level of the quad tree with sub-vectors of length Z by ℓ_Z . We emphasize that we consider the length of the sub-vectors for the last notation by their length if we do no merging in any level of the quad tree.

Convolution by SetDisjointness. The convolution c of two X -length vectors v and u can be calculated using SetDisjointness in the following way: Let us denote by v_i (for any $0 \leq i \leq X-1$) a $(2X-1)$ -length vector, such that $v_i[j+i] = v[j]$ for every $0 \leq j \leq X-1$ and all other elements of v_i are zeroes. It is clear that v_i is the vector v that its elements were shifted by i locations and the empty locations are filled with zeroes. Therefore, we call the vector v_i an i -shift of v . We define u_i in a similar way. Let us denote by v^R the vector v in reverse order of elements. It is straightforward to observe that $c[j]$ (the j th element in the convolution result of v and u) equals to the inner product of v_j^R (we note that the reverse operation is done before the shift operation) and u_{X-1} . Informally, the complete convolution of v and u can be calculated by the inner product of (padded) u and the reversed version of (padded) v in $X-1$ different shifts. We can reduce the number of shifts to v by shifting both v and u . Specifically, the value of $c[j]$ can be obtained by the inner product of $v_{j \bmod \sqrt{X}}^R$ and $u_{X-1-\lfloor \frac{j}{\sqrt{X}} \rfloor \cdot \sqrt{X}}$. Therefore, the convolution of v and u can be calculated by the inner product of $O(\sqrt{X})$ shifted versions of both v and u .

Each of the $(2X-1)$ -length boolean vectors can be represented by a set corresponding to the ones in the vector. Formally, for a vector w we construct a set S_w such that $S_w = \{j | w[j] = 1\}$. Instead of calculating the inner product of $v_{j \bmod \sqrt{X}}^R$ and $u_{X-1-\lfloor \frac{j}{\sqrt{X}} \rfloor \cdot \sqrt{X}}$, we can calculate $|S_{v_{j \bmod \sqrt{X}}^R} \cap S_{u_{X-1-\lfloor \frac{j}{\sqrt{X}} \rfloor \cdot \sqrt{X}}}|$ and get the same result. In our query process through the quad tree we just need to know in each node if the value in some position of the convolution within that node is zero or not. Thus, instead of calculating $|S_{v_{j \bmod \sqrt{X}}^R} \cap S_{u_{X-1-\lfloor \frac{j}{\sqrt{X}} \rfloor \cdot \sqrt{X}}}|$ we just need to determine if $S_{v_{j \bmod \sqrt{X}}^R} \cap S_{u_{X-1-\lfloor \frac{j}{\sqrt{X}} \rfloor \cdot \sqrt{X}}} = \emptyset$ or not. All in all, the convolution of two X -length vectors v and u can be determined by a

SetDisjointness instance that contains $O(\sqrt{X})$ sets such that their size equals to the number of ones in either v or u . Consequently, instead of saving explicitly the convolution result in each node in some level of the quad tree that represents sub-vectors of length X , we can create an instance of SetDisjointness that can be used to determine if a specific position in a convolution result is zero or not.

Hybrid Quad Tree Construction. Using the idea from the previous paragraph we modify the quad tree construction in the following way: We construct in the regular way, that is explained in detail above, each of the quad trees until level $\ell_{X^{1-\epsilon}}$. From level $\ell_{X^{1-\epsilon}}$ to level $\ell_{X^{1+\epsilon}}$ we do not save the convolution results explicitly in the quad tree for each level, but rather we create a SetDisjointness instance that can be used to answer if a specific position in a convolution result is zero or not. This is a hybrid construction in which we create an $(X^{1+\epsilon})$ -quad-tree that the bottom $X^{2\epsilon}$ levels are not saved explicitly. Instead, the information for these bottom levels is determined by the SetDisjointness instances we create. These levels are called the implicit levels of the hybrid quad tree while the levels in which we save the convolution results explicitly are called the explicit levels of the hybrid quad tree.

Query. Given a query integer number z , we search for a pair of integers $x \in A$ and $y \in B$ such that $x + y = z$. First of all, we check for each element $x \in L_A$ if there is $y \in B$ such that $x + y = z$ and we also check for each element $y \in L_B$ if there is $x \in A$ such that $x + y = z$. This can be done easily in $\tilde{O}(R)$ time using the sorted versions of A and B . Then, if x is in bucket A_i then by the (almost) linearity property of h_1 we expect y to be in bucket B_j such that $j = i - h_1(z)$. In order to find out if there is $x \in A_i$ and $y \in B_j$ such that $x + y = z$ we can calculate the convolution of v_{A_i} and v_{B_j} . Denote the vector that contains their convolution result by $C_{i,j}$. If $C_{i,j}[h_2(z)] = 0$ then there are no $x \in A_i$ and $y \in B_j$ such that $x + y = z$. However, if $C_{i,j}[h_2(z)] \neq 0$ then there may be $x \in A_i$ and $y \in B_j$ such that $x + y = z$, but it may also be the case that $h_2(x) + h_2(y) = h_2(z)$ while $x + y \neq z$. Therefore, in order to verify if there are $x \in A_i$ and $y \in B_j$ such that $x + y = z$, we need to find all pairs of $x' \in A_i$ and $y' \in B_j$ such that $h_2(x') + h_2(y') = h_2(z)$ and check if indeed $x' + y' = z$. There are exactly $C_{i,j}[h_2(z)]$ such pairs, which are also called witnesses.

In order to efficiently find the witnesses of $C_{i,j}[h_2(z)]$, we use the hybrid quad tree we have constructed for buckets A_i and B_j in the following way: We start at the root of the hybrid quad tree if the convolution result in the root is non-zero at location $h_2(z)$, we look at the children of the root node and continue the search at each child that contains a non-zero value in the convolution result it saves in the index that corresponds to index $h_2(z)$ of the convolution in the root. This way we continue downward all the way to the leaves. In the levels of the hybrid quad tree that the convolution results are not saved explicitly we query the SetDisjointness instances in order to get an indication for the existence of a witness in the search path from the root.

If we reach a leaf of the quad tree and the convolution result within this leaf is non-zero in the location that corresponds to the index $h_2(z)$ of the convolution in the root, then we do a “2SUM-like” search within this leaf.

The “2SUM-like” search is done as follows: Let us assume that the leaf represents 2 sub-vectors $v_{A_i}^k$ and $v_{B_j}^\ell$. We recover the original elements that these sub-vectors represent. Let the array A_i^k contain all $x \in A_i$ such that there is one in $v_{A_i}^k$ that corresponds to $h_2(x)$. In the same way we construct array B_j^ℓ . We sort both A_i^k and B_j^ℓ . Let d be the size of A_i^k . Then, if $A_i^k[d-1] + B_j^\ell[0] = z$ we are done. Otherwise, if the sum is greater than z we check if $A_i^k[d-2] + B_j^\ell[0] = z$ and if it is smaller than z we check if $A_i^k[d-1] + B_j^\ell[1] = z$. This way we continue until we get to the end of one of the arrays or find a pair of elements that its sum equals z .

Analysis. There are R^2 possible pairs of buckets A_i and B_j . Therefore, we construct R^2 quad trees. In order to save the convolution results in all the nodes in an explicit level ℓ_Z of some hybrid quad tree, the space we need to use is $O(n^2/Z)$ (for each pair A_i and B_j , there are $O(n^2/Z^2)$ pairs of sub-vectors one from v_{A_i} and the other from v_{B_j} . The size of the convolution of the two sub-vectors is $O(Z)$). Therefore, the total space for constructing the explicit levels of the hybrid quad trees is $\tilde{O}(n^2/X^{1+\epsilon} \cdot R^2)$ (a level that is closer to the root requires less space than a level that is farther away from the root. There are at most $\log n$ levels in each quad tree. The bottom explicit level is $\ell_{X^{1+\epsilon}}$). This is also the preprocessing time for constructing these levels of the hybrid quad trees as the convolution of two n -length vector can be calculated in $\tilde{O}(n)$ time.

From level $\ell_{X^{1-\epsilon}}$ to level $\ell_{X^{1+\epsilon}}$ we do not save the convolution results explicitly in the quad tree for each level, but rather we create a SetDisjointness instance that can be used to answer if a specific position in a convolution result is zero or not, as explained in detail previously. Let us analyse the cost of the SetDisjointness instance for some implicit level ℓ_Z . We have $O(R)$ buckets. Each bucket is represented by a characteristic vector that is partitioned into $O(n/Z)$ parts of length Z , such that each part contains $O(Z/R)$ ones. For each sub-vector we create $O(\sqrt{Z})$ sets that represent $O(\sqrt{Z})$ shifts of the sub-vector as explained previously. Therefore, the total number of sets we have is $O(R \cdot n/Z \cdot \sqrt{Z}) = O(nR/\sqrt{Z})$. Each set contains $O(Z/R)$ elements, so the total number of elements in all sets is $O(R \cdot n/\sqrt{Z} \cdot Z/R) = O(n\sqrt{Z})$. The universe of all the elements in the sets is Z .

For a query integer z we have $O(R)$ pairs of buckets A_i and B_j in which we may have two elements, one from each array, that sum up to z (as $j = i - h_1(z)$). For a pair of buckets A_i and B_j , we search for all the witnesses of $C_{i,j}[h_2(z)]$ in the quad tree of A_i and B_j . Searching for a witness from the root to a leaf of the quad tree can be done in $O(\log n)$ time in the levels we save the convolution explicitly and a constant number of queries for each SetDisjointness instance. Within a leaf we do a “2SUM-like” search on 2 arrays that contain $O(X^{1-\epsilon}/R)$ elements. Therefore, the total search time per witness is at most $\tilde{O}(X^{1-\epsilon}/R)$. A false witness is a witness pair of elements (x, y) such that $x + y \neq z$, but $h_2(x) + h_2(y) = h_2(z)$. The probability that a pair of numbers (x, y) is a false witness is $1/n$ (because the range of h_2 is $[n]$). Therefore, the expected number of false witnesses within a specific pair of buckets is at most $O((n/R)^2 \cdot 1/n) = O(n/R^2)$ by the union-bound (notice that the number of elements in each bucket is $O(n/R)$). Consequently, the total expected number of false witnesses is at most $O(Rn/R^2) = O(n/R)$. As explained before, the total search time per witness is at most $\tilde{O}(X^{1-\epsilon}/R)$. Thus, the total query time is $\tilde{O}(nX^{1-\epsilon}/R^2)$.

All in all, the total space and preprocessing time that is required by the explicit levels of the $O(R^2)$ hybrid quad trees is $\tilde{O}(n^2/X^{1+\epsilon} \cdot R^2)$ which is truly subquadratic in n if we set $R = \sqrt{X}$. Moreover, the total query time is $\tilde{O}(nX^{1-\epsilon}/R^2)$ which is truly sublinear in n if we set $R = \sqrt{X}$. Therefore, by setting $R = \sqrt{X}$ we have that the space and preprocessing time of the reduction is truly subquadratic in n . Additionally, a query can be answer by at most $O(n/\sqrt{X})$ queries to each SetDisjointness instance plus some additional time that is truly sublinear in n . ◀

▶ **Theorem 7.** *Any solution to SetDisjointness with sets $S_1, S_2, \dots, S_m \subseteq [u]$ for any value of $u \in [N^\delta, N]$, such that $N = \sum_{i=1}^m |S_i|$ and $\delta > 0$, must either have $\Omega(m^{2-o(1)})$ preprocessing time or have $\Omega(u^{1/2-o(1)})$ query time, unless the 3SUM Conjecture is false.*

Proof. Given an instance of the 3SUM problem that contains 3 arrays A, B and C with n numbers in each of them, we can solve this instance simply by creating a 3SUM indexing instance with arrays A and B and n queries - one for each number in C . Thus,

using the previous lemma the given 3SUM instance can be reduced for any integer value of X in $[n^\delta, n]$ (for any $\delta > 0$) and for any $\epsilon > 0$ to $2\epsilon \log X$ instances of SetDisjointness $SD_1, SD_2, \dots, SD_{2\epsilon \log X}$. For any $1 \leq i \leq 2\epsilon \log X$, instance SD_i have $N = n\sqrt{u_i}$ elements from universe $[u_i]$ and $m = n\sqrt{\frac{X}{u_i}}$ sets that each one of them is of size $O(\sqrt{u_i})$, where $u_i = X^{1+\epsilon}/2^{i-1}$. The total time for this reduction is $O(n^{2-\epsilon_1})$ for some $\epsilon_1 > 0$, and the total number of queries is $\tilde{O}(n^2/\sqrt{X})$. Consequently, if we assume to the contradiction that there is an algorithm that solves SetDisjointness on m sets from a universe $[u]$ with $O(m^{2-\epsilon_2})$ preprocessing time for some $\epsilon_2 > 0$ and $O(u^{1/2-\epsilon_3})$ query time for some $0 < \epsilon_3 \leq 1/2$, then we have a solution to 3SUM with $O(n^{2-\epsilon_1}) + \sum_{i=1}^{2\epsilon \log X} O((n\sqrt{\frac{X}{u_i}})^{2-\epsilon_2} + \frac{n^2}{\sqrt{X}}u_i^{1/2-\epsilon_3})$ time. We have that for any i , $u_i \leq X^{1+\epsilon}$ and $\sqrt{\frac{X}{u_i}} \leq \sqrt{\frac{X}{X^{1-\epsilon}}} = X^{\epsilon/2}$. Therefore, $\sum_{i=1}^{2\epsilon \log X} O((n\sqrt{\frac{X}{u_i}})^{2-\epsilon_2} + \frac{n^2}{\sqrt{X}}u_i^{1/2-\epsilon_3}) = \tilde{O}(n^{(1+\epsilon/2)(2-\epsilon_2)} + \frac{n^2}{\sqrt{X}}X^{(1+\epsilon)(1/2-\epsilon_3)})$. Thus, by setting $\epsilon = \min(\epsilon_2, \epsilon_3)$ we have a total running time that is truly subquadratic in n . This contradicts the 3SUM Conjecture. \blacktriangleleft

Another implication of our reduction in Lemma 6 is a similar reduction from 3SUM to SetIntersection. This reduction leads to a similar conditional lower bound on the preprocessing and query time tradeoff of SetIntersection with bounded universe. This is done in Appendix A.

4 Applications

In this section we present several applications of our lower bounds on SetDisjointness and SetIntersection with bounded universe. Several hardness results on the reporting variants of the problems in this section appear in Appendix B

4.1 Range Mode

As mentioned in the introduction, the range mode problem can be solved using S space and T query time such that: $S \cdot T^2 = \tilde{O}(n^2)$ [12, 26]. In the following Theorem we prove that $S \cdot T^4 = \tilde{\Omega}(n^2)$. This lower bound is proved based on the Strong SetDisjointness Conjecture using Theorem 4. We note that if the lower bound on the query time in Theorem 4 was $\Omega(u^{1-o(1)})$ instead of $\Omega(u^{1/2-o(1)})$ then the lower bound and upper bound were tight.

► Theorem 8. *Any data structure that answers Range Mode Queries in T time on a string of length n must use $S = \tilde{\Omega}(n^2/T^4)$ space, unless the Strong SetDisjointness Conjecture is false.*

Proof. We use the idea of Chan et al. [12] and apply our theorem on the hardness of SetDisjointness with bounded universe. We begin with an instance of SetIntersection with sets $S_1, S_2, \dots, S_m \subseteq [u]$ such that $u \in [N^\delta, N]$, $N = \sum_{i=1}^m |S_i|$ and $\delta > 0$. We create a string STR that is the concatenation of two string T_1 and T_2 of equal length. The string T_1 is the concatenation of the strings $T_{11}, T_{12}, \dots, T_{1m}$. For each i the string T_{1i} is of length u and each character in it is a different number in $[u]$. The prefix of T_{1i} contains all the numbers in $[u] \setminus S_i$ in a sorted order. This prefix is followed by all the numbers in S_i in a sorted order. This is called the suffix of T_{1i} . T_2 is constructed very similar to T_1 but with a change in the order of the suffix and prefix. Specifically, the string T_2 is given by the concatenation of the strings $T_{21}, T_{22}, \dots, T_{2m}$. For each i the string T_{2i} is of length u and each character in it is a different number in $[u]$. The prefix of T_{2i} contains all the numbers in S_i in a sorted order.

This prefix is followed by all the numbers in $[u] \setminus S_i$ in a sorted order. This is called the suffix of T_{2i} . For every $1 \leq i \leq m$, let us denote by a_i the index where the prefix of T_{1i} ends and by b_i the index where the prefix of T_{2i} ends.

The string STR is preprocessed for range mode queries. Then, given a query pair (i, j) for SetDisjointness, we need to decide if $S_i \cap S_j = \emptyset$ or not. This is done by a range mode query for the range $[a_i + 1, b_j]$. For every $p \in [2]$ and $q \in [m]$, the string T_{pq} contains characters that represent all the numbers in $[u]$, such that each of these numbers occurs exactly once in the string. Between T_{1i} and T_{2j} we have $m - i + j - 1$ substrings that each of them contains all the characters from $[u]$. Therefore, each character occurs $m - i + j - 1$ times between T_{1i} and T_{2j} . The suffix of T_{1i} starting at index $a_i + 1$ contains all the characters that represent the elements of S_i , while the prefix of T_{2j} ending at index b_j contains all the characters that represent the elements of S_j . Consequently, if there is an intersection between S_i and S_j we will have at least one character that occurs in both the suffix of T_{1i} and the prefix of T_{2j} . Thus, the mode of the range $[a_i + 1, b_j]$ will be $m - i + j + 1$ if $S_i \cap S_j \neq \emptyset$, and less than $m - i + j + 1$ if the $S_i \cap S_j = \emptyset$. Therefore, if we get from the range mode query a character c that occurs $m - i + j + 1$ times in the query range we know that the intersection is not empty, and if not we know that the intersection is empty. Even if the range mode query does not return the frequency of the mode within the query range, but rather just the mode element itself, we can save a hash table for every input set and use this tables to check in constant time if the returned element occurs in both S_i and S_j .

Consequently, an instance of SetDisjointness with m sets from universe $[u]$ (such that $u \in [N^\delta, N]$, $N = \sum_{i=1}^m |S_i|$ and $\delta > 0$), can be reduced to an instance of the range mode problem with a string of length $n = 2mu$, such that every query to the SetDisjointness instance can be answered by a query to the range mode instance. Let us assume to the contrary that the range mode problem can be solved by a data structure that answers queries in $\tilde{O}(T)$ time per query using $\tilde{O}(S)$ space such that $S \cdot T^4 = \tilde{O}(n^{2-\epsilon})$. Let $T = \tilde{O}(u^{1/2-\epsilon/4})$, we have that $S = \tilde{O}(n^{2\epsilon}/T^4) = \tilde{O}((mu)^{2-\epsilon}/u^{4(1/2-\epsilon/4)}) = \tilde{O}(m^{2-\epsilon}u^{2-\epsilon}/u^{2-\epsilon}) = \tilde{O}(m^{2-\epsilon})$. Therefore, we have a solution to SetDisjointness with m sets from universe $[u]$ with query time $\tilde{O}(u^{1/2-\epsilon/4})$ and space $\tilde{O}(mu + m^{2-\epsilon})$ (we add mu to the space usage, as we must at least save the string ST). According to Corollary 5 the reduction from general SetDisjointness to SetDisjointness with bounded universe holds for $N/\sqrt{u} \leq m$. Therefore, for any value of $u \leq N^{2/3-\epsilon}$ we have that $\sqrt{u} \leq N^{1/3-\epsilon/2}$. Thus, the following holds: $\sqrt{u} \leq N^{1/3-\epsilon/2} \Rightarrow \frac{1}{N^{1/3-\epsilon/2}} \leq \frac{1}{\sqrt{u}} \Rightarrow \frac{N}{N^{1/3-\epsilon/2}} \leq \frac{N}{\sqrt{u}} \Rightarrow N^{2/3+\epsilon/2} \leq \frac{N}{\sqrt{u}} \leq m \Rightarrow N^{2/3+\epsilon/2-\frac{2}{3}\epsilon-\frac{\epsilon^2}{2}} \leq m^{1-\epsilon}$. Consequently, we have that $u \leq N^{2/3-\epsilon} < N^{2/3-\epsilon/6-\epsilon/2} \leq N^{2/3-\epsilon/6-\epsilon^2/2} \leq m$. All in all, for any $u \leq N^{2/3-\epsilon}$ the reduction holds and $mu = \tilde{O}(m^{2-\epsilon})$. Consequently, the total space for solving SetDisjointness with bounded universe using our reduction to the range mode problem is $\tilde{O}(m^{2-\epsilon})$ and the query time is $\tilde{O}(u^{1/2-\epsilon/4})$. This contradicts the Strong SetDisjointness Conjecture according to Corollary 5. \blacktriangleleft

Using Theorem 7 and the same idea from the proof of Theorem 8, we obtain the following result regarding the preprocessing and query time tradeoff for solving the range mode problem:

► **Corollary 9.** *Any data structure that answers Range Mode Queries in T time on a string of length n must have $P = \tilde{\Omega}(n^2/T^4)$ preprocessing time, unless the 3SUM Conjecture is false.*

4.2 Distance Oracles

Agarwal [6] presented space-time tradeoffs for distance oracles for undirected graph $G = (V, E)$ with average degree μ (that is, $\mu = \frac{2|E|}{|V|}$): (i) $(1 + \frac{1}{k})$ -stretch distance oracles that use $\tilde{O}(|E| + \frac{|V|^2}{\alpha})$ space and have $O((\alpha\mu)^k)$ query time, for any $1 \leq \alpha \leq |V|$ (ii) $(1 + \frac{1}{k+0.5})$ -stretch distance oracles that use $\tilde{O}(|E| + \frac{|V|^2}{\alpha})$ space and have $O(\alpha(\alpha\mu)^k)$ query time, for any $1 \leq \alpha \leq |V|$. (iii) $(1 + \frac{2}{3})$ -stretch distance oracle that uses $\tilde{O}(|E| + \frac{|V|^2}{\alpha})$ space and has $O(\alpha\mu)$ query time for any $1 \leq \alpha \leq (\frac{|V|^2}{|E|})^{\frac{1}{3}}$. In the last result ((iii)) Agarwal managed to shave an α factor of the query time in (ii) (for $k = 1$). Therefore, both $\frac{5}{3}$ -stretch distance oracle and 2-stretch distance oracle (by setting $k = 1$ in (i)) have the same space-time tradeoff. It is known that 3-stretch distance oracle has a better tradeoff (see [8]). Moreover, by (i) and (ii) the tradeoff for stretch less than $5/3$ gets worse as the stretch guarantee is better. Thus, it seems natural to expect a better tradeoff for stretch more than $5/3$ and less-than-equal to 2.

In the following theorem we prove that improving the tradeoff of Agarwal [6] is impossible for any stretch $t \in [\frac{2}{3}, 2)$, unless the Strong SetDisjointness Conjecture is false:

► **Theorem 10.** *Any distance oracle for undirected graph $G = (V, E)$ with stretch less than 2 must either use $\Omega(|V|^{2-o(1)})$ space or have $\Omega(\mu^{1-o(1)})$ query time, where μ is the average degree of a vertex in G , unless Strong SetDisjointness Conjecture is false.*

Proof. We use the idea of Cohen and Porat [16] with our hardness results for SetDisjointness with bounded universe. Given an instance of SetDisjointness with sets $S_1, S_2, \dots, S_m \subseteq [u]$ such that $u \in [N^\delta, N]$, $N = \sum_{i=1}^m |S_i|$ and $\delta > 0$, we construct a bipartite graph $G = (V, E)$ as follows: In one side, we create a vertex v_i for each set S_i . In the other side, we create a vertex u_j for each element $j \in [u]$. For each element x in some set S_i we create an edge (v_i, u_x) . Formally, $V = \{v_i | 1 \leq i \leq m\} \cup \{u_j | j \in [u]\}$ and $E = \{(v_i, u_x) | x \in S_i\}$. For any $i, j \in [m]$, if $S_i \cap S_j \neq \emptyset$ then it is clear that the distance between v_i and v_j is exactly 2. Otherwise, the distance is at least 4. A stretch less-than 2 distance oracle can distinguish between these two possibilities and therefore a SetDisjointness query can be answered by one query to a stretch less-than 2 distance oracle for G .

It is clear that $|V| = m + u$ and $|E| = N$. We assume to the contradiction that there is a stretch less than two distance oracle that uses $\tilde{O}(|V|^{2-\epsilon_1})$ space and answers queries in $\tilde{O}(\mu^{1-\epsilon_2}) = \tilde{O}((\frac{|E|}{|V|})^{1-\epsilon_2})$ time, for some $\epsilon_1, \epsilon_2 > 0$. Therefore, SetDisjointness with bounded universe can be solved using $\tilde{O}((m + u)^{2-\epsilon_1})$ space and queries can be answered using $\tilde{O}((\frac{N}{m+u})^{1-\epsilon_2})$ time. According to Corollary 5 the reduction from general SetDisjointness to SetDisjointness with bounded universe holds for $N/\sqrt{u} \leq m$. Therefore, for any value of $u \leq N^{2/3}$ we have that the reduction holds and $u \leq m$ (see the full details in the proof of Theorem 8). Moreover, we have that $N/(m + u) \leq N/m \leq \sqrt{u}$. Consequently, for any $u \leq N^{2/3}$ we have a solution to SetDisjointness with bounded universe that uses $\tilde{O}((m + u)^{2-\epsilon_1}) = \tilde{O}(m^{2-\epsilon_1})$ space and answers queries in $\tilde{O}(\frac{N}{m+u}^{1-\epsilon_2}) = \tilde{O}((\sqrt{u})^{1-\epsilon_2}) = \tilde{O}(u^{1/2-\epsilon_2/2})$ time. This contradicts Strong SetDisjointness Conjecture according to Corollary 5. ◀

The previous theorem can be stated in a different way that makes it clear that the space-time tradeoff of Agarwal [6] is tight for distance oracles with stretch t such that $5/3 \leq t < 2$.

► **Corollary 11.** *There is no stretch less-than-2 distance oracle for undirected graph $G = (V, E)$ that uses $\tilde{O}(\frac{|V|^2}{\alpha})$ space and have $\tilde{O}(\alpha^{1-\epsilon}\mu)$ query time for any $|V|^\delta \leq \alpha$ and any $\delta, \epsilon > 0$, unless conjecture 1 is false.*

Using Theorem 7 and the same idea from the proof of Theorem 10, we obtain the following result regarding the preprocessing and query time tradeoff for distance oracles with stretch less-than-2:

► **Theorem 12.** *Any distance oracle for undirected graph $G = (V, E)$ with stretch less than 2 must either be constructed in $\Omega(|V|^{2-o(1)})$ preprocessing time or have $\Omega(\mu^{1-o(1)})$ query time, where μ is the average degree of a vertex in G , unless the 3SUM Conjecture is false.*

4.3 3SUM-Indexing with Small Universe

In the following theorem we prove a conditional lower bound on the space-time tradeoff for solving 3SUM-Indexing with universe size that is $[n^{2+\epsilon}]$ for any $\epsilon > 0$ (n is the size of the input arrays).

► **Theorem 13.** *For any $\epsilon > 0$ and $0 < \delta \leq 1$, any solution to 3SUM-Indexing with arrays $A = a_1, a_2, \dots, a_n$ and $B = b_1, b_2, \dots, b_n$ such that for every $i \in [n]$ $a_i, b_i \in [n^{2+\epsilon}]$ must either use $\Omega(n^{2-\delta-o(1)})$ space or have $\Omega(n^{\frac{\delta}{2}-o(1)})$ query time, unless Strong SetDisjointness Conjecture is false.*

Proof. We use the idea of Goldstein et al. [21] with our hardness for SetDisjointness with bounded universe. We begin with an instance of SetDisjointness with sets $S_1, S_2, \dots, S_m \subseteq [u]$ such that $u = N^\delta$, $m \in [\frac{N}{u^{1/2}}, \frac{N}{u^{1/2-\epsilon'}}]$, $N = \sum_{i=1}^m |S_i|$, $\epsilon' = \epsilon/2$ and $\delta > 0$.

For every element x in some set S_i we create two numbers $x_{1,i}$ and $x_{2,i}$. The number $x_{1,i}$ consists of 3 blocks of bits (ordered from the least significant bit toward the most significant bit): (i) A block of $\log m$ bits that contains the value of the index i . (ii) A block of $\log m$ padding zero bits. (iii) A block of $\log u$ bits that contains the value of $x - 1$. The number $x_{2,i}$ consists of 3 blocks of bits (ordered from the least significant bit toward the most significant bit): (i) A block of $\log m$ padding zero bits. (ii) A block of $\log m$ bits that contains the value of the index i . (iii) A block of $\log u$ bits that contains the value of $u - x$. We place the number $x_{1,i}$ in array A and the number $x_{2,i}$ in array B . The number of elements in each of these arrays is N , as we add a number to each array for every element in the input sets. These two arrays form an instance of 3SUM-Indexing which is preprocessed in order to answer queries.

Given a query asking whether $S_i \cap S_j = \emptyset$ or not, we can answer it by creating a query number z to the 3SUM-Indexing instance as follows: The number z consists of 3 blocks of bits (ordered from the least significant bit toward the most significant bit): (i) A block of $\log m$ bits that contain the value of the index i . (ii) A block of $\log m$ that contain the value of the index j . (iii) A block of $\log u$ bits that contains the value of $u - 1$. It straightforward to see that we get a positive answer to the query number z iff $S_i \cap S_j \neq \emptyset$: (i) If we have $x_{1,k_1} \in A$ and $y_{2,k_2} \in B$ such that $x_{1,k_1} + y_{2,k_2} = z$, then we must have that: (1) $k_1 = i$ which means that x is in S_i . (2) $k_2 = j$ which means that y is in S_j . (3) $x - 1 + u - y = u - 1$ which means that $x = y$. (ii) If $S_i \cap S_j \neq \emptyset$ then there is an element x such that $x \in S_i$ and $x \in S_j$. From our construction it is clear that indeed $x_{1,i} + x_{2,j} = z$.

Thus, we have reduced our SetDisjointness instance to an instance of 3SUM-Indexing such that each query to the SetDisjointness instance can be answered by a query to the 3SUM-Indexing instance. The size of each array in the 3SUM-Indexing instance is N . All the numbers in these arrays have $2 \log m + \log u$ bits. Let $u = N^\delta$ and $m \in [\frac{N}{u^{1/2}}, \frac{N}{u^{1/2-\epsilon'}}]$, for $\epsilon' \leq \frac{\epsilon}{2}$, then the number of bits in each number of A and B is bounded by $2 \log \frac{N}{u^{1/2-\epsilon'}} + \log N^\delta = 2 \log N^{1-\delta/2+\epsilon'\delta} + \log N^\delta = 2(1-\delta/2+\epsilon'\delta) \log N + \delta \log N = (2+2\epsilon'\delta) \log N \leq (2+\epsilon) \log N$. By setting $n = N$ we have that both A and B have n elements and all the numbers are in $[n^{2+\epsilon}]$.

We assume to the contradiction that 3SUM-Indexing with universe $[n^{2+\epsilon}]$ can be solved using $\tilde{O}(n^{2-\delta-\gamma_1})$ space, while answering queries in $\tilde{O}(n^{\frac{\delta}{2}-\gamma_2})$ time, for some $\gamma_1, \gamma_2 > 0$. Following our reduction this means that we can solve SetDisjointness with m from universe $[u]$ using $S = \tilde{O}(n^{2-\delta-\gamma_1})$ space, while answering queries in $T = \tilde{O}(n^{\frac{\delta}{2}-\gamma_2})$ time. We have that $u = n^\delta$, so $n = u^{1/\delta}$. Moreover, $m \geq n^{1-\delta/2}$, so $n \leq m^{1/(1-\delta/2)}$. Therefore, $S = \tilde{O}(m^{(2-\delta-\gamma_1)/(1-\delta/2)}) = \tilde{O}(m^{2-\frac{\gamma_1}{1-\frac{\delta}{2}}})$ and $T = \tilde{O}(u^{(\frac{\delta}{2}-\gamma_2)/\delta}) = \tilde{O}(u^{1/2-\gamma_2/\delta})$. This contradicts Corollary 5. ◀

Using Theorem 7 and the same idea from the proof of Theorem 13, we obtain the following result regarding the preprocessing and query time tradeoff for distance oracles with stretch less-than-2:

► **Theorem 14.** *For any $\epsilon > 0$ and $0 < \delta \leq 1$, any solution to 3SUM-Indexing with arrays $A = a_1, a_2, \dots, a_n$ and $B = b_1, b_2, \dots, b_n$ such that for every $i \in [n]$ $a_i, b_i \in [n^{2+\epsilon}]$ must either have $\Omega(n^{2-\delta-o(1)})$ preprocessing time or have $\Omega(n^{\frac{\delta}{2}-o(1)})$ query time, unless the 3SUM Conjecture is false.*

References

- 1 Amir Abboud and Kevin Lewi. Exact Weight Subgraphs and the k-Sum Conjecture. In *International Colloquium on Automata, Languages and Programming, ICALP 2013*, pages 1–12, 2013.
- 2 Amir Abboud and Virginia Vassilevska Williams. Popular Conjectures Imply Strong Lower Bounds for Dynamic Problems. In *Foundations of Computer Science, FOCS 2014*, pages 434–443, 2014.
- 3 Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of Faster Alignment of Sequences. In *International Colloquium on Automata, Languages and Programming, ICALP 2014*, pages 39–51, 2014.
- 4 Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching Triangles and Basing Hardness on an Extremely Popular Conjecture. In *Symposium on Theory of Computing, STOC 2015*, pages 41–50, 2015.
- 5 Peyman Afshani and Jesper Sindahl Nielsen. Data Structure Lower Bounds for Document Indexing Problems. In *International Colloquium on Automata, Languages, and Programming, ICALP 2016*, pages 93:1–93:15, 2016.
- 6 Rachit Agarwal. The Space-Stretch-Time Tradeoff in Distance Oracles. In *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, pages 49–60, 2014.
- 7 Rachit Agarwal and Philip Brighten Godfrey. Distance Oracles for Stretch Less Than 2. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 526–538, 2013.
- 8 Rachit Agarwal, Philip Brighten Godfrey, and Sarel Har-Peled. Approximate distance queries and compact routing in sparse graphs. In *INFOCOM 2011. 30th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 10-15 April 2011, Shanghai, China*, pages 1754–1762, 2011.
- 9 Amihoud Amir, Timothy M. Chan, Moshe Lewenstein, and Noa Lewenstein. On Hardness of Jumbled Indexing. In *International Colloquium on Automata, Languages and Programming, ICALP 2014*, pages 114–125, 2014.
- 10 Amihoud Amir, Tsvi Kopelowitz, Avivit Levy, Seth Pettie, Ely Porat, and B. Riva Shalom. Mind the Gap: Essentially Optimal Algorithms for Online Dictionary Matching with One Gap. In *International Symposium on Algorithms and Computation, ISAAC 2016*, pages 12:1–12:12, 2016.

- 11 Gill Barequet and Sarel Har-Peled. Polygon-containment and Translational min-Hausdorff-Distance between segment Sets are 3SUM-hard. In *Symposium on Discrete Algorithms, SODA 1999*, pages 862–863, 1999.
- 12 Timothy M. Chan, Stephane Durocher, Kasper Green Larsen, Jason Morrison, and Bryan T. Wilkinson. Linear-Space Data Structures for Range Mode Query in Arrays. *Theory Comput. Syst.*, 55(4):719–741, 2014.
- 13 Shiri Chechik. Approximate distance oracles with constant query time. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 654–663, 2014.
- 14 Hagai Cohen. Fast Set Intersection and Two-Patterns Matching. Master’s thesis, Bar-Ilan University, Ramat-Gan, Israel, 2010.
- 15 Hagai Cohen and Ely Porat. Fast set intersection and two-patterns matching. *Theor. Comput. Sci.*, 411(40-42):3795–3800, 2010.
- 16 Hagai Cohen and Ely Porat. On the hardness of distance oracle for sparse graph. *CoRR*, abs/1006.1117, 2010. [arXiv:1006.1117](https://arxiv.org/abs/1006.1117).
- 17 Pooya Davoodi, Michiel H. M. Smid, and Freek van Walderveen. Two-Dimensional Range Diameter Queries. In *LATIN 2012: Theoretical Informatics - 10th Latin American Symposium, Arequipa, Peru, April 16-20, 2012. Proceedings*, pages 219–230, 2012.
- 18 Paul F. Dietz, Kurt Mehlhorn, Rajeev Raman, and Christian Uhrig. Lower Bounds for Set Intersection Queries. *Algorithmica*, 14(2):154–168, 1995.
- 19 Anka Gajentaan and Mark H. Overmars. On a Class of $O(n^2)$ Problems in Computational Geometry. *Comput. Geom.*, 5:165–185, 1995.
- 20 Isaac Goldstein, Tsvi Kopelowitz, Moshe Lewenstein, and Ely Porat. How Hard is it to Find (Honest) Witnesses? In *European Symposium on Algorithms, ESA 2016*, pages 45:1–45:16, 2016.
- 21 Isaac Goldstein, Tsvi Kopelowitz, Moshe Lewenstein, and Ely Porat. Conditional Lower Bounds for Space/Time Tradeoffs. In *Algorithms and Data Structures Symposium, WADS 2017*, pages 421–436, 2017.
- 22 Alexander Golovnev, Siyao Guo, Thibaut Horel, Sunoo Park, and Vinod Vaikuntanathan. 3SUM with Preprocessing: Algorithms, Lower Bounds and Cryptographic Applications. *arXiv*, 2019. [arXiv:1907.08355](https://arxiv.org/abs/1907.08355).
- 23 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Dynamic Set Intersection. In *Algorithms and Data Structures - 14th International Symposium, WADS 2015, Victoria, BC, Canada, August 5-7, 2015. Proceedings*, pages 470–481, 2015.
- 24 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher Lower Bounds from the 3SUM Conjecture. In *Symposium on Discrete Algorithms, SODA 2016*, pages 1272–1287, 2016.
- 25 Tsvi Kopelowitz and Ely Porat. The Strong 3SUM-INDEXING Conjecture is False. *arXiv*, 2019. [arXiv:1907.11206](https://arxiv.org/abs/1907.11206).
- 26 Danny Krizanc, Pat Morin, and Michiel H. M. Smid. Range Mode and Range Median Queries on Lists and Trees. *Nord. J. Comput.*, 12(1):1–17, 2005.
- 27 Mihai Patrascu. Towards polynomial lower bounds for dynamic problems. In *Symposium on Theory of Computing, STOC 2010*, pages 603–610, 2010.
- 28 Mihai Patrascu and Liam Roditty. Distance Oracles beyond the Thorup-Zwick Bound. *SIAM J. Comput.*, 43(1):300–311, 2014.
- 29 Mihai Patrascu, Liam Roditty, and Mikkel Thorup. A New Infinity of Distance Oracles for Sparse Graphs. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 738–747, 2012.
- 30 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005.
- 31 Joshua R. Wang. Space-Efficient Randomized Algorithms for K-SUM. In *European Symposium on Algorithms, ESA 2014*, pages 810–829, 2014.
- 32 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *International Congress of Mathematicians, ICM 2018*, 2018.
- 33 Virginia Vassilevska Williams and Ryan Williams. Finding, Minimizing, and Counting Weighted Subgraphs. *SIAM J. Comput.*, 42(3):831–854, 2013.

A Conditional Lower Bounds for SetIntersection

In the following theorem we prove a conditional lower bound on SetIntersection with bounded universe based on the Strong SetDisjointness Conjecture by generalizing the ideas from Theorem 4. Specifically, we demonstrate that for SetIntersection we either have the same space lower bound as for SetDisjointness or we have a $\tilde{\Omega}(u^{1-o(1)} + out)$ bound on the query time. The query time bound is stronger than the $\Omega(u^{1/2-o(1)})$ bound that we have for SetDisjointness. However, we argue that this lower bound for SetIntersection holds only when the output is large. If we have an upper bound on the size of the output we still have a lower bound on the query time, but this lower bound gets closer to $\tilde{\Omega}(u^{1/2-o(1)} + out)$ as the size of the output gets smaller. Eventually, this coincides with the lower bound we have for SetDisjointness (notice that in order to answer SetDisjointness queries we just need to output a single element from the intersection if there is any).

► **Theorem 15.** *Any solution to SetIntersection with sets $S_1, S_2, \dots, S_m \subseteq [u]$ for any value of $u \in [N^\delta, N]$, such that $N = \sum_{i=1}^m |S_i|$ and $\delta > 0$, must either use $\Omega(m^{2-o(1)})$ space or have $\tilde{\Omega}(u^{\alpha-o(1)} + out)$ query time, for any $1/2 \leq \alpha \leq 1$ and any output size out such that $out = \Omega(u^{2\alpha-1-\delta})$ and $\delta > 0$, unless Strong SetDisjointness Conjecture is false.*

Proof. We use the same idea as in the proof of Theorem 4. Let us assume to the contradiction that Strong SetDisjointness Conjecture is true but there is an algorithm A' that solves SetIntersection on m sets from a universe $[u]$ and creates a data structure D , such that the space complexity of the data structure D is $O(m^{2-\epsilon_1})$ for some $\epsilon_1 > 0$ and the query time of algorithm A' is $O(u^{\alpha-\epsilon_2})$ for some $0 < \epsilon_2 \leq 1/2$. We define $\epsilon = \min(\epsilon_1, \epsilon_2)$.

In the proof, we call those sets with at least $u^{\alpha-3/4\epsilon}$ elements *large sets* and those sets with at most $O(u^{\alpha-\epsilon})$ elements *small sets*. All other sets are called *medium sets*.

SetIntersection (for general universe) can be solved in the following way:

The preprocessing phase is similar to the one that is done in the proof of Theorem 4 with the following changes: 1. In step (5) we check for each pair of medium sets S_i and S_j such that $S_i \cap S_j = \emptyset$ that the size of $h_k(S_i) \cap h_k(S_j)$ is no more than $u^{2\alpha-1-3/2\epsilon}$ for at least one $h_k : U \rightarrow [8u]$ that we pick in step (4). This is done instead of just checking for the emptiness of $h_k(S_i) \cap h_k(S_j)$. 2. In step (6.2) we use algorithm A' to create a data structure D_k that solves the SetIntersection problem instead of the SetDisjointness problem.

The query phase is also very similar to the one from Theorem 4 with the following change: In step (3), for each k , we get one by one the elements in the intersection of $h_k(S_i)$ and $h_k(S_j)$ by querying the data structure D_k . For each element e in that intersection we verify that it is contained in both S_i and S_j using the tables T_i and T_j . If this is the case, then we return that the sets are not disjoint. Otherwise, we add one to a counter of the number of elements in $(h_k(S_i) \cap h_k(S_j)) \setminus (S_i \cap S_j)$. If this counter exceeds $u^{2\alpha-1-3/2\epsilon}$ we stop the query immediately and continue to the next value of k .

The correctness of this reduction follows from the same arguments as in the proof of Theorem 4. The difference is in analysing the hash functions and their properties. For any two unequal elements $x_1 \in S_i$ and $x_2 \in S_j$, where both S_i and S_j are medium sets, and for any hash function $h_k : N \rightarrow [8u]$ we have that $\Pr[h_k(x_1) = h_k(x_2)] \leq 1/(8u)$. We call two unequal elements $x_1 \in S_i$ and $x_2 \in S_j$ such that $h_k(x_1) = h_k(x_2)$ a false-positive of h_k . The number of elements in the medium sets is no more than $u^{\alpha-3/4\epsilon}$. Consequently, the expected number of false-positives in $h_k(S_i) \cap h_k(S_j)$ is no more than $(u^{\alpha-3/4\epsilon})^2/8u = u^{2\alpha-1-3/2\epsilon}/8$. By Markov inequality the probability that the number of false-positives for a specific h_k is more than $u^{2\alpha-1-3/2\epsilon}$ is no more than $1/8$. Therefore, the probability that a pair of medium sets S_i and S_j has more than $u^{2\alpha-1-3/2\epsilon}$ false-positives when applying h_k for all $k \in [\log n]$

is no more than $(1/8)^{\log n} = 1/n^3$. Thus, the probability that the number of false-positives for any pair of medium sets is more than $u^{2\alpha-1-3/2\epsilon}$ when applying h_k for all $k \in [\log n]$ is no more than $n^2/n^3 = 1/n$ by the union-bound. Using the probability method we get that there must be $\log n$ hash functions such that for every pair of medium sets S_i and S_j the number of false-positives is no more than $u^{2\alpha-1-3/2\epsilon}$ after applying the hash functions by at least one of the $\log n$ hash functions.

Complexity analysis

Space complexity. The space for the tables in step (1) of the preprocessing is clearly $O(N)$ - linear in the total number of elements. The total number of large sets d is at most $O(N/u^{\alpha-3/4\epsilon})$. The total number of medium sets e is at most $O(N/u^{\alpha-\epsilon})$. Therefore, the size of the matrix M is at most $O(N/u^{\alpha-3/4\epsilon} \cdot N/u^{\alpha-\epsilon}) = O(N^2/u^{2\alpha-7/4\epsilon})$. There are $\log n$ data structures that are created in step (6). Each data structure uses at most $O((N/u^{\alpha-\epsilon})^{2-\epsilon}) = O(N^{2-\epsilon}/u^{2\alpha-(2+\alpha)\epsilon+\epsilon^2})$ space. Consequently, the total space complexity is $S = \tilde{O}(N^2/u^{2\alpha-7/4\epsilon} + N^{2-\epsilon}/u^{2\alpha-(2+\alpha)\epsilon+\epsilon^2})$.

Query time complexity. Step (1) of the query algorithm can be done in $O(u^{\alpha-\epsilon})$ as this is the size of the largest small set. Step (2) is done in constant time by looking at the right position in M . In step (3) we do $\log n$ queries using algorithm A' and the data structures D_k . the universe of the sets after applying any hash function h_k is $[8u]$, so the query time for each query is $O(u^{\alpha-\epsilon} + out)$ (out is the size of the output we get from the query). We do not allow the query to output more than $u^{2\alpha-1-3/2\epsilon} < u^{\alpha-\epsilon}$ elements. Therefore, the total query time is $T = O(u^{\alpha-\epsilon})$.

Following our analysis we have that $S \cdot T^2 = \tilde{O}((N^2/u^{2\alpha-7/4\epsilon} + N^{2-\epsilon}/u^{2\alpha-(2+\alpha)\epsilon+\epsilon^2}) \cdot (u^{\alpha-\epsilon})^2) = \tilde{O}(N^2 u^{-1/4\epsilon} + N^{2-\epsilon} u^{\alpha-\epsilon^2})$. As $\alpha \leq 1$ and $u \leq N$, we have that $u^{\alpha\epsilon} \leq N^\epsilon$. Therefore, $S \cdot T^2 = \tilde{O}(N^2 u^{-1/4\epsilon} + N^2 u^{-\epsilon^2})$. This contradicts the Strong SetDisjointness Conjecture and therefore our assumption is false. \blacktriangleleft

A better lower bound on the space complexity for solving SetIntersection can be obtained based on the Strong SetIntersection Conjecture. This is demonstrated by the following theorem:

► Theorem 16. *Any solution to SetIntersection with sets $S_1, S_2, \dots, S_m \subseteq [u]$ for any value of $u \in [N^\delta, N]$, such that $N = \sum_{i=1}^m |S_i|$ and $\delta > 0$, must either use $\Omega((m^2 u^\alpha)^{1-o(1)})$ space or have $\tilde{\Omega}(u^{\alpha-o(1)} + out)$ query time for any $1/2 \leq \alpha \leq 1$ and any output size out such that $out = \Omega(u^{2\alpha-1-\delta})$ and $\delta > 0$, unless Strong SetIntersection Conjecture is false.*

Proof. The proof is very similar to the proof of Theorem 4. Let us assume to the contradiction that the Strong SetIntersection Conjecture is true but there is an algorithm A' that solves SetIntersection on m sets from a universe $[u]$ and creates a data structure D , such that the space complexity of the data structure D is $O(m^2 u^\alpha)^{1-\epsilon_1}$ for some $\epsilon_1 > 0$ and the query time of algorithm A' is $O(u^{\alpha-\epsilon_2})$ for some $0 < \epsilon_2 \leq 1/2$. We define $\epsilon = \min(\epsilon_1, \epsilon_2)$.

In order to solve SetIntersection for general universe we use almost the same preprocessing and query procedures as in the the proof of Theorem 4 except for the following changes: 1. In the preprocessing phase, we do not save in matrix M in the entries $M[p(i), p(j)]$ or $M[p(i), d + q(j)]$ just the answer to the emptiness of the intersection of S_i and S_j , but rather we save in this location a list of all the elements within the intersection of S_i and S_j . 2. In the query phase, in step (2) we return a list of elements and not just a single bit. 3. In the query phase, in step (3) for each k we get the intersection of $h_k(S_i)$ and $h_k(S_j)$ by querying

the data structure D_k . For each element e in that intersection we return it after verifying that it is contained in both S_i and S_j using the tables T_i and T_j . Moreover, we count the number of elements in $(h_k(S_i) \cap h_k(S_j)) \setminus (S_i \cap S_j)$ as we get them from the query and if they exceed $u^{2\alpha-1-3/2\epsilon}$ we stop the query immediately and continue with the next value of k .

The correctness of the above solution to set intersection follows from the same arguments as in the proof of Theorem 15.

Complexity analysis

Space complexity. The space for the tables in step (1) of the preprocessing is clearly $O(N)$. Matrix M in this solution contains in each entry the complete list of elements in the intersection of some pair of sets. The total number of large sets d is at most $O(N/u^{\alpha-3/4\epsilon})$. The total number of medium sets e is at most $O(N/u^{\alpha-\epsilon})$. The total number of elements in all sets is N . Therefore, the size of the matrix M is at most $O(N/u^{\alpha-3/4\epsilon} \cdot N/u^{\alpha-\epsilon} \cdot u^{\alpha-\epsilon}) = O(N^2/u^{\alpha-3/4\epsilon})$ (see the full details in the full version of this paper). There are $\log n$ data structures that are created in step (6). Each data structure use at most $O(((N/u^{\alpha-\epsilon})^2 u^\alpha)^{1-\epsilon}) = O(N^{2-2\epsilon}/u^{\alpha-(2+\alpha)\epsilon+2\epsilon^2})$ space. Consequently, the total space complexity is $S = \tilde{O}(N^2/u^{\alpha-3/4\epsilon} + N^{2-2\epsilon}/u^{\alpha-(2+\alpha)\epsilon+2\epsilon^2})$.

Query time complexity. Step (1) of the query algorithm can be done in $O(u^{\alpha-\epsilon})$ as this is the size of the largest small set. Step (2) is done in constant time plus the output size by looking at the right position in M . In step (3) we do $\log n$ queries using algorithm A' and the data structures D_k . The universe of the sets after applying any hash function h_k is $[u]$, so the query time for each query is $O(u^{\alpha-\epsilon} + out)$ (out is the size of the output we get from the query). We do not allow the query to output more than $u^{2\alpha-1-3/2\epsilon} < u^{\alpha-\epsilon}$ false-positive elements. Therefore, the total query time is $O(T + out)$, where $T = O(u^{\alpha-\epsilon})$.

Following our analysis we have that $S \cdot T = \tilde{O}((N^2/u^{\alpha-3/4\epsilon} + N^{2-2\epsilon}/u^{\alpha-(2+\alpha)\epsilon+2\epsilon^2}) \cdot (u^{\alpha-\epsilon})) = \tilde{O}(N^2 u^{-1/4\epsilon} + N^{2-2\epsilon} u^{(1+\alpha)\epsilon-2\epsilon^2})$. As $\alpha \leq 1$ and $u \leq N$, we have that $u^{(1+\alpha)\epsilon} \leq N^{2\epsilon}$. Therefore, $S \cdot T = \tilde{O}(N^2 u^{-1/4\epsilon} + N^2 u^{-2\epsilon^2})$. This contradicts the Strong SetIntersection Conjecture and therefore our assumption is false. \blacktriangleleft

The construction in the proof of Lemma 6 can be modified in order to obtain the following reduction from 3SUM-Indexing to SetIntersection:

► Lemma 17. *For any $0 < \gamma < \delta \leq 1$, an instance of 3SUM-Indexing that contains 2 arrays with n integers can be reduced to an instance SI of SetIntersection. The instance SI have $N = n\sqrt{u}$ elements from universe $[u]$ and $m = n^{1+\gamma-\delta/2}$ sets that each one of them is of size $O(\sqrt{u})$, where $u = n^\delta$ and $0 < 2\gamma < \delta \leq 1$. The time and space complexity of the reduction is $\tilde{O}(n^{2+2\gamma-\delta})$. Each query to the 3SUM-Indexing instance can be answered by at most $O(n^{1+\gamma-\delta})$ queries to SI plus some additional $O(\log n)$ time.*

Proof. We follow the construction from the proof of Lemma 6. In each quad tree we construct for some two buckets A_i and B_j , we save the convolution results of the corresponding sub-vectors until the bottom level in which the size of each subvector is X . In this level, for each pair of sub-vectors we create $O(\sqrt{X})$ sets (representing different shifts) in the same way we construct the sets for the SetDisjointness instances in the proof of Lemma 6. These sets form a SetIntersection instance that contains $O(R \cdot n/X \cdot \sqrt{X}) = O(nR/\sqrt{X})$ sets. In the query phase, whenever we search a quad tree and get to a leaf node we can immediately report all pairs of elements that are witnesses for $C_{i,j}[h_2(z)]$. This is easily done by a single SetIntersection query. The number of sub-vectors in the bottom level is $O(n/X)$ for both

v_{A_i} and v_{B_j} . For every sub-vector of v_{A_i} there are at most $O(1)$ sub-vectors of v_{B_j} that their convolution with v_{A_i} may contain a witness pair for $C_{i,j}[h_2(z)]$. Consequently, we do at most $O(n/X)$ intersection queries within each quad tree.

Therefore, the total space for constructing the quad trees' levels with explicit convolution results is $\tilde{O}(n^2/X \cdot R^2)$ (see the full analysis in the proof of Lemma 6). This is also the preprocessing time for constructing these quad trees as the convolution of two n -length vectors can be calculated in $\tilde{O}(n)$ time. It is clear that the space and preprocessing time are truly subquadratic in n for any $\delta > 2\gamma > 0$. Moreover, the query time overhead is no more than $O(\log n)$ for every query (a search through a path from the root to a leaf in some quad tree). ◀

► **Theorem 18.** *Any solution to SetIntersection with sets $S_1, S_2, \dots, S_m \subseteq [u]$ for any value of $u \in [N^\delta, N]$, such that $N = \sum_{i=1}^m |S_i|$ and $\delta > 0$, must either have $\Omega(m^{2-o(1)})$ preprocessing time or have $\tilde{\Omega}(u^{1-o(1)} + out)$ query time, unless the 3SUM Conjecture is false.*

Proof. Given an instance of the 3SUM problem that contains 3 arrays A, B and C with n numbers in each of them, we can solve this instance simply by creating a 3SUM indexing instance with arrays A and B and n queries - one for each number in C . Thus, using the previous lemma the given 3SUM instance can be reduced to an instance of SetIntersection with $m = n^{1+\gamma-\delta/2}$ sets from universe $[u]$ using $O(n^{2+2\gamma-\delta})$ time for preprocessing, where the total number of queries to these instances is $O(n^{2+\gamma-\delta})$.

We assume to the contradiction that there is an algorithm that solves SetIntersection on m sets from a universe $[u]$ with $O(m^{2-\epsilon_1})$ preprocessing time for some $\epsilon_1 > 0$ and $O(u^{1-\epsilon_2} + out)$ query time for some $0 < \epsilon_2 \leq 1$. If we choose the value of δ such that $\delta > \max\{2, \frac{1}{\epsilon_2}\}$, then we have a solution to 3SUM with truly subquadratic running time. This contradicts the 3SUM Conjecture. ◀

B Hardness of Reporting Problems

B.1 Range Mode Reporting

In the reporting variant of the Range Mode problem we are required to report all elements in the query range that are the mode of this range. We have stronger lower bounds for this variant using the same construction as in the proof of Theorem 8 with the conditional lower bounds for SetIntersection with bounded universe. The results refer to both the interplay between space and query time and the interplay between preprocessing and query time.

► **Theorem 19.** *Any data structure that answers Range Mode Reporting in $O(T + out)$ time on a string of length n , where out is the output size, must use $S = \tilde{\Omega}(n^2/T^2)$ space, unless the Strong SetIntersection Conjecture is false.*

► **Theorem 20.** *Any data structure that answers Range Mode Reporting in $O(T + out)$ time on a string of length n , where out is the output size, must have $P = \tilde{\Omega}(n^2/T^2)$ preprocessing time, unless the 3SUM Conjecture is false.*

B.2 3SUM-Indexing Reporting

In the reporting variant of 3SUM-Indexing we are required to report all pairs of numbers $a \in A$ and $b \in B$ such that their sum equals the query number. Using our hardness results for SetIntersection with bounded universe we prove the following conditional lower bounds on 3SUM-Indexing reporting. These results are obtained by applying the same techniques as in the proof of Theorem 13.

7:22 On the Hardness of Set Disjointness and Set Intersection with Bounded Universe

► **Theorem 21.** *For any $\epsilon > 0$ and $0 < \delta \leq 1$, any solution to 3SUM-Indexing reporting with arrays $A = a_1, a_2, \dots, a_n$ and $B = b_1, b_2, \dots, b_n$ such that for every $i \in [n]$ $a_i, b_i \in [n^{2+\epsilon-\delta}]$ must either use $\Omega(n^{2-\delta-o(1)})$ space or have $\tilde{\Omega}(n^{\delta-o(1)} + \text{out})$ query time, where out is the output size, unless Strong SetIntersection Conjecture is false.*

► **Theorem 22.** *For any $\epsilon > 0$ and $0 < \delta \leq 1$, any solution to 3SUM-Indexing reporting with arrays $A = a_1, a_2, \dots, a_n$ and $B = b_1, b_2, \dots, b_n$ such that for every $i \in [n]$ $a_i, b_i \in [n^{2+\epsilon-\delta}]$ must either have $\Omega(n^{2-\delta-o(1)})$ preprocessing time or have $\tilde{\Omega}(n^{\delta-o(1)} + \text{out})$ query time, where out is the output size, unless the 3SUM Conjecture is false.*

Gathering and Election by Mobile Robots in a Continuous Cycle

Paola Flocchini

School of Electrical Eng. and Comp. Sci., University of Ottawa, Ottawa, ON, K1N 6N5, Canada

Ryan Killick

School of Computer Science, Carleton University, Ottawa, ON, K1S 5B6, Canada

Evangelos Kranakis

School of Computer Science, Carleton University, Ottawa, ON, K1S 5B6, Canada

Nicola Santoro

School of Computer Science, Carleton University, Ottawa, ON, K1S 5B6, Canada

Masafumi Yamashita

Dept. of Comp. Sci. and Comm. Eng., Kyushu University, Motooka, Fukuoka, 819-0395, Japan

Abstract

Consider a set of n mobile computational entities, called *robots*, located and operating on a continuous cycle \mathcal{C} (e.g., the perimeter of a closed region of \mathcal{R}^2) of arbitrary length ℓ . The robots are identical, can only see their current location, have no location awareness, and cannot communicate at a distance. In this weak setting, we study the classical problems of *gathering* (GATHER), requiring all robots to meet at a same location; and *election* (ELECT), requiring all robots to agree on a single one as the “leader”. We investigate how to solve the problems depending on the amount of knowledge (exact, upper bound, none) the robots have about their number n and about the length of the cycle ℓ . Cost of the algorithms is analyzed with respect to *time* and number of *random bits*. We establish a variety of new results specific to the continuous cycle – a geometric domain never explored before for GATHER and ELECT in a mobile robot setting; compare Monte Carlo and Las Vegas algorithms; and obtain several optimal bounds.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Cycle, Election, Gathering, Las Vegas, Monte Carlo, Randomized Algorithm

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.8

Funding Paola Flocchini, Evangelos Kranakis, Nicola Santoro: Research supported in part by NSERC Discovery grant.

Paola Flocchini: University Research Chair.

Ryan Killick: Research supported by the NSERC Canada Graduate Scholarship.

1 Introduction

1.1 The Framework

Consider a distributed system composed of a set \mathbf{R} of autonomous mobile computational entities, called *robots*, located and operating in an Euclidean space \mathcal{U} . The robots are identical: without identifiers or distinguishing features, they have the same capabilities and execute the same algorithm. Although autonomous, their goal is to collectively perform some assigned system task or to solve a given problem. Among the important tasks and problems are: *gathering* (GATHER), requiring all robots to meet at a same location; and *election* (ELECT), requiring all robots to agree on a single one as the “leader”. Indeed, GATHER is one of the fundamental problems in theoretical mobile robotics, while ELECT is typically solved as an intermediate step in the resolution of many important problems, in particular



© Paola Flocchini, Ryan Killick, Evangelos Kranakis, Nicola Santoro, and Masafumi Yamashita; licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 8; pp. 8:1–8:19

Leibniz International Proceedings in Informatics



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

pattern formations. Both GATHER and ELECT have been extensively investigated under a variety of assumptions on the capabilities of the robots (e.g., memory, communication, visibility, orientation, speed), on the space in which they operate, and on the power of the adversary. From the point of view of the behaviour of the robots, the two main models are *Look-Compute-Move (LCM)* and *Continuous Time (CT)*. In *LCM* the robots operate by cycling through three separate processes: observing the space (*Look*), executing the algorithm to determine a destination (*Compute*), and moving towards it (*Move*). In *CT* the robots are permanently active and continuously performing all three processes. For a recent overview see [15] and the chapters therein.

In all investigations, in both models, the theoretical concern is to identify the weakest possible conditions that make the problems solvable.

In this paper, we consider GATHER and ELECT by identical robots when the space \mathcal{U} is a continuous *cycle* \mathcal{C} (e.g., the perimeter of a closed region of \mathcal{R}^2). This spatial setting has been investigated in the *LCM* model with respect to the *scattering* problem, requiring identical robots to place themselves at uniform distance along the cycle [13]. In the *CT* model, a continuous *cycle* has been studied in the context of solving *patrolling* when the robots are identical [9] and when they have different motorial capabilities [7]; *gathering* has also been investigated, but only with robots having different motorial capabilities [22].

We study GATHER and ELECT in the *CT* model in a very weak computational setting: the identical robots can only see their current location and have no location awareness; furthermore they cannot communicate at a distance (i.e., communication is possible only between robots located at the same point at the same time).

It is immediate to observe that, in our setting, both problems are deterministically unsolvable: there is no deterministic algorithm that, in all possible executions of the algorithm by the robots and regardless of the initial position of the robots in the cycle, will always correctly solve the problem within finite time. This is obvious in the case of ELECT because, to render a single robot uniquely different from all others it requires the existence of some *asymmetry* in the system (e.g., in the initial placement of the robots, in shape of the Euclidean space) if no difference is present among the robots (e.g., distinct ids, different speeds). In our setting the impossibility holds also for GATHER, which does not have such a stringent requirement, and can sometimes be deterministically solved in absence of asymmetries and differences among the robots (e.g. [5]). Further observe that, since visibility is limited to the current robot's location, in our setting both problems are deterministically unsolvable even if the initial configuration is asymmetric, and the robots are aware of this fact. Summarizing, the only possible solution algorithms are randomized ones.

1.2 Main Contributions

In this paper we start the investigation of solving GATHER and ELECT by the set of robots \mathbf{R} deployed in a continuous cycle \mathcal{C} . Since GATHER is of easy resolution once a leader has been elected, we primarily focus on ELECT.

We propose both Las Vegas and Monte Carlo decentralized election protocols where: a Las Vegas algorithm correctly terminates with probability one in an unpredictable amount of time; a Monte Carlo algorithm has a fixed termination time but pays for this determinism with a positive – yet bounded – probability that it has terminated incorrectly. In other words, a Las Vegas algorithm “gambles with resources” and a Monte Carlo algorithm “gambles with correctness”.

We evaluate the complexity of the proposed algorithms with respect to two cost measures: the *time* until the algorithm terminates, and the total number of *random bits* (coin flips) required by the algorithm. The costs depend not only on the length ℓ of the cycle and the number n of mobile robots (note that n can be arbitrarily larger than ℓ), but also and more importantly on the *knowledge* (none, exact, upper bound) the robots have on ℓ and/or n .

We establish several results. In particular, we prove that, with knowledge of ℓ , a leader can be elected with probability one in *optimal time* with an *optimal number of random bits*, even without any knowledge of (an upper bound on) n . If only an upper bound $L = O(\ell)$ is known, then a leader can be elected with high probability in *optimal time* with an *optimal number of random bits*, even without any knowledge of (an upper bound on) n .

The results of the paper are summarized in Tables 1 and 2. As we are analyzing randomized algorithms, the cost measures are often random variables; when this is the case, we give both the value achieved in the average and that with high probability.

■ **Table 1** Results according to the knowledge of the robots (“Ex.” = exact, “-” = no knowledge, “UB” = upper bound). T_{exp} (resp. B_{exp}) represents the expected time (resp. random-bit) complexity. The column “Type” gives the type of randomized algorithm (LV = Las Vegas, MC = Monte Carlo). The last column gives the corresponding algorithm label in the text. When an upper bound on ℓ (resp. n) is known it is represented by L (resp. N); and the constructed upper bound on n is $\hat{N} = \frac{Ln}{\ell}$.

n	ℓ	T_{exp}	B_{exp}	Type	Algo.
Ex.	UB	$O(L)$	$O(n)$	LV	A1
Ex.	-	$O(n + \ell)$	$O(n + n \log \lceil \ell/n \rceil)$	LV	A1 + A7
-	Ex.	$O(\ell)$	$O(n)$	LV	A1 + A6
UB	UB	$O(L)$	$O(n)$	MC	A3
UB	-	$O(N + N \cdot \ell/n)$	$O(n + n \log \lceil \ell/n \rceil)$	MC	A3 + A7
-	UB	$O(L)$	$O(n)$	MC	A3+A8

■ **Table 2** Same as Table 1 for time and bit complexities with high probability.

n	ℓ	T_{whp}	B_{whp}	Type	Algo.
Ex.	UB	$O(L \log n)$	$O(n \log n)$	LV	A1
Ex.	-	$O(n + \ell \log n)$	$O(n \log n + n \log \lceil \ell/n \rceil)$	LV	A1 + A7
-	Ex.	$O(\ell \log n)$	$O(n \log n)$	LV	A1 + A6
UB	UB	$O(L \log N)$	$O(n \log N)$	MC	A3
UB	-	$O(N + N \cdot \ell/n \cdot \log N)$	$O(n \log n + n \log \lceil \ell/n \rceil)$	MC	A3 + A7
-	UB	$O(L \log \hat{N})$	$O(n \log \hat{N})$	MC	A3+A8

The paper is organized as follows. We first consider the case when the robots have some level of knowledge (exact or upper bound) of both parameters (Section 3). We prove that, when the robots possess knowledge of n , the knowledge of an upper bound $L = O(\ell)$ allows for a LV solution which is *optimal* with respect to both complexity measures. In case the robots know only upper bounds on both n and ℓ , we give a Monte Carlo algorithm. In Section 4 we consider the cases when the robots have no knowledge (exact nor upper bound) of one of the two parameters. In these cases we provide Las Vegas algorithms by which the robots can obtain knowledge of the unknown parameter efficiently, and subsequently elect a leader using the algorithms of Section 3. In Section 5 we demonstrate that unless the robots know n and/or ℓ exactly, a Las Vegas algorithm cannot exist that solves ELECT. Extensions, including the solutions for GATHER using the results for ELECT, and open questions are discussed in Section 6.

1.3 Related work

There exists an extensive literature on problem solving by n *identical* mobile robots in *continuous* spaces, both within the distributed computing and the control communities; e.g., see the books [4, 14, 15]. In distributed computing, the problem of gathering identical robots has been the focus of intensive investigations under a variety of assumptions on the computational power and communication capabilities of the robots (e.g., [5, 6, 16, 27]). Similarly, the problem of electing a leader and its relationship to asymmetry has been observed, investigated and discussed when studying solvability of a variety of problems by autonomous mobile robots, in particular *pattern formations* (e.g., [10, 17, 19]). Indeed, a great deal of research has been devoted to the link between degree of symmetries and deterministic problem solving; see [15] and chapters therein for a recent account, in particular [30]. Almost all of this work is on deterministic solutions, with few exceptions (e.g., [20]).

Robots operating specifically in a *continuous cycle* have been studied in the context of rendezvous and gathering, but only with robots having different motorial capabilities [11, 22]. Other investigated problems in a continuous cycle are: *patrolling*, studied both when the robots are identical and when they have different motorial capabilities (e.g. see [7, 8, 9]); and *scattering*, where the robots must place themselves at uniform distance on the cycle [13].

The geometric continuous settings in which the mobile entities can move freely are in general more suitable than discrete settings for distributed computing applications in robotics [4]. This is further enforced by the fact that after a system shut-down in a robot application the participating robots cannot be guaranteed to occupy the vertices of a graph but rather might be placed at arbitrary locations in the underlying geometric domain.

Settings of identical *mobile entities* operating in *discrete spaces* (i.e., in graphs) are extremely important as they naturally describe a wide variety of computational environments, including networked systems supporting mobile software agents, and ad-hoc wireless networks. In these settings, the analogue of a set of mobile robots in a continuous cycle is a set of identical *mobile agents* in a *ring* of identical nodes. Interestingly, this discrete setting has been extensively studied, especially for rendezvous and gathering; e.g., see the monograph [26]. In absence of distinct features of the agents and of the nodes (e.g., ids, markers, tokens), solutions are necessarily randomized, and their development has been the object of several investigations. In particular Ooshita et al. studied the gathering problem in anonymous unidirectional ring networks for multiple (mobile) agents with limited knowledge and characterized the relation between probabilistic solvability and termination detection [29]. Izumi et al. investigated the feasibility of polynomial-expected-round randomized gathering for n robots and show that any randomized algorithm has $\Omega(\exp(n))$ expected-round lower bound [24].

In the computational universe of *static* (or *stationary*) *entities* connected via a communication network (i.e. the traditional message-passing universe in distributed computing), the computational entities coincide with the network nodes (i.e., the nodes are the active agents). Note that, in this universe, the problem GATHER does not exist; on the other hand, ELECT is a fundamental problem. When the entities are identical, the system is known as an *anonymous network*, and several researchers have focused on computing in an *anonymous ring* (e.g., [1, 2, 12]). The problem of electing a leader in an anonymous network, known also as *symmetry breaking* and for which clearly only probabilistic solutions exist, has been investigated in an anonymous ring network (e.g., [3, 18, 23]). In particular, Itai and Rodeh proposed probabilistic algorithms for both the synchronous and asynchronous case; they considered both cases when the size of the ring may be either known or unknown to the nodes and studied its impact on termination with a nonzero probability [23].

Interestingly, of all the related work, the one closest in spirit to our investigation is that of symmetry breaking in an anonymous ring, in spite of the fact that the computational universes are completely different: static entities and discrete space in one while mobile entities and continuous space in ours.

2 Model

Let \mathbf{R} be a set of $n \geq 2$ autonomous mobile computational entities, called robots, located in a continuous *cycle* \mathcal{C} (e.g., the perimeter of a closed region of \mathcal{R}^2) of real length ℓ in arbitrary and pairwise distinct positions.

The robots are identical: without identifiers or distinguishing features, they have the same (computational, motorial and communication) capabilities and execute the same algorithm. We assume that all robots move at speed one. Each robot $r \in \mathbf{R}$ has a local memory composed of a finite set of registers, including a special register $state(r)$ which stores the current state of r ; initially, the content of the memory of every robot is the same. Each robot is in possession of a fair coin which outputs H or T each with probability $1/2$. At any time a robot may flip its coin and base a decision on the outcome of that flip. For a robot r we will use the notation $b(r)$ to represent a special register which always contains the outcome of its most recent coin-flip. We will use the notation $b(r) \leftarrow flip()$ to represent the action of flipping a coin and assigning the outcome to $b(r)$.

The robots can only see their current location and have no location awareness. Furthermore they cannot communicate at a distance; that is, communication is possible only between robots located at the same point at the same time (face-to-face). A robot may move along \mathcal{C} in either the CW (clockwise) or CCW (counter-clockwise) direction and may stop and/or reverse its direction of movement at any time. For simplicity, we will assume that the robots have consistent orientations and argue in Section 6 why this assumption is not necessary.

The robots are permanently active and continuously performing three processes: executing the algorithm (which might require flipping a coin), moving in a given direction or not at all (if so prescribed by the algorithm), and communicating with co-located robots. A robot can distinguish among its co-located robots and is able to instantaneously exchange any amount of information with each of them. When two robots moving in opposite directions meet, or a moving robot meets a stopped robot, the two robots become co-located; we call this an *encounter*. During an encounter, one of the robots can decide to *merge* with the other, thereby committing itself to following all actions of the robot it has merged with. As a result of this process, robots will form robot *stacks* with the head of the stack the only robot actively participating in an algorithm (the stack acts as a single robot). A robot r will keep track of the number of robots present in its stack in a special register denoted by $CNR(r)$.

We assume a fully synchronous system in the following sense. Each robot possesses an identical copy of the same clock and each robot can use their respective clocks to measure arbitrarily small intervals with respect to the same unit of time (which we may take to be 1 without loss of generality). All robots will begin an algorithm at the same moment and all robots move with the same speed (which we may also take to be 1 without loss of generality). This implies that robots can fix a unit length as the distance traveled in one unit of time.

We study how such robots can solve ELECT and GATHER, and at what cost. The *election* problem, ELECT, requires the robots to transition from an initial configuration where each robot is in an identical state, to one where a single robot can be uniquely distinguished from the others. When solving this problem, we will assume the robots can be found in one of the three states CANDIDATE, FOLLOWER, or LEADER. The *gathering* problem,

GATHER, requires the robots to transition from an initial configuration where each robot is in an identical state, to one where all robots are co-located and will no longer move. Since GATHER is of easy resolution once a leader has been elected, we primarily focus on ELECT.

We distinguish between two types of randomized algorithms: those of the *Las Vegas* type and those of the *Monte Carlo* type [28, 21]. An algorithm is of the Las Vegas type, if, for any problem instance, it is correct when it terminates and it terminates with probability 1. In contrast, an algorithm is of the Monte Carlo type if, for any problem instance, it always terminates and it is correct with a probability p which is bounded away from zero.

The costs of a solution algorithm are evaluated with respect to two measures: 1) *time complexity* – the time until the algorithm terminates; and 2) *random-bit complexity* – the total number of random bits/coin flips used by the algorithm. The costs depend not only on the system parameters, the length ℓ of the cycle and the number n of mobile robots, but also and more importantly on the type of knowledge available to the robots about the values of those parameters. As we are analyzing randomized algorithms, these complexity measures will often be random variables. When this is the case, we will give the value achieved in the average and with high probability.

3 Election with knowledge of both n and ℓ

In this section we consider ELECT when the robots possess knowledge of both n and ℓ (either exact or upper bounds). We begin with the case that the robots have exact knowledge. Pseudocode for all algorithms can be found in the appendix.

3.1 Exact knowledge of n and ℓ

► **Theorem 1.** *Let n and ℓ be known to the robots. There is a Las Vegas algorithm solving ELECT which terminates in time $O(\ell)$ on average and in time $O(\ell \log n)$ with high probability; and requires $O(n)$ random bits on average and $O(n \log n)$ with high probability.*

The proof is based on the algorithm ELECTLV(n, ℓ). This algorithm is formally described as Algorithm 1 and takes as inputs the number of robots n and the length of the cycle ℓ . Initially all robots begin in the same CANDIDATE state and each robot r has $\text{CNR}(r)$ set to 1. The algorithm proceeds in a series of rounds beginning with the round $t = 0$. In each round the CANDIDATE robots will run the procedure ELECTIONROUND(D) with input $D_t = \min\{\frac{\ell}{2}, \frac{\ell}{n}(4/3)^t\}$, the result of which is that a subset of the robots merge and enter the FOLLOWER state. This will continue on until only a single CANDIDATE robot remains with a stack containing all n robots. As the robots know the value of n , this last remaining robot will know it is the last and will thus enter the LEADER state.

The procedure ELECTIONROUND(D) is formally described as Algorithm 2. The idea of this procedure is as follows. Each robot begins by flipping a coin. Those that flip T will remain stationary for a time $4D_t$. Those that flip H will: move CCW a distance D_t ; return to their initial positions; move CW a distance D_t ; and again return to their initial positions. If ever it occurs that a robot r who flipped H encounters a robot s who flipped T then s will merge with r and r will update the value of $\text{CNR}(r)$ to reflect this.

We begin our analysis by determining how effective the procedure ELECTIONROUND(D) is at reducing the number of candidates. This will be the subject of the next two lemmas.

► **Lemma 2.** *Let n and n' respectively represent the number of CANDIDATE robots before and after ELECTIONROUND(D) is run with input $D > 0$. Then $E[n'] \leq \frac{n}{2} + \frac{1}{2} \lceil \frac{\ell}{2D} \rceil$.*

Proof. Partition the cycle into $m = \lceil \frac{\ell}{2D} \rceil$ disjoint intervals such that each interval has length $\frac{\ell}{m} \leq 2D$. For each $i \in [1, m]$ let n_i and n'_i respectively represent the number of CANDIDATE robots contained in the i^{th} interval at the beginning and end of $\text{ELECTIONROUND}(D)$. Then it is clear that $n = \sum_{i=1}^m n_i$ and $n' = \sum_{i=1}^m n'_i$. This latter expression allows us to write the expectation of n' as follows:

$$E[n'] = \sum_{i=1}^m E[n'_i] = \sum_{i=1}^m \sum_{x=1}^{n_i} x \Pr[n'_i = x]. \quad (1)$$

To determine the probability $\Pr[n'_i = x]$ consider the i^{th} interval which initially contains $n_i > 0$ CANDIDATE robots. If at least one of these n_i robots flipped H then the number of them that will remain CANDIDATE is exactly the number of them that flipped H. Thus, if we let k_i represent the random variable which counts the number of CANDIDATE robots that flipped H in an interval i then we can conclude that $\Pr[n'_i = x | k_i \geq 1] = 1$ if $x = k_i$ and 0 otherwise. For $x \in [1, n_i]$ this implies that $\Pr[n'_i = x] = \sum_{j=0}^{n_i} \Pr[n'_i = x | k_i = j] \Pr[k_i = j]$ or $\Pr[n'_i = x] = \Pr[k_i = x] + \Pr[n'_i = x | k_i = 0] \Pr[k_i = 0]$. Using this expression for $\Pr[n'_i = x]$ we find that $E[n'_i] = \sum_{x=0}^{n_i} x \Pr[k_i = x] + \sum_{x=0}^{n_i} x \Pr[n'_i = x | k_i = 0] \Pr[k_i = 0]$.

It is not hard to see that k_i is binomially distributed with parameters n_i and $p = 1/2$ implying that $\sum_{x=0}^{n_i} x \Pr[k_i = x] = n_i/2$, and that $\Pr[k_i = 0] = (1/2)^{n_i}$. The sum $\sum_{x=0}^{n_i} x \Pr[n'_i = x | k_i = 0] \Pr[k_i = 0]$ represents the expected number of CANDIDATE robots surviving in an interval i given that they all flipped T. Clearly this expectation is bounded by n_i and we can thus conclude that $E[n'_i] \leq \frac{n_i}{2} + n_i \left(\frac{1}{2}\right)^{n_i} \leq \frac{n_i}{2} + \frac{1}{2}$.

To bound the expectation of n' we can substitute this inequality into (1) to get $E[n'] = \sum_{i=1}^m E[n'_i] \leq \sum_{i=1}^m \left(\frac{n_i}{2} + \frac{1}{2}\right) = \frac{n}{2} + \frac{m}{2}$ where we have used the fact that $n = \sum_{i=1}^m n_i$ in the last step. Since $m = \lceil \frac{\ell}{2D} \rceil$ the lemma follows. ◀

► **Lemma 3.** *Let n_t count the number of CANDIDATE robots remaining in round $t \geq 0$ of $\text{ELECTLV}(n, \ell)$. Then $E[n_t] \leq \left\lceil \left(\frac{3}{4}\right)^t n \right\rceil$.*

Proof. The proof is by induction on t . The base case $t = 0$ is clearly true. We assume that the claim holds up to $t = k$. Using the induction hypothesis and Lemma 2 we can write $E[n_{k+1}] \leq \frac{1}{2} \left\lceil \left(\frac{3}{4}\right)^k n \right\rceil + \frac{1}{2} \left\lceil \frac{\ell}{2D_k} \right\rceil$ where $D_t = \min \left\{ \frac{\ell}{2}, \frac{\ell}{n} \left(\frac{4}{3}\right)^t \right\}$. The lemma clearly holds if $D_k \geq \frac{\ell}{2}$. If this is not the case then $D_k = \frac{\ell}{n} \left(\frac{4}{3}\right)^k$ and again it is easy to see that the lemma holds. ◀

In the next three lemmas (Lemma 4, Lemma 5, and Lemma 6) we bound the number of rounds, time, and random-bits required until only a single candidate robot remains. In order to do so we will employ a useful theorem by Karp [25] concerning the solutions of stochastic recurrence relations. This theorem is described in the appendix as Theorem 22.

► **Lemma 4.** *Let T be the first round of $\text{ELECTLV}(n, \ell)$ in which only a single CANDIDATE robot remains. Then $E[T] \leq \left\lceil \log_{4/3}(n) \right\rceil + 1$ and, for any positive integer w , $\Pr \left[T \geq \left\lceil \log_{4/3}(n) \right\rceil + 1 + w \right] \leq \left(\frac{3}{4}\right)^w \frac{n}{(4/3)^{\left\lceil \log_{4/3}(n) \right\rceil}}$.*

Proof. Observe that $T = T(n)$ satisfies the stochastic recurrence relation $T(n) = 1 + T(h(n))$ with base condition $T(1) = 0$ and where the expectation of $h(n)$ is bounded using Lemma 3, i.e., $E[h(n)] \leq \left\lceil \frac{3}{4}n \right\rceil$. With this observation the lemma follows easily from Theorem 22. ◀

► **Lemma 5.** *Let τ be the time required until only a single CANDIDATE robot remains in ELECTLV(n, ℓ). Then $E[\tau] \leq 8L$ and, for any positive integer w , $\Pr[T \geq 2L(4+w)] \leq \left(\frac{3}{4}\right)^w \frac{n}{(4/3)^{\lfloor \log_{4/3}(n) \rfloor}}$.*

Proof. Set t_L as the first round which satisfies $L/n(4/3)^t \geq L/2$, i.e. $t_L = \lceil \log_{4/3}(n/2) \rceil$. Assume that it takes $T > t_L$ rounds until only one CANDIDATE robot remains. The time τ required to complete these T rounds is $\tau = 4\frac{L}{n} \sum_{t=0}^{t_L-1} (4/3)^t + 2 \sum_{t=t_L}^T L \leq 12\frac{L}{n}(4/3)^{t_L} + 2(T - t_L)L \leq 8L + 2(T - t_L)L$. The lemma now follows from Lemma 4. ◀

► **Lemma 6.** *Let B be the random variable which counts the number of coin-flips used in ELECTLV(n, ℓ). Then $E[B] \leq 4n$ and, for any positive integer w , $\Pr[B \geq (4+w)n] \leq \left(\frac{3}{4}\right)^w$.*

Proof. Similarly to the proof of Lemma 4 we observe $B = B(n)$ satisfies the stochastic recurrence relation $B(n) = n + B(h(n))$ with base condition $B(1) = 0$ and where $h(n)$ has expectation $E[h(n)] \leq \lceil \frac{3}{4}n \rceil$. With this observation the lemma follows easily from Theorem 22. ◀

The proof of Theorem 1 now follows immediately from Lemmas 5, and 6.

3.2 Inexact knowledge of n and/or ℓ

We now consider the cases that the robots are provided with inexact knowledge (upper bounds) of at least one of n or ℓ . We begin with the case that the robots know n and an upper bound on ℓ .

Observe that nowhere in the proof of Theorem 1 did we require the robots to know exactly the value of ℓ . In particular, if the robots were to instead use an upper bound L on ℓ then the only change we need to make is to replace ℓ with L in the time complexity. This observation thus easily leads to the following corollary of Theorem 1:

► **Corollary 7.** *Let n and an upper bound $L \geq \ell$ be known to the robots. There is a Las Vegas algorithm solving this problem which terminates in time $O(L)$ on average and in time $O(L \log n)$ with high probability; and requires $O(n)$ random bits on average and $O(n \log n)$ with high probability.*

The same argument does not work if the robots know ℓ and an upper bound $N \geq n$ since ELECTLV requires the exact value of n in order to terminate. We will see in the next section that exact knowledge of ℓ however allows the robots to determine n and we will therefore postpone a discussion of this case until then.

If the robots only possess upper bounds on both n and ℓ then a Las Vegas algorithm does not exist (see Section 5). We thus provide a Monte Carlo algorithm (Algorithm 3) to solve the problem.

► **Theorem 8.** *Let upper bounds $N \geq n$ and $L \geq \ell$ be known to the robots. Then, for any positive integer w there is a Monte Carlo algorithm solving ELECT with error probability $O((3/4)^w)$. This algorithm terminates in time $O(wL)$ and requires $O(wN)$ random bits.*

Proof. The proof is based on the algorithm ELECTMC(N, L, w) which takes as inputs the upper bounds N and L , and a positive integer w which controls the runtime. This algorithm is formally described as Algorithm 3. This algorithm is identical to ELECTLV(N, L) except that it deterministically terminates on the round $t_\infty = \lceil \log_{4/3}(N) \rceil + w$. We may therefore reuse many of our previously derived results. In particular, the time τ until termination

follows from the proof of Lemma 5 and is given by $\tau = 8L + 2(w + 1)L$. The random-bit complexity follows from Lemma 6. The error probability of the algorithm is also easy to derive. In particular, if we let T be the number of rounds required until only a single CANDIDATE remains then the probability that the algorithm terminates incorrectly is simply the probability $\Pr[T > t_\infty] = \Pr\left[T > \left\lceil \log_{4/3}(N) \right\rceil + w\right] = \Pr\left[T \geq \left\lceil \log_{4/3}(N) \right\rceil + 1 + w\right]$ and this probability is given by Lemma 4. ◀

4 Election with knowledge of either n or ℓ

In this section we investigate ELECT when the robots are provided with knowledge of only one of n or ℓ (exact or upper bounds). In all cases we use the same strategy to solve the problem: we develop algorithms by which the robots gain knowledge of the unknown of n or ℓ and then use the algorithms of the previous section to solve ELECT. Pseudocode for all algorithms presented can be found in the appendix.

4.1 Exact knowledge of n or ℓ

► **Theorem 9.** *Let either n or ℓ be known to the robots. Then there are Las Vegas algorithms solving ELECT. If ℓ is known the algorithm terminates in time $O(\ell)$ on average and in time $O(\ell \log n)$ with high probability; and requires $O(n)$ random bits on average and $O(n \log n)$ with high probability. If n is known the algorithm terminates in time $O(n + \ell)$ on average and in time $O(n + \ell \log n)$ with high probability; and requires $O(n + n \log \lceil \frac{\ell}{n} \rceil)$ random bits on average and $O(n \log(n) + n \log \lceil \frac{\ell}{n} \rceil)$ with high probability.*

As previously stated, our proof strategy is to first develop algorithms by which the robots can gain knowledge of the unknown of n or ℓ . More specifically, the goal of this section is to constructively demonstrate the validity of the following two lemmas from which Theorem 9 will easily follow.

► **Lemma 10.** *Consider n robots on a cycle of length ℓ and assume the robots know only the value of ℓ . Then there exists a Las Vegas algorithm by which the robots can determine the value of n . This algorithm terminates in time $O(\ell)$ on average and with high probability; and requires $O(n)$ random bits on average and with high probability.*

► **Lemma 11.** *Consider n robots on a cycle of length ℓ and assume the robots know only the value of n . Then there exists a Las Vegas algorithm by which the robots can determine an $O(\ell)$ upper bound L on ℓ . This algorithm terminates in time $O(n + \ell)$ on average and with high probability; and requires $O(n + n \log \lceil \frac{\ell}{n} \rceil)$ random bits on average and with high probability.*

We will begin by introducing two procedures which will be used throughout the remainder of the section. The first procedure will be used by the robots to count coin flips, and the second is a minimum finding procedure.

A procedure to count coin flips. The procedure COUNTFLIPS(D) is formally described as Algorithm 4 and takes as input a distance D . For simplicity in the following description we will assume that $D = \ell$. The procedure presumes that each robot r has flipped a coin and stored the result in $b(r)$. It will result in each robot either knowing the total number of robots or that all robots have flipped the same thing.

At the beginning the robots that flip H will move CW a distance ℓ around the cycle and count each robot they encounter which flipped T. The robots that flipped T will likewise wait for a time ℓ and count each robot they encounter that flipped H. Since each moving

8:10 Gathering and Election by Mobile Robots in a Continuous Cycle

robot makes a full traversal of the cycle they are guaranteed to see all stationary robots. Thus, after the first ℓ time units, each robot will determine the number of robots which flipped opposite to themselves. In the last ℓ time units of the algorithm the robots which initially flipped H (resp. T) will move CCW a distance ℓ around the cycle (resp. wait for ℓ time units). In either case, a robot will determine the total number of robots that flipped the same as themselves from the first robot they encounter which flipped opposite to themselves. Thus, after 2ℓ time units each robot will have determined both the total number of robots which flipped H and the number that flipped T and from this they can compute n . If all robots flipped the same thing then the robots will know this since each will have determined that $N_H(r) = N_T(r) = 0$. From this description it is easy to establish the following lemma:

► **Lemma 12.** *Assume that all robots have flipped a coin. Then in exactly 2ℓ time units the procedure COUNTFLIPS(ℓ) will result in either each robot knowing n or that all robots have flipped the same thing.*

When an input $D > \ell$ is used in the procedure we claim the following:

► **Lemma 13.** *Assume that all robots have flipped a coin and that $D \geq \ell$. Then in exactly $2D$ time units the procedure COUNTFLIPS(D) will result in either each robot r computing an upper bound $N(r) \geq n$ or that all robots have flipped the same thing.*

Proof. Clearly, if all robots flip the same then each robot will compute $N_H(r) + N_T(r) = 0$. Thus, assume that at least two robots flip differently. Let n_T and n_H represent the actual number of robots that flipped T and H respectively, i.e. $n_T + n_H = n$. Since each robot that flipped H traverses the cycle at least once each such robot is guaranteed to encounter all robots that flipped T. Likewise, each robot that flipped T is guaranteed to encounter each robot that flipped H. It is therefore not possible for a robot r to compute a value of $N_H(r) < n_H$ or $N_T(r) < n_T$ and thus it is ensured that $N_T(r) + N_H(r) \geq n$ for all robots. ◀

Finally, if an input $D < \ell$ is used in the procedure then we claim the following:

► **Lemma 14.** *Assume that all robots have flipped a coin and that $D < \ell$. Then in exactly $2D$ time units the procedure COUNTFLIPS(D) will result in each robot r computing a lower-bound $N(r) \leq n$.*

Proof. The only thing we need to demonstrate is that all robots will compute a value $N(r) \leq n$. Clearly, in order for this not to be true, at least one of the robots must double count another robot. This, however, is not possible unless a robot traverses the cycle more than once and this will clearly not be the case if $D < \ell$. ◀

A minimum finding procedure. The minimum finding procedure FINDMIN(L, N_0) is formally described as Algorithm 5 and takes as input an upper bound $L \geq \ell$ on the cycle length, and a value N_0 (which is specific to each robot). The algorithm results in each robot computing the minimum of the inputs N_0 . It assumes that all robots have flipped a coin and that at least two robots have flipped differently.

Each robot that flipped H will initially move CW a distance $L \geq \ell$ around the cycle and is guaranteed to encounter every robot that flipped T. Likewise every robot that flipped T will encounter every robot that flipped H. Thus, after the first L time units, every robot that flipped H (resp. T) will know the minimum value of every robot that flipped T (resp. H). In the second L time units the robots that flipped H will move CCW a distance L and will again encounter every robot that had flipped T. They can thus determine the minimum value of

all robots that flipped H from the first robot they encounter that flipped T. Likewise, each robot that flipped T will determine the minimum value of all robots that flipped T from the first robot they encounter that flipped H. The algorithm clearly terminates after $2L$ time units. We can thus claim the following without proof:

► **Lemma 15.** *Assume that all robots have flipped a coin, at least two have flipped differently, and that $L \geq \ell$. Then in exactly $2L$ times units the procedure $\text{FINDMIN}(L, N_0(r))$ will result in each robot r computing the minimum of all inputs $N_0(r)$.*

Computing n using ℓ . We will now tackle the proof of Lemma 10 which is based off of the algorithm $\text{COUNTROBOTS}(\ell)$. This algorithm is formally described as Algorithm 6 and takes as input the length of the cycle. The idea is to repeatedly flip coins and run the procedure $\text{COUNTFLIPS}(\ell)$ until the first round in which at least two robots flip differently. When this occurs each robot will compute the total number of robots that flipped T and the total number that flipped H and will thus determine n to be the sum of these values.

Proof. (Lemma 10) The correctness of $\text{COUNTROBOTS}(\ell)$ is obvious. The algorithm will terminate on the first round during which at least two robots flip differently. The probability that all robots flip the same is 2^{1-n} and therefore the algorithm terminates after an expected $\frac{1}{1-2^{1-n}} \leq 2$ rounds. The probability that the algorithm terminates after T rounds is $2^{(T-1)(1-n)}(1-2^{1-n})$. From this it is clear that the algorithm terminates after $O(1)$ rounds with high probability. The time and random-bit complexities follow from the fact that each round lasts time at most 2ℓ and in each round all n robots flip their coins. ◀

Computing a $O(\ell)$ upper bound on ℓ using n . The proof of Lemma 11 is based off of the algorithm $\text{BOUNDCYCLE}(n)$. This algorithm is formally described as Algorithm 7 and takes as input the number of robots on the cycle. In each round $t \geq 0$ the robots will employ the procedure COUNTFLIPS in an attempt to determine a strict upper bound on the number of robots using an estimate $L_t = n \cdot 2^t$ for an upper bound on ℓ . This will result in each robot r computing a value $N(r)$. If $L_t < \ell$ then, by Lemma 14, the robots will each compute $N(r) \leq n$ and the algorithm will proceed to the next round. If $L_t \geq \ell$ then the robots will each compute $N(r) \geq n$ and, after performing FINDMIN , they will all agree on the computed value of $N(r)$. Let t_* be the first round in which all robots compute $N(r) > n$. The corresponding value of L_t in the round t_* will then be an upper bound on ℓ . We reduce L_{t_*} by a factor $\frac{1}{2} \left\lfloor \frac{N(r)}{n} \right\rfloor$ to ensure that the returned upper bound is $O(\ell)$.

Proof. (Lemma 11) To determine the running time we let t_0 be the first round for which $L_t > 2\ell$. Then $t_0 = \lceil \log \frac{2\ell}{n} \rceil$ if $n < 2\ell$ and $t_0 = 0$ if $n \geq 2\ell$. The algorithm will certainly terminate in the first round $t_* > t_0$ in which at least two robots flip differently. Since the probability that all robots flip the same is 2^{1-n} we will have $t_* = t_0 + O(1)$ with high probability. The algorithm will therefore take at most $\lceil \log \frac{2\ell}{n} \rceil + O(1)$ rounds. Since the procedures $\text{COUNTFLIPS}(L_t)$ and $\text{FINDMIN}(L_t)$ each take time $2L_t$ to complete, each round of the algorithm lasts time $4L_t = n \cdot 2^{t+2}$. The total time required is thus $\sum_{t=0}^{t_*} n \cdot 2^{t+2} = 4n(2^{t_*+1} - 1)$. If $n > 2\ell$ then the above is clearly $O(n)$. If $n \leq 2\ell$ then we have that $4n(2^{t_*+1} - 1) = 4n \left(2^{\lceil \log \frac{2\ell}{n} \rceil + O(1)} - 1 \right) = O(\ell)$.

Thus, we can conclude that the algorithm terminates in time $O(n + \ell)$ on average and with high probability. In each round of the algorithm all robots flip a coin and thus the algorithm requires $O(n)$ random bits if $n > 2\ell$ and otherwise $O(n \log \lceil \frac{2\ell}{n} \rceil)$ when $n \leq 2\ell$. ◀

4.2 Inexact knowledge of n or ℓ

We now consider the cases that the robots are only provided with an upper bound on n or only an upper bound on ℓ . The main result follows:

► **Theorem 16.** *Let only an upper bound $L \geq \ell$ or an upper bound $N \geq n$ be known to the robots. Then, for any positive integer w there are Monte Carlo algorithms solving **ELECT** with error probability $O((3/4)^w)$. If the robots know $L \geq \ell$ then the algorithm terminates in time $O(wL)$ and requires $O(wn)$ random bits. If the robots know $N \geq n$ then the algorithm terminates in time $O(N + w\frac{N}{n}\ell)$ and requires $O(wn + n \log \lceil \frac{\ell}{n} \rceil)$ random bits.*

Our goal is again to develop algorithms by which the robots will gain knowledge of the unknown of n or ℓ and then employ the algorithm **ELECTMC** to solve **ELECT**. We therefore want to demonstrate the following two lemmas:

► **Lemma 17.** *Consider n robots on a cycle of length ℓ and assume the robots know an upper bound $L \geq \ell$. Then there exists a Las Vegas algorithm by which the robots can determine an upper bound $N = O(\frac{L}{\ell}n)$ on n . This algorithm terminates in time $O(L)$ on average and with high probability; and requires $O(n)$ random bits on average and with high probability.*

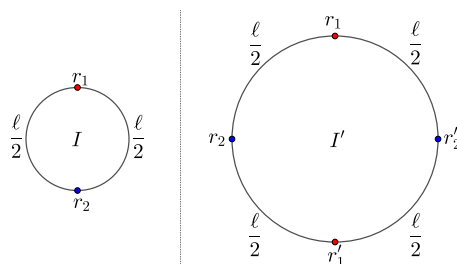
► **Lemma 18.** *Consider n robots on a cycle of length ℓ and assume the robots know only an upper bound on the value of n . Then there exists a Las Vegas algorithm by which the robots can determine an $O(\frac{N}{n}\ell)$ upper bound L on ℓ . This algorithm terminates in time $O(N + \frac{N}{n}\ell)$ on average and with high probability; and requires $O(n + n \log \lceil \frac{\ell}{n} \rceil)$ random bits on average and with high probability.*

Clearly Theorem 16 will directly follow from the above two lemmas as well as Theorem 8. We begin with the case that the robots know $L \geq \ell$.

Computing an upper bound on n from an upper bound on ℓ . Here we will use an algorithm essentially identical to **COUNTROBOTS**(ℓ) except with the addition of a **FINDMIN** procedure. The robots will repeatedly flip coins and run the procedure **COUNTFLIPS**(L) until at least two robots flip differently. At this point each robot r will know an upper bound $N(r) \geq n$. They will then run the procedure **FINDMIN**($L, N(r)$) in order to determine the same upper bound. The correctness of the algorithm follows easily from Lemmas 13 and 15. The fact that the robots compute a $O(\frac{L}{\ell}n)$ upper bound follows from the fact that the robots will traverse the cycle $\frac{L}{\ell}$ times. The asymptotic running time of the algorithm is identical to that of **COUNTROBOTS** with ℓ replaced with L . The random-bit complexity does not change. Lemma 17 follows without proof from this discussion.

Computing an upper bound on ℓ from an upper bound on n . Here we simply use the algorithm **BOUNDCYCLE** with the input $N \geq n$ instead of n .

Proof. The proof is nearly identical to that of Lemma 11 except we replace n with N and require at least t_0 rounds where t_0 is the first round in which $L_t = N \cdot 2^t \geq 2 \lceil \frac{N}{n} \rceil \ell$, i.e. $t_0 = \lceil \log(\lceil \frac{N}{n} \rceil \frac{2\ell}{N}) \rceil = O(\log \lceil \frac{\ell}{n} \rceil)$. ◀



■ **Figure 1** Left: The instance I with two robots r_1 and r_2 on a cycle of length ℓ . Right: The instance I' with four robots r_1 , r_2 , r'_1 , and r'_2 on a cycle of length 2ℓ .

5 Impossibility results

In the previous sections we have developed Las Vegas algorithms which solve ELECT when one of n or ℓ is known exactly to the robots. We have also developed Monte Carlo algorithms when only upper-bounds on n and/or ℓ are known. In the sequel we demonstrate that, unless the robots know at least one of n or ℓ exactly, there does not exist a Las Vegas algorithm which solves ELECT.

► **Theorem 19.** *Assume that the robots do not know ℓ nor n exactly. Then there is no Las Vegas type algorithm which solves ELECT.*

To demonstrate this we first prove the weaker statement that a Las Vegas algorithm cannot exist if the robots know nothing of n nor ℓ .

► **Lemma 20.** *If neither n nor ℓ is available then there is no Las Vegas type algorithm which solves ELECT.*

Proof. To derive a contradiction suppose that there is a Las Vegas type algorithm A which solves the problem. Consider an instance I in which there are two robots r_1 and r_2 at antipodal positions on a cycle with circumference ℓ . Since A solves the problem it terminates with probability 1 in a finite, though unpredictable, amount of time T . Let O_1 and O_2 be the sequence of outcomes of coin flips of r_1 and r_2 .

Consider another instance I' in which there are four robots r_1 , r_2 , r'_1 , and r'_2 at equally spaced locations of a cycle with circumference 2ℓ such that r_1 and r'_1 (resp. r_2 and r'_2) are antipodal (see Figure 1). Assume that the pair r_1 and r'_1 (resp. r_2 and r'_2) each have the same orientation and each receives the outcome of coin flips O_1 (resp. O_2). Call an encounter between a pair of robots r_1 and r_2 a *left encounter* (resp. a *right encounter*) if r_1 and r_2 encounter each other while either r_1 is moving CCW and r_2 is stationary, r_2 is moving CW and r_1 is stationary, or r_1 is moving CCW and r_2 is moving CW (resp. while either r_1 is moving CW and r_2 is stationary, r_2 is moving CCW and r_1 is stationary, or r_1 is moving CW and r_2 is moving CCW). Then for every left encounter of r_1 and r_2 in I there is a corresponding identical left encounter between r_1 and r_2 in I' and between r'_1 and r'_2 in I' . Likewise, for every right encounter of r_1 and r_2 in I there are corresponding identical right encounters between r_1 and r'_2 in I and between r_2 and r'_1 in I' . Thus, at time T , each of r_1 and r'_1 (resp. r_2 and r'_2) in I' must come to the same conclusion as r_1 (resp. r_2) in I . However, this implies that at the end of the execution of A in I' we will have elected two leaders. Since there is a positive probability that r_1 and r'_1 (resp. r_2 and r'_2) both get the outcome of coin flips O_1 (resp. O_2) then there is a positive probability that A incorrectly terminates in time T . This contradicts our assumption that A correctly terminates with probability one. ◀

It is not hard to extend this to the situation that the robots know only an upper bound on n :

► **Corollary 21.** *Suppose that the robots only know an upper bound N on n . Then there is no Las Vegas type algorithm which solves ELECT.*

Proof. To derive a contradiction suppose that there is a Las Vegas type algorithm A for ELECT. We use the instances I and I' given in the proof of Theorem 19. Provided that $N = 5$ is given, consider the execution of A for I . Then in time T , A terminates in which O_1 and O_2 are the sequences of outcomes of the coin flips of r_1 and r_2 .

Then A terminates incorrectly in time T , when it is executed for I' with $N = 5$, as argued in the proof of Lemma 20, which is a contradiction. ◀

Proof. (Theorem 19) Assume that a Las Vegas algorithm A exists by which the robots can solve ELECT if they know upper bounds N and L on n and ℓ respectively. Now consider an instance of the problem when only an upper bound N on n is known. Then by Lemma 18 there exists a Las Vegas algorithm by which the robots can determine L . Once the robots know L they run algorithm A to elect a leader. This implies that there exists a Las Vegas algorithm by which the robots can elect a leader when they only know an upper bound N on n . This contradicts the previous result of Corollary 21 which states that such an algorithm cannot exist. We may therefore conclude that a Las Vegas algorithm does not exist if the robots know both upper bounds N and L . This further implies that a Las Vegas algorithm does not exist when the robots know only L . ◀

6 Extensions and Open Questions

Here we discuss why the consistent orientation assumption is unnecessary; the extension of our election algorithms to the GATHER problem; and other extensions/open problems.

Orientation. In the previous sections we have assumed that the robots have consistent orientations. Here we will argue why this assumption is not required.

First, observe that with the consistent orientation assumption it will never occur that two moving robots encounter each other. By removing this assumption we will have to deal with the extra encounters involving two robots which move in opposite directions. For most of these encounters the solution is simple – the two moving robots will simply ignore each other. A more problematic encounter occurs if two moving robots encounter a stationary robot from opposite directions at the same time. Fortunately, this is also easily remedied – we simply have the stationary robot choose to “process” the moving robot arriving from its, say, CW direction first. We can thus conclude that all of our results still hold if we remove the consistent orientation assumption.

Gathering. In the previous sections our primary goal has been on how to solve ELECT. However, it is easy to see that our algorithms also solve GATHER at no extra cost. Indeed, consider Algorithm 1 where, during the election process, robots only enter a FOLLOWER state when they merge with a remaining CANDIDATE robot. When only a single CANDIDATE remains all other robots will be part of its stack. This is also the case for Algorithm 3, however, since this is a Monte Carlo algorithm, there is a bounded probability that more than one stack remains when the algorithm terminates. Thus, by construction, Algorithm 1 is a Las Vegas algorithm which solves GATHER and Algorithm 3 is a Monte Carlo algorithm which solves GATHER. Clearly, the complexities of these algorithms remain the same when applied to either the ELECT or GATHER problems.

6.1 Discussions and Open Problems

In this paper we have studied the ELECT and GATHER problems for n identical robots in the \mathcal{CT} model on a continuous cycle of length ℓ . We have established several results including optimal algorithms with respect to time and random bits when the robots know ℓ , or an upper bound $L = O(\ell)$ (in the latter case with high probability).

There are a number of open questions remaining. Firstly, we have not considered the possibility (or lack thereof) of a Monte Carlo algorithm when the robots do not possess any knowledge of n or ℓ . In addition, we have only considered a fully synchronous time model and a natural extension is therefore to study ELECT and GATHER when this assumption is removed. In particular one can consider a model where the robots do not begin an algorithm simultaneously but otherwise their respective clocks tick at the same rate, or a model where even the robots' clocks are not synchronized.

References

- 1 Hagit Attiya and Yishay Mansour. Language complexity on the synchronous anonymous ring. *Theoretical Computer Science*, 53(2-3):169–185, 1987.
- 2 Hagit Attiya, Marc Snir, and Manfred K. Warmuth. Computing on an Anonymous Ring. *J. ACM*, 35(4):845–875, 1988.
- 3 Rena Bakhshi, Wan Fokkink, Jun Pang, and Jaco van de Pol. Leader Election in Anonymous Rings: Franklin Goes Probabilistic. In *5th IFIP International Conference On Theoretical Computer Science (TCS)*, pages 57–72, 2008.
- 4 Francesco Bullo, Jorge Cortes, and Sonia Martinez. *Distributed Control of Robotic Networks*. Princeton University Press, 2009.
- 5 Mark Cielibak, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Distributed computing by mobile robots: Gathering. *SIAM Journal on Computing*, 41(4):829–879, 2012.
- 6 Reuven Cohen and David Peleg. Convergence Properties of the Gravitational Algorithm in Asynchronous Robot Systems. *SIAM Journal on Computing*, 34(6):1516–1528, 2005.
- 7 Jurek Czyzowicz, Leszek Gasiñec, Adrian Kosowski, and Evangelos Kranakis. Boundary Patrolling by Mobile Agents with Distinct Maximal Speeds. In *19th Annual European Symposium on Algorithms, ESA*, pages 701–712, 2011.
- 8 Jurek Czyzowicz, Kostantinos Georgiou, and Evangelos Kranakis. *Patrolling*. In P. Flocchini, G. Prencipe, and N. Santoro, editors, *Distributed Computing by Mobile Entities*, chapter 15, pages 371–400. Springer, 2019.
- 9 Jurek Czyzowicz, Evangelos Kranakis, Dominik Pajak, and Najmeh Taleb. Patrolling by Robots Equipped with Visibility. In *21st International Colloquium on Structural Information and Communication Complexity, SIROCCO*, pages 224–234, 2014.
- 10 Yoann Dieudonné, Franck Petit, and Vincent Franck. Leader election problem versus pattern formation problem. In *24th International Symposium on Distributed Computing (DISC)*, pages 267–281, 2010.
- 11 Ofer Feinerman, Amos Korman, Shay Kutten, and Yoav Rodeh. Fast rendezvous on a cycle by agents with different speeds. In *5th International Conference on Distributed Computing and Networking, ICDCN*, pages 1–13, 2014.
- 12 Paola Flocchini, Evangelos Kranakis, Danny Krizanc, Flaminia L. Luccio, and Nicola Santoro. Sorting and election in anonymous asynchronous rings. *Journal of Parallel and Distributed Computing*, 64(2):254–265, 2004.
- 13 Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Self-deployment of mobile sensors on a ring. *Theoretical Computer Science*, 402(1):67–80, 2008.
- 14 Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. *Distributed Computing by Oblivious Mobile Robots*. Morgan & Claypool, 2012.

- 15 Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. *Distributed Computing by Mobile Entities*. Springer, 2019.
- 16 Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. Gathering of asynchronous mobile robots with limited visibility. *Theoretical Computer Science*, 337(1-3):147–168, 2006.
- 17 Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. Arbitrary pattern formation by asynchronous, anonymous, oblivious robots. *Theoretical Computer Science*, 407(1-3):412–447, 2008.
- 18 Greg N. Frederickson and Nicola Santoro. Breaking Symmetry in Synchronous Networks. In *1st Aegean Workshop on Computing (now SPAA)*, pages 26–33, 1986.
- 19 Nao Fujinaga, Yukiko Yamauchi, Shuji Kijima, and Masafumi Yamashita. Asynchronous pattern formation by anonymous oblivious mobile robots. *SIAM Journal on Computing*, 44(3):740–785, 2015.
- 20 Noam Gordon, Israel A. Wagner, and Alfred M. Bruckstein. A randomized gathering algorithm for multiple robots with limited sensing capabilities. In *International Workshop on Multi-Agent Robotic Systems*, 2005.
- 21 Rajiv Gupta, Scott A. Smolka, and Shaji Bhaskar. On Randomization in Sequential and Distributed Algorithms. *ACM Comput. Surv.*, 26(1):7–86, 1994.
- 22 Evan Huus and Evangelos Kranakis. Rendezvous of many agents with different speeds in a cycle. In *14th International Conference on Ad-Hoc Networks and Wireless, ADHOC-NOW*, pages 195–209, 2015.
- 23 Alon Itai and Michael Rodeh. Symmetry breaking in distributed networks. *Information and Computation*, 88(1):60–87, 1990.
- 24 Taisuke Izumi, Tomoko Izumi, Sayaka Kamei, and Fukuhito Ooshita. Randomized gathering of mobile robots with local-multiplicity detection. In *11th International Symposium on Stabilizing, Safety, and Security of Distributed Systems, SSS*, pages 384–398, 2009.
- 25 Richard M. Karp. Probabilistic Recurrence Relations. *J. ACM*, 41(6):1136–1150, 1994.
- 26 Evangelos Kranakis, Danny Krizanc, and Euripides Markou. *The Mobile Agent Rendezvous Problem in the Ring*. Morgan & Claypool, 2010.
- 27 Ji Lin, A. Stephen Morse, and Brian D.O. Anderson. The Multi-Agent Rendezvous Problem. Parts 1 and 2. *SIAM Journal on Control and Optimization*, 46(6):2096–2147, 2007.
- 28 Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
- 29 Fukuhito Ooshita, Shinji Kawai, Hirotsugu Kakugawa, and Toshimitsu Masuzawa. Randomized gathering of mobile agents in anonymous unidirectional ring networks. *IEEE Transactions on Parallel and Distributed Systems*, 25(5):1289–1296, 2014.
- 30 Yukiko Yamauchi. *Symmetry of Anonymous Robots*. In P. Flocchini, G. Prencipe, and N. Santoro, editors, *Distributed Computing by Mobile Entities*, chapter 6, pages 109–133. Springer, 2019.

A **Description of Karp’s theorem**

Consider the stochastic recurrence relation

$$T(n) = a(n) + T(h(n)) \tag{2}$$

which describes a process in which we start with an input of size n and after investing some amount of resources (represented by $a(n)$) we are left with a smaller problem of size $h(n)$ upon which we recurse. As it applies here, n represents the number of candidate robots, $a(n)$ will represent the number of rounds/time/random-bits, and $h(n)$ the expected number of robots remaining after one iteration of a leader election algorithm.

Formally, n is a nonnegative integer variable; $a(n)$ a nonnegative real-valued function of n ; $h(n)$ a random variable with support $[0, n]$ and expectation bounded by $m(n)$; and $m(n)$ is a nonnegative real-valued function of n . The equation $\tau(n) = a(n) + \tau(m(n))$ is the deterministic analogue of (2) and, when it exists, has the unique least nonnegative solution $u(n)$ given by

$$u(n) = \sum_{k=0}^{\infty} a(m^{[k]}(n)) \quad (3)$$

with $m^{[k]}(n)$ inductively defined by $m^{[0]}(n) = n$ and $m^{[k]}(n) = m(m^{[k-1]}(n))$, $k \geq 1$. Karp proved the following:

► **Theorem 22** (Karp [25], Theorems 1.1 and 1.2). *Consider the stochastic recurrence (2), a continuous function $m(n)$ with $m(n)/n$ non-decreasing, and let $u(n)$ be given by (3).*

1. *Suppose there is a constant d such that $a(n) = 0$, $n < d$; and $a(n) = 1$, $n \geq d$. Let $c_k = \min\{n | u(n) \geq k\}$. Then, for every positive integer n and every positive integer w ,*

$$\Pr[T(n) \geq u(n) + w] \leq \left(\frac{m(n)}{n}\right)^{w-1} \frac{m(n)}{c_{u(n)}}.$$
2. *Suppose that $a(n)$ is strictly increasing on $\{n | a(n) > 0\}$. Then, for every positive integer n and every positive integer w ,*

$$\Pr[T(n) > u(n) + wa(n)] \leq \left(\frac{m(n)}{n}\right)^w.$$

B Pseudocode for algorithms of Section 3.1

■ **Algorithm 1** ELECTLV(n, ℓ).

Input: $n > 0$ (integer); $\ell > 0$ (real); ▷ The number of robots and the length of the cycle.

Initialize: $state(r) \leftarrow \text{CANDIDATE}$; $\text{CNR}(r) \leftarrow 1$; $t \leftarrow 0$;

Begin:

1: **repeat**

2: $D \leftarrow \min\left\{\frac{\ell}{2}, \frac{\ell}{n} \left(\frac{4}{3}\right)^t\right\}$;

3: $\text{ELECTIONROUND}(D)$; $t \leftarrow t + 1$; ▷ Run one election round.

4: **if** $\text{CNR}(r) = n$ **then** $state(r) \leftarrow \text{LEADER}$; ▷ Stack contains n robots, terminate.

5: **until** $state(r) = \text{FOLLOWER}$ or LEADER

:End

■ **Algorithm 2** ELECTIONROUND(D).

Input: $D > 0$ (real);

Begin: $b(r) \leftarrow \text{flip}()$;

1: **if** $b(r) = \text{H}$ **then** ▷ H was flipped

2: Move CCW a distance D ; CW a distance $2D$; CCW a distance D ;

3: **if** a robot s with $b(s) = \text{T}$ is encountered while moving **then**

4: $\text{CNR}(r) \leftarrow \text{CNR}(r) + \text{CNR}(s)$; ▷ Update $\text{CNR}(r)$ since s will merge with r .

5: **else** ▷ T was flipped

6: Remain stationary for time $4D$;

7: **if** a robot s with $b(s) = \text{H}$ is encountered while waiting **then**

8: $state(r) = \text{FOLLOWER}$;

9: Merge with robot s ;

:End

C Pseudocode for algorithms of Section 3.2**Algorithm 3** ELECTMC(N, L, w).

Input: $N > 0$ (integer); $L > 0$ (real); $w \geq 0$ (integer); \triangleright upper bounds on n and ℓ ; termination parameter w .

Initialize: $state(r) \leftarrow$ CANDIDATE; $t \leftarrow 0$; $t_\infty \leftarrow \lceil \log_{4/3}(n) \rceil + w$; $\triangleright t_\infty =$ termination round.

Begin:

- 1: **repeat**
- 2: $D_t \leftarrow \min \left\{ \frac{L}{2}, \frac{L}{N} \left(\frac{4}{3} \right)^t \right\}$;
- 3: ELECTIONROUND(D_t); $t \leftarrow t + 1$; \triangleright Run one election round.
- 4: **until** $state(r) =$ FOLLOWER or $t = t_\infty$
- 5: **if** $state(r) =$ CANDIDATE **then** $state(r) \leftarrow$ LEADER;

:End

D Pseudocode for algorithms of Section 4.1**Algorithm 4** COUNTFLIPS(D).

Input: $D > 0$ (real); \triangleright An estimate of the length of the cycle.

Initialize: $N_H(r) \leftarrow 0$; $N_T(r) \leftarrow 0$; \triangleright To count the robots flipping H and T.

Begin:

- 1: **if** $b(r) =$ H **then** \triangleright H was outcome of last coin flip
- 2: Move CW a distance D ;
- 3: **if** a robot s with $b(s) =$ T is encountered while moving **then** $N_T(r) \leftarrow N_T(r) + 1$;
- 4: Move CCW a distance D ;
- 5: **if** $N_H(r) = 0$ and a robot s with $b(s) =$ T is encountered while moving **then**
- 6: $N_H(r) \leftarrow N_H(s)$; \triangleright Determine N_H .
- 7: **else** \triangleright T was outcome of last coin flip
- 8: Wait for time D ;
- 9: **if** a robot s with $b(s) =$ H is encountered while waiting **then** $N_H(r) \leftarrow N_H(r) + 1$;
- 10: Wait for time D ;
- 11: **if** $N_T(r) = 0$ and a robot s with $b(s) =$ H is encountered while waiting **then**
- 12: $N_T(r) \leftarrow N_T(s)$; \triangleright Determine N_T .
- 13: **return** $N_H(r) + N_T(r)$; \triangleright Returns 0 if all robots flipped the same.

:End

Algorithm 5 FINDMIN(L, N_0).

Input: $L > 0$ (real); N_0 (real); \triangleright upper bound cycle length; quantity to find the minimum of.

Initialize: $N(r) \leftarrow N_0$; \triangleright Will contain the minimum of the inputs N_0 .

Begin:

- 1: **if** $b(r) =$ H **then** \triangleright H was outcome of last coin-flip
- 2: Move CW a distance L and then move CCW a distance L ;
- 3: **if** robot s with $b(s) =$ T is encountered **then** $N(r) \leftarrow \min\{N(r), N(s)\}$;
- 4: **else** \triangleright T was outcome of last coin-flip
- 5: Wait for time $2L$;
- 6: **if** robot s with $b(s) =$ H is encountered **then** $N(r) \leftarrow \min\{N(r), N(s)\}$;
- 7: **return** $N(r)$;

:End

Algorithm 6 COUNTROBOTS(ℓ).

Input: $\ell > 0$ (real); ▷ The length of the cycle.
Initialize: $N(r)$; ▷ Will contain the computed value of n .
Begin:
 1: **repeat**
 2: $b(r) \leftarrow \text{flip}()$; $N(r) \leftarrow \text{COUNTFLIPS}(\ell)$;
 3: **until** $N(r) > 0$
 4: **return** $N(r)$;
:End

Algorithm 7 BOUNDCYCLE(n).

Input: $n > 0$ (integer); ▷ The number of robots.
Initialize: $N(r)$; $t \leftarrow -1$;
Begin:
 1: **repeat**
 2: $t \leftarrow t + 1$;
 3: $L_t = n \cdot 2^{t-1}$;
 4: $b(r) \leftarrow \text{flip}()$;
 5: $N(r) \leftarrow \text{COUNTFLIPS}(L_t)$;
 6: $N(r) \leftarrow \text{FINDMIN}(L_t, N(r))$;
 7: **until** $N(r) > n$
 8: **return** $\frac{2L_t}{\lfloor N(r)/n \rfloor}$;
:End

E Pseudocode for algorithms of Section 4.2

Algorithm 8 BOUNDROBOTS(L).

Input: $L > 0$, real ▷ upper bound on the length of the cycle.
Initialize: $N(r)$; ▷ Will contain the computed upper bound on n .
Begin:
 1: **repeat**
 2: $b(r) \leftarrow \text{flip}()$; $N(r) \leftarrow \text{COUNTFLIPS}(L)$;
 3: **until** $N(r) > 0$
 4: $N(r) \leftarrow \text{FINDMIN}(L, N(r))$;
 5: **return** $N(r)$;
:End

Strategy-Proof Approximation Algorithms for the Stable Marriage Problem with Ties and Incomplete Lists

Koki Hamada 

NTT Corporation, 3-9-11, Midori-cho, Musashino-shi, Tokyo 180-8585, Japan
Graduate School of Informatics, Kyoto University, Yoshida-Honmachi,
Sakyo-ku Kyoto 606-8501, Japan
koki.hamada.rb@hco.ntt.co.jp

Shuichi Miyazaki 

Academic Center for Computing and Media Studies, Kyoto University, Yoshida-Honmachi,
Sakyo-ku, Kyoto 606-8501, Japan
shuichi@media.kyoto-u.ac.jp

Hiroki Yanagisawa 

IBM Research – Tokyo, 19-21, Hakozaki-cho, Nihombashi, Chuoh-ku, Tokyo 103-8510, Japan
yanagis@jp.ibm.com

Abstract

In the stable marriage problem (SM), a mechanism that always outputs a stable matching is called a *stable mechanism*. One of the well-known stable mechanisms is the man-oriented Gale-Shapley algorithm (MGS). MGS has a good property that it is strategy-proof to the men’s side, i.e., no man can obtain a better outcome by falsifying a preference list. We call such a mechanism a *man-strategy-proof mechanism*. Unfortunately, MGS is not a woman-strategy-proof mechanism. (Of course, if we flip the roles of men and women, we can see that the woman-oriented Gale-Shapley algorithm (WGS) is a woman-strategy-proof but not a man-strategy-proof mechanism.) Roth has shown that there is no stable mechanism that is simultaneously man-strategy-proof and woman-strategy-proof, which is known as Roth’s impossibility theorem.

In this paper, we extend these results to the stable marriage problem with ties and incomplete lists (SMTI). Since SMTI is an extension of SM, Roth’s impossibility theorem takes over to SMTI. Therefore, we focus on the one-sided-strategy-proofness. In SMTI, one instance can have stable matchings of different sizes, and it is natural to consider the problem of finding a largest stable matching, known as MAX SMTI. Thus we incorporate the notion of approximation ratios used in the theory of approximation algorithms. We say that a stable-mechanism is a *c-approximate-stable mechanism* if it always returns a stable matching of size at least $1/c$ of a largest one. We also consider a restricted variant of MAX SMTI, which we call MAX SMTI-1TM, where only men’s lists can contain ties (and women’s lists must be strictly ordered).

Our results are summarized as follows: (i) MAX SMTI admits both a man-strategy-proof 2-approximate-stable mechanism and a woman-strategy-proof 2-approximate-stable mechanism. (ii) MAX SMTI-1TM admits a woman-strategy-proof 2-approximate-stable mechanism. (iii) MAX SMTI-1TM admits a man-strategy-proof 1.5-approximate-stable mechanism. All these results are tight in terms of approximation ratios. Also, all these results apply for strategy-proofness against coalitions.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Theory of computation → Algorithmic game theory

Keywords and phrases Stable marriage problem, strategy-proofness, approximation algorithm, ties, incomplete lists

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.9

Funding *Shuichi Miyazaki*: Supported by JSPS KAKENHI Grant Number JP16K00017.

Acknowledgements We would like to thank the anonymous reviewers for their helpful comments.



© Koki Hamada, Shuichi Miyazaki, and Hiroki Yanagisawa;
licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 9; pp. 9:1–9:14

Leibniz International Proceedings in Informatics



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

1 Introduction

An instance of the *stable marriage problem* (*SM*) [5] consists of n men m_1, m_2, \dots, m_n , n women w_1, w_2, \dots, w_n , and each person's preference list, which is a total order of all the members of the opposite gender. If a person q_i precedes a person q_j in a person p 's preference list, then we write $q_i \succ_p q_j$ and interpret it as “ p prefers q_i to q_j ”. In this paper, we denote a preference list in the following form:

$$m_2 : w_3 \ w_1 \ w_4 \ w_2,$$

which means that m_2 prefers w_3 best, w_1 second, w_4 third, and w_2 last (this example is for $n = 4$).

A *matching* is a set of n (man, woman)-pairs in which no person appears more than once. For a matching M , $M(p)$ denotes the partner of a person p in M . If, for a man m and a woman w , both $w \succ_m M(m)$ and $m \succ_w M(w)$ hold, then we say that (m, w) is a *blocking pair* for M or (m, w) *blocks* M . Note that both m and w have incentive to be matched with each other ignoring the given partner, so it can be thought of as a threat for the current matching M . A matching with no blocking pair is a *stable matching*. It is known that any instance admits at least one stable matching, and one can be found by the *Gale-Shapley algorithm* (or *GS algorithm* for short) in $O(n^2)$ time [5]. There have been a plenty of research results on this problem from viewpoints of Economics, Computer Science, Mathematics, etc (see [7, 21, 14] e.g.).

1.1 Strategy-Proofness

The stable marriage problem can be seen as a game among participants, who have true preferences in mind, but may submit a falsified preference list hoping to obtain a better partner than the one assigned when true preference lists are used. Formally, let S be a *mechanism*, that is, a mapping from instances to matchings, and we denote $S(I)$ the matching output by S for an instance I . We say that S is a *stable mechanism* if, for any instance I , $S(I)$ is a stable matching for I . For a mechanism S , let I be an instance, M be a matching such that $M = S(I)$, and p be a person. We say that p *has a successful strategy in* I if there is an instance I' in which people except for p have the same preference lists in I and I' , and p prefers M' to M (i.e., $M'(p) \succ_p M(p)$ with respect to p 's preference list in I), where M' is a matching such that $M' = S(I')$. This situation is interpreted as follows: I is the set of true preference lists, and by submitting a falsified preference list (which changes the set of lists to I'), p can obtain a better partner $M'(p)$. We say that S is a *strategy-proof mechanism* if, when S is used, no person has a successful strategy in any instance. Also we say that S is a *man-strategy-proof mechanism* if, when S is used, no man has a successful strategy in any instance. A *woman-strategy-proof mechanism* is defined analogously. A mechanism is a *one-sided-strategy-proof mechanism* if it is either a man-strategy-proof mechanism or a woman-strategy-proof mechanism.

It is known that there is no strategy-proof stable mechanism for SM [18], which is known as *Roth's impossibility theorem*. By contrast, the man-oriented GS algorithm, *MGS* for short, (in which men send and women receive proposals; see Appendix A) is a man-strategy-proof stable mechanism for SM [18, 2]. Of course, by the symmetry of men and women, the woman-oriented GS algorithm (*WGS*) is a woman-strategy-proof stable mechanism.

1.2 Ties and Incomplete Lists

One of the most natural extensions of SM is the *Stable Marriage with Ties and Incomplete lists*, denoted *SMTI*. An instance of SMTI consists of n men, n women, and each person's preference list. A preference list may include *ties*, which represent indifference between two or more persons, and may be *incomplete*, meaning that a preference list may contain only a *subset* of people in the opposite gender. Such a preference list may be of the following form:

$$m_2 : w_3 (w_1 w_4),$$

which represents that m_2 prefers w_3 best, w_1 and w_4 second with equal preference, but does not want to be matched with w_2 . If a person q is included in p 's preference list, we say that q is *acceptable* to p . A *matching* is a set of mutually acceptable (man, woman)-pairs in which no person appears more than once. The *size* of a matching M , denoted $|M|$, is the number of pairs in M . For a matching M , (m, w) is a *blocking pair* if (i) m and w are acceptable to each other, (ii) m is single in M or $w \succ_m M(m)$, and (iii) w is single in M or $m \succ_w M(w)$. A matching without blocking pairs is a *stable matching*. (When ties come into consideration, there are three definitions for stability, *super*, *strong*, and *weak* stabilities. Here we are considering weak stability which is the most natural notion among the three. In the case of super and strong stabilities, there exist instances that do not admit a stable matching. See [7, 14] for more details.)

Note that in the case of SM, the size of a matching is always n by definition, but it may be less than n in the case of SMTI. In fact, there is an SMTI-instance that admits stable matchings of different sizes, and the problem of finding a stable matching of the maximum size, called *MAX SMTI*, is NP-hard [10, 15]. There are a plenty of approximability and inapproximability results for MAX SMTI. The current best upper bound on the approximation ratio is 1.5 [16, 17, 11] and lower bounds are $33/29 \simeq 1.1379$ assuming $P \neq NP$ and $4/3 \simeq 1.3333$ assuming the Unique Games Conjecture (UGC) [22]. There are several attempts to obtain better algorithms (e.g., polynomial-time exact algorithms or polynomial-time approximation algorithms with better approximation ratio) for restricted instances; one of the most natural restrictions is to admit ties in preference lists of only one gender, which we call *SMTI-1T*. MAX SMTI-1T (i.e., the problem of finding a maximum cardinality stable matching in SMTI-1T) remains NP-hard, and as for the approximation ratio, the current best upper bound is $1 + 1/e \simeq 1.368$ [13] and lower bounds are $21/19 \simeq 1.1052$ assuming $P \neq NP$ and $5/4 = 1.25$ assuming UGC [8, 22].

1.3 Our Contributions

In this paper, we consider the strategy-proofness in MAX SMTI, and investigate the trade-off between strategy-proofness and approximability. In the case of incomplete preference lists, there may be unmatched (i.e., single) persons. Thus, we have to extend the definition of a person preferring one matching to another. We say that a person p prefers M' to M if either $M'(p) \succ_p M(p)$ holds or p is single in M but is matched in M' with some acceptable woman. Then the definition of strategy-proofness for SM naturally takes over to SMTI.

Let I be a MAX SMTI instance and M_{opt} be a maximum size stable matching for I . A stable matching M for I is called an *r*-*approximate solution* for I if $\frac{|M_{opt}|}{|M|} \leq r$. A stable mechanism S is called an *r*-*approximate-stable mechanism* if $S(I)$ is an *r*-approximate solution for any MAX SMTI instance I .

Firstly, since SMTI is a generalization of SM, Roth's impossibility theorem for SM [18] holds also for MAX SMTI (regardless of approximability):

► **Proposition 1.** *There is no strategy-proof stable mechanism for MAX SMTI.*

Therefore, we focus on *one-sided*-strategy-proofness. We show that there is a 2-approximate-stable mechanism, which is achieved by a simple extension of the GS algorithm. We also show that this result is tight:

► **Theorem 2.** *MAX SMTI admits both a man-strategy-proof 2-approximate-stable mechanism and a woman-strategy-proof 2-approximate-stable mechanism. On the other hand, for any positive ϵ , MAX SMTI admits neither a man-strategy-proof $(2 - \epsilon)$ -approximate-stable mechanism nor a woman-strategy-proof $(2 - \epsilon)$ -approximate-stable mechanism.*

We next consider a restricted version, MAX SMTI-1T. Throughout the paper, we assume that ties appear in men's lists only (and women's lists must be strict). In the following, we use the name *MAX SMTI-1TM* to stress that only men's preference lists may contain ties. As for woman-strategy-proofness, we obtain the same result as for MAX SMTI, which is a direct consequence of Theorem 2:

► **Corollary 3.** *MAX SMTI-1TM admits a woman-strategy-proof 2-approximate-stable mechanism, but no woman-strategy-proof $(2 - \epsilon)$ -approximate-stable mechanism for any positive ϵ .*

For man-strategy-proofness, we can reduce the approximation ratio to 1.5, which is the main result of this paper.

► **Theorem 4.** *MAX SMTI-1TM admits a man-strategy-proof 1.5-approximate-stable mechanism, but no man-strategy-proof $(1.5 - \epsilon)$ -approximate-stable mechanism for any positive ϵ .*

We remark that no assumptions on running times are made for our negative results, while algorithms in our positive results run in linear time. Note also that the current best polynomial-time approximation algorithms for MAX SMTI and MAX SMTI-1TM have the approximation ratios better than those in our negative results (Theorems 2 and 4). Hence our results provide gaps between polynomial-time computation and strategy-proof computation.

Coalition. In the above discussion, man-strategy-proofness (woman-strategy-proofness) is defined in terms of a manipulation of a preference list by one man (woman). We can extend this notion to a *coalition* of men (or women) as follows; a coalition C of men has a successful strategy if there is a way of falsifying preference lists of members of C which improves the outcome of *every* member of C . It is known that MGS is strategy-proof against a coalition of men in this sense (Theorem 1.7.1 of [7]), and this strategy-proofness holds also in the stable marriage with incomplete lists (SMI) (page 57 of [7]). Since all our strategy-proofness results (Lemmas 5 and 11) are attributed to strategy-proofness of MGS in SMI, we can easily modify the proofs so that Theorem 2, Corollary 3, and Theorem 4 hold for strategy-proofness against coalitions.

Many-to-One Setting. Clearly, the negative parts of Theorem 2, Corollary 3, and Theorem 4 hold for a many-to-one extension of MAX SMTI, denoted *MAX HRT*. Also, we can show that man-strategy-proofness in Theorems 2 and 4 carry over to resident-strategy-proofness in MAX HRT by cloning hospitals (see e.g., page 283 of [9] for cloning). By contrast, woman-strategy-proofness in Theorem 2 and Corollary 3 do not hold for hospital-strategy-proofness in MAX HRT; there is no hospital-strategy-proof stable mechanism even without ties (see Sec. 1.7.3 of [7]).

Overview of Techniques. Since MGS is a man-strategy-proof stable mechanism for SM, such types of algorithms are good candidates for proving the positive part of Theorem 4. Existing 1.5-approximation algorithms for MAX SMTI for one-sided ties are of GS-type, but in these algorithms, proposals are made from the side with no ties (women, in our case), so we cannot use them for our purpose. As mentioned above, there are 1.5-approximation algorithms for the general MAX SMTI [16, 17, 11], which are fortunately of GS-type and can handle proposals from the side with ties (men, in our case). Hence one may expect that these algorithms will work. However, it is not the case. The main reason is as follows: Suppose that some man m is going to propose to a woman, and the head of m 's current list is a tie, which is a mixture of unmatched and matched women. In this case, m 's proposal will be sent to an unmatched woman, say w . Suppose that, just one step before, another man m' has proposed to w' . Then if m' moves w to the position just before w' , he can make w already matched when m is about to propose to her, and as a result of this, m does not propose to w but to another unmatched woman. In this way, a man can change another man's proposal order, which destroys the strategy-proofness (see Appendix B for more details). To overcome it, we modify Király's 1.5-approximation algorithm [11] (or more precisely, the algorithm M-KNA given in Appendix B) to be *robust* in the sense that a man's proposal order is not affected by other men's preference lists.

Ties or Incomplete Lists. When only ties are present (SMT) or only incomplete lists are present (SMI), all the stable matchings of one instance have the same cardinality. The former is due to the fact that any stable matching is a perfect matching, and the latter is due to the Rural Hospitals theorem [6, 19, 20]. Hence approximability is not an important issue in these cases. As for strategy-proofness, since SMT and SMI are generalizations of SM, Roth's impossibility theorem holds and no strategy-proof stable mechanism exists. Existence of one-sided strategy-proofness for SMI is already known as we have mentioned in "Coalition" part above, and that for SMT follows directly from Theorem 2.

1.4 Related Work

There are some literature studying trade-offs between approximability and strategy-proofness. Krysta et al. [12] consider to approximate the size of a Pareto optimal matching in the House Allocation problem, where preference lists may include ties. They give upper and lower bounds on the approximation ratio of randomized strategy-proof mechanisms for computing a Pareto optimal matching. Dughmi and Ghosh [3] study the generalized assignment problem (GAP) and its variants. Their objective is to maximize the sum of the values of the assigned jobs. They present a strategy-proof $O(\log n)$ -approximate mechanism for the GAP, where n represents the number of jobs.

The following papers discuss strategy-proofness in the stable matching problem with indifference. Erdil and Ergin [4] consider the Hospitals/Residents problem where only hospitals' preference lists may have ties. They consider the algorithm that first breaks ties according to a tie-breaking rule τ and then applies the resident-oriented GS algorithm (let us call this algorithm GS^τ). They give an instance and a tie-breaking rule τ such that GS^τ does not produce a resident-optimal stable matching. They also show that seeking for a resident-optimal stable matching loses strategy-proofness, that is, no deterministic resident-optimal stable mechanism can be resident-strategy-proof. Abdulkadiroğlu et al. [1] give an evidence to support GS^τ . They show that for any tie-breaking rule τ , no resident-strategy-proof mechanism dominates GS^τ (with respect to residents).

2 Results for MAX SMTI

In this section, we give a proof of Theorem 2. We start with the positive part:

► **Lemma 5.** *MAX SMTI admits both a man-strategy-proof 2-approximate-stable mechanism and a woman-strategy-proof 2-approximate-stable mechanism.*

Proof. Consider a mechanism S^* that is described by the following algorithm. Given a MAX SMTI instance I , S^* first breaks each tie so that persons in a tie are ordered increasingly in their indices, that is, if q_i and q_j are in the same tie of p 's list, then after the tie break $q_i \succ_p q_j$ holds if and only if $i < j$. Let I' be the resulting instance. Its preference lists are incomplete but do not include ties; such an instance is called an *SMTI instance*. It then applies MGS modified for SMTI [7] to I' and obtains a stable matching M for I' . It is easy to see that M is stable for I . Also it is well-known that in MAX SMTI, any stable matching is a 2-approximate solution [15]. Hence S^* is a 2-approximate-stable mechanism.

We then show that S^* is a man-strategy-proof mechanism. Suppose not. Then there is a MAX SMTI instance I and a man m who has a successful strategy in I . Let J be a MAX SMTI instance in which only m 's preference list differs from I , and by using it m obtains a better outcome. Let M_I and M_J be the outputs of S^* on I and J , respectively. Then m prefers M_J to M_I , that is, either (i) $M_J(m) \succ_m M_I(m)$ with respect to m 's true preference list in I , or (ii) m is single in M_I and matched in M_J , and $M_J(m)$ is acceptable to m in I . Let I' and J' , respectively, be the SMTI-instances constructed from I and J by breaking ties in the above mentioned manner. Then M_I and M_J are, respectively, the results of MGS applied to I' and J' . Since I' is the result of tie-breaking of I and m prefers M_J to M_I in I , m prefers M_J to M_I in I' . Note that, due to the tie-breaking rule, the preference lists of people except for m are same in I' and J' . This means that when MGS is used in SMTI, m can have a successful strategy in I' (i.e., to change his list to that of J'), contradicting man-strategy-proofness of MGS for SMTI (page 57 of [7]).

If we exchange the roles of men and women in S^* , we obtain a woman-strategy-proof 2-approximate-stable mechanism. ◀

We then show the negative part. We remark that ϵ is not necessarily a constant.

► **Lemma 6.** (1) *For any positive ϵ , there is no man-strategy-proof $(2 - \epsilon)$ -approximate-stable mechanism for MAX SMTI, even if ties appear in only women's preference lists. (2) For any positive ϵ , there is no woman-strategy-proof $(2 - \epsilon)$ -approximate-stable mechanism for MAX SMTI, even if ties appear in only men's preference lists.*

Proof. (1) Consider the instance I_1 given in Fig. 1, where m_3 's preference list is empty. It is straightforward to verify that I_1 has two stable matchings $M_1 = \{(m_1, w_1), (m_2, w_2)\}$ and $M_2 = \{(m_1, w_2), (m_2, w_3)\}$, both of which are of maximum size.

m_1 :	w_2	w_1	w_1 :	m_1
m_2 :	w_2	w_3	w_2 :	$(m_1 \quad m_2)$
m_3 :			w_3 :	m_2

■ **Figure 1** A MAX SMTI instance I_1 .

Let S be an arbitrary $(2 - \epsilon)$ -approximate-stable mechanism for MAX SMTI. Since S is a stable mechanism, it must output either M_1 or M_2 on I_1 . First suppose that it outputs M_1 . Let I'_1 be the instance obtained from I_1 by deleting w_1 from m_1 's preference list. Then since

M_2 is still a stable matching for I'_1 and S is a $(2 - \epsilon)$ -approximate-stable mechanism, S must output a stable matching of size 2. But since M_2 is now the only stable matching of size 2, S outputs M_2 on I'_1 . Thus m_1 can obtain a better partner by manipulating his preference list. On the other hand, suppose that S outputs M_2 on I_1 . Then let I''_1 be the instance obtained from I_1 by deleting w_3 from m_2 's preference list. By a similar argument, S must output M_1 on I''_1 and hence m_2 can obtain a better partner by manipulation. We have shown that, for any $(2 - \epsilon)$ -approximate-stable mechanism S , some man has a successful strategy in I_1 and hence S is not a man-strategy-proof mechanism.

(2) We use the instance I_2 given in Fig. 2, which is symmetric to I_1 . By the same argument as above, we can show that for any $(2 - \epsilon)$ -approximate-stable mechanism S , some woman has a successful strategy in I_2 and hence S is not a woman-strategy-proof mechanism.

m_1 :	w_1	w_1 :	m_2	m_1
m_2 :	$(w_1 \ w_2)$	w_2 :	m_2	m_3
m_3 :	w_2	w_3 :		

■ **Figure 2** A MAX SMTI instance I_2 . ◀

3 Results for MAX SMTI-1TM

Recall that MAX SMTI-1TM is a restriction of MAX SMTI where ties can appear in men's preference lists only. Then Corollary 3 is immediate from Lemma 5 and Lemma 6(2).

We then move to man-strategy-proofness and give a proof for Theorem 4. We start with the negative part:

► **Lemma 7.** *For any positive ϵ , there is no man-strategy-proof $(1.5 - \epsilon)$ -approximate-stable mechanism for MAX SMTI-1TM.*

Proof. The proof goes like that of Lemma 6. Consider the instance I_3 in Fig. 3. I_3 has four matchings of size 3, namely, $M_3 = \{(m_1, w_1), (m_2, w_2), (m_3, w_3)\}$, $M_4 = \{(m_1, w_1), (m_2, w_2), (m_3, w_4)\}$, $M_5 = \{(m_1, w_1), (m_2, w_3), (m_3, w_4)\}$, and $M_6 = \{(m_1, w_2), (m_2, w_3), (m_3, w_4)\}$. Among them, M_3 and M_6 are stable (M_4 is blocked by (m_3, w_3) and M_5 is blocked by (m_1, w_2)). Hence any $(1.5 - \epsilon)$ -approximate-stable mechanism outputs either M_3 or M_6 , since a stable matching of size 2 is not a $(1.5 - \epsilon)$ -approximate solution.

m_1 :	w_2	w_1	w_1 :	m_1	
m_2 :	$(w_2 \ w_3)$		w_2 :	m_2	m_1
m_3 :	w_3	w_4	w_3 :	m_2	m_3
m_4 :			w_4 :	m_3	

■ **Figure 3** A MAX SMTI-1TM instance I_3 .

Consider an arbitrary $(1.5 - \epsilon)$ -approximate-stable mechanism S for MAX SMTI-1TM, and suppose that S outputs M_3 on I_3 . Then if m_1 deletes w_1 from the list, M_6 is the unique maximum stable matching (of size 3); hence S must output M_6 and so m_1 can obtain a better partner w_2 . Similarly, if S outputs M_6 on I_3 , m_3 can force S to output M_3 by deleting w_4 from the list. In either case, some man has a successful strategy in I_3 and hence S is not a man-strategy-proof mechanism. ◀

Finally, we give a proof for the positive part, which is the main result of this paper.

► **Lemma 8.** *There exists a man-strategy-proof 1.5-approximate-stable mechanism for MAX SMTI-1TM.*

Proof. We give Algorithm 1 and show that it is a man-strategy-proof 1.5-approximate-stable mechanism by three subsequent lemmas (Lemmas 9–11). Algorithm 1 first translates an SMTI-1TM instance I to an SMI instance I' using Algorithm 2, then applies MGS to I' and obtains a matching M' , and finally constructs a matching M of I from M' . The new instance I' contains $2n$ men a_i and b_j ($1 \leq i \leq n$, $1 \leq j \leq n$) and $2n$ women s_j and t_j ($1 \leq j \leq n$) (lines 2 and 3 of Algorithm 2). It is important to note that a man a_i corresponds to a man m_i of I , while a man b_j and two women s_j and t_j correspond to a woman w_j of I . As will be seen later, b_j is definitely matched with s_j or t_j in M' , and the other woman (i.e., either s_j or t_j who is not matched with b_j) plays a role of woman w_j of I : If she is single in M' , then w_j is single in M . If she is matched with a_i in M' , then w_j is matched with m_i in M .

■ **Algorithm 1** An algorithm for MAX SMTI-1TM.

Input: An instance I for MAX SMTI-1TM.

Output: A matching M for I .

- 1: Construct an SMI instance I' from I using Algorithm 2.
 - 2: Apply MGS to I' and obtain a matching M' .
 - 3: Let $M := \{(m_i, w_j) \mid (a_i, s_j) \in M' \vee (a_i, t_j) \in M'\}$ and output M .
-

■ **Algorithm 2** Translating instances.

Input: An instance I for MAX SMTI-1TM.

Output: An instance I' for SMI.

- 1: Let X and Y be the sets of men and women of I , respectively.
- 2: Let $X' := \{a_i \mid m_i \in X\} \cup \{b_j \mid w_j \in Y\}$ be the set of men of I' .
- 3: Let $Y' := \{s_j \mid w_j \in Y\} \cup \{t_j \mid w_j \in Y\}$ be the set of women of I' .
- 4: Each a_i 's list is constructed as follows: Consider a tie $(w_{j_1} w_{j_2} \cdots w_{j_k})$ in m_i 's list in I . We assume without loss of generality that $j_1 < j_2 < \cdots < j_k$. (If not, just arrange the order, which does not change the instance.) Replace each tie $(w_{j_1} w_{j_2} \cdots w_{j_k})$ by a strict order of $2k$ women $t_{j_1} t_{j_2} \cdots t_{j_k} s_{j_1} s_{j_2} \cdots s_{j_k}$. A woman who is not included in a tie is considered as a tie of length one.
- 5: Each b_j 's list is defined as “ $b_j : s_j t_j$ ”.
- 6: For each j , let $P(w_j)$ be the list of w_j in I , and $Q(w_j)$ be the list obtained from $P(w_j)$ by replacing each man m_i by a_i . Then s_j and t_j 's lists are defined as follows:

$$\begin{array}{ll} s_j : & Q(w_j) \quad b_j \\ t_j : & b_j \quad Q(w_j) \end{array}$$

We briefly give a high-level idea behind Algorithm 1. Consider an application of MGS to I' at line 2. Since men's proposal order does not affect the outcome, it is convenient to first let b_j propose to his first choice woman s_j for each j . At this moment, there are n pairs (b_j, s_j) ($1 \leq j \leq n$). We regard this as an initial state, and as long as (b_j, s_j) is a pair, t_j acts as w_j . At some point, if s_j receives a proposal from some man a_i for the first time, s_j rejects b_j and b_j then proposes to his second choice woman t_j , which is accepted. We regard this as a change of the state, and the role of w_j is taken over to s_j . Once this happens, (b_j, t_j) remains a pair till the end of the algorithm. Recall that at line 4 of Algorithm 2, each man makes two copies of each tie. This is regarded as allowing a man to propose to woman w_j twice, first to t_j and second to s_j .

With these observations in mind, we can see that MGS for I' simulates the following GS-type algorithm for the original MAX SMTI instance I .

- Each free man proposes to a woman from the top of the list. When he encounters a tie T , he proposes to the women in T in a predetermined order (i.e., smaller index first). If he is rejected by all of them, he starts the second sequence of proposals to the women in T in the same order. If he is rejected by all the women in T again, then he proceeds to the next tie.
- Each woman's acceptance/rejection policy is as follows: If two proposals are first proposals, she respects her preference list. Similarly, if both are second proposals, she respects her preference list. If one is a first proposal and the other is a second proposal, she always chooses the second proposal (regardless of her list). Hence, once a woman receives a second proposal of some man, she never accepts a first proposal thereafter.

This algorithm achieves an approximation ratio of 1.5 for MAX SMTI, although we do not prove it here. A beneficial point of this algorithm is that a man's proposal order is predetermined and is not affected by other persons' states. As we explained in Sec. 1.3, absence of this property prevented existing algorithms from being man-strategy-proof.

The reason why we do not use this algorithm directly but translate it to an algorithm using MGS for SMI is to make the proof of man-strategy-proofness simpler; this translation allows us to attribute man-strategy-proofness of Algorithm 1 to that of MGS for SMI, as we did in the proof of Lemma 5.

Now we start formal proofs for the correctness.

► **Lemma 9.** *Algorithm 1 always outputs a stable matching.*

Proof. Let M be the output of Algorithm 1 and M' be the matching obtained at line 2 of Algorithm 1. We first show that M is a matching. Since M' is a matching, a_i appears at most once in M' , so m_i appears at most once in M . Observe that b_j is matched in M' , as otherwise (b_j, t_j) blocks M' , contradicting the stability of M' in I' . Hence at most one of s_j and t_j can be matched with a_i for some i , which implies that w_j appears at most once in M . Thus M is a matching.

We then show the stability of M . Since M' is the output of MGS, it is stable in I' . Now suppose that M is unstable in I and there is a blocking pair (m_i, w_j) for M . There are four cases:

Case (i): both m_i and w_j are single. Since m_i is single in M , line 3 of Algorithm 1 implies that a_i is single in M' . Since w_j is single in M , s_j is not matched in M' with anyone in $Q(w_j)$, i.e., s_j is single or matched with b_j . Note that (a_i, s_j) is a mutually acceptable pair because (m_i, w_j) is a blocking pair, and $a_i \succ_{s_j} b_j$ in I' by construction. Thus (a_i, s_j) blocks M' , a contradiction.

Case (ii): $w_j \succ_{m_i} M(m_i)$ and w_j is single. Let $M(m_i) = w_k$. Then, by construction of M , $M'(a_i)$ is either s_k or t_k . By construction of I' , $w_j \succ_{m_i} w_k$ implies both $s_j \succ_{a_i} s_k$ and $s_j \succ_{a_i} t_k$, and in either case we have that $s_j \succ_{a_i} M'(a_i)$ in I' . Since w_j is single in M , by the same argument as Case (i), s_j is either single or matched with b_j in M' . Hence (a_i, s_j) blocks M' .

Case (iii): m_i is single and $m_i \succ_{w_j} M(w_j)$. Since m_i is single in M , a_i is single in M' by the same argument as Case (i). Let $M(w_j) = m_k$. Then, by construction of M , either s_j or t_j is matched with a_k , and the other is matched with b_j since b_j can never be single as we have seen in an earlier stage of this proof. In particular, $M'(s_j)$ is either a_k or b_j . Note that $m_i \succ_{w_j} m_k$ in $P(w_j)$ implies $a_i \succ_{s_j} a_k$ in $Q(w_j)$, so in either case $a_i \succ_{s_j} M'(s_j)$ in I' due to the construction of s_j 's list. Therefore (a_i, s_j) blocks M' .

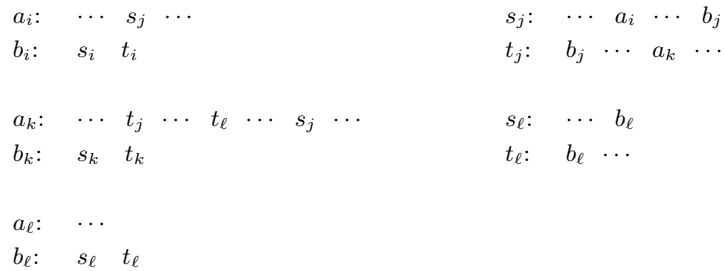
Case (iv): $w_j \succ_{m_i} M(m_i)$ and $m_i \succ_{w_j} M(w_j)$. By the same argument as Case (ii), we have that $s_j \succ_{a_i} M'(a_i)$ in I' . By the same argument as Case (iii), we have that $a_i \succ_{s_j} M'(s_j)$ in I' . Hence (a_i, s_j) blocks M' . ◀

► **Lemma 10.** *Algorithm 1 always outputs a 1.5-approximate solution.*

Proof. Let I be an input, M_{opt} be a maximum stable matching for I , and M be the output of Algorithm 1. We show that $\frac{|M_{opt}|}{|M|} \leq 1.5$. Let $G = (X \cup Y, E)$ be a bipartite (multi-)graph with vertex bipartition X and Y , where X corresponds to men and Y corresponds to women of I . The edge set E is a union of M and M_{opt} , that is, $(m_i, w_j) \in E$ if and only if (m_i, w_j) is a pair in M or M_{opt} . If (m_i, w_j) is a pair in both M and M_{opt} , then E contains two edges (m_i, w_j) , which constitute a “cycle” of length two. An edge in E corresponding to M (resp. M_{opt}) is called an M -edge (resp. M_{opt} -edge). Since the degree of each vertex of G is at most 2, each connected component of G is an isolated vertex, a cycle, or a path.

It is easy to see that G does not contain a single M_{opt} -edge as a connected component, since if such an edge (m_i, w_j) exists, then (m_i, w_j) is a blocking pair for M , contradicting the stability of M . In the following, we show that G does not contain, as a connected component, a path of length three $m_i - w_j - m_k - w_\ell$ such that (m_i, w_j) and (m_k, w_ℓ) are M_{opt} -edges and (m_k, w_j) is an M -edge. If this is true, then for any connected component C of G , the number of M -edges in C is at least two-thirds of the number of M_{opt} -edges in C , implying $\frac{|M_{opt}|}{|M|} \leq 1.5$.

Suppose that such a path exists. Note that m_i and w_ℓ are single in M . If $m_i \succ_{w_j} m_k$, then (m_i, w_j) blocks M . Since women’s preference lists do not contain ties, we have that $m_k \succ_{w_j} m_i$. If $w_\ell \succ_{m_k} w_j$, then (m_k, w_ℓ) blocks M . If $w_j \succ_{m_k} w_\ell$, then (m_k, w_j) blocks M_{opt} . Hence w_j and w_ℓ are tied in m_k ’s list. Then by construction of I' , (i) $t_\ell \succ_{a_k} s_j$. (Hereafter, referring to Fig. 4 would be helpful. Here, the order of t_j and t_ℓ in a_k ’s list is uncertain, i.e., it may be the opposite, but this order is not important in the rest of the proof.) Since w_ℓ is single in M , either s_ℓ or t_ℓ is single in M' . If s_ℓ is single in M' , then (b_ℓ, s_ℓ) blocks M' , a contradiction. Hence (ii) t_ℓ is single in M' . Since $M(m_k) = w_j$, either $M'(a_k) = s_j$ or $M'(a_k) = t_j$ holds. In the former case, (i) and (ii) above imply that (a_k, t_ℓ) blocks M' , so assume the latter, i.e., $M'(a_k) = t_j$. Recall from the proof of Lemma 9 that either s_j or t_j is matched with b_j in M' , so $M'(s_j) = b_j$. Since (m_i, w_j) is an acceptable pair in I , we have that $a_i \succ_{s_j} b_j$ due to the construction of s_j ’s list. Since m_i is single in M , a_i is single in M' . Hence (a_i, s_j) blocks M' , a contradiction. ◀



■ **Figure 4** A part of the preference lists of I' .

► **Lemma 11.** *Algorithm 1 is a man-strategy-proof mechanism.*

Proof. The proof is similar to that of Lemma 5. Suppose that Algorithm 1 is not a man-strategy-proof mechanism. Then there are MAX SMTI-1TM instances I and J and a man m_i having the following properties: I and J differ in only m_i ’s preference list, and m_i prefers

M_J to M_I , where M_I and M_J are the outputs of Algorithm 1 for I and J , respectively. Then either (i) $M_J(m_i) \succ_{m_i} M_I(m_i)$ in I , or (ii) m_i is single in M_I and $M_J(m_i)$ is acceptable to m_i in I .

Let I' and J' be the SMI-instances constructed by Algorithm 2. Since I and J differ in only m_i 's preference list, I' and J' differ in only a_i 's preference list. Let $M_{I'}$ and $M_{J'}$, respectively, be the outputs of MGS applied to I' and J' . In case of (i), we have that $M_{J'}(a_i) \succ_{a_i} M_{I'}(a_i)$ in I' , due to line 4 of Algorithm 2 and line 3 of Algorithm 1. In case of (ii), a_i is single in $M_{I'}$ because m_i is single in M_I , and $M_{J'}(a_i)$ is acceptable to a_i in I' because $M_J(m_i)$ is acceptable to m_i in I . This implies that a_i has a successful strategy in I' , contradicting man-strategy-proofness of MGS for SMI [7]. ◀

By Lemmas 9, 10, and 11, we can conclude that Algorithm 1 is a man-strategy-proof 1.5-approximate-stable mechanism for MAX SMTI-1TM. ◀

References

- 1 Atila Abdulkadiroğlu, Parag A. Pathak, and Alvin E. Roth. Strategy-proofness versus efficiency in matching with indifferences: Redesigning the NYC high school match. *American Economic Review*, 99(5):1954–1978, 2009. doi:10.1257/aer.99.5.1954.
- 2 L. E. Dubins and D. A. Freedman. Machiavelli and the Gale-Shapley Algorithm. *The American Mathematical Monthly*, 88(7):485–494, 1981. doi:10.1080/00029890.1981.11995301.
- 3 Shaddin Dughmi and Arpita Ghosh. Truthful assignment without money. In *Proceedings 11th ACM Conference on Electronic Commerce (EC-2010), Cambridge, Massachusetts, USA, June 7-11, 2010*, pages 325–334, 2010. doi:10.1145/1807342.1807394.
- 4 Aytekin Erdil and Haluk Ergin. What's the matter with tie-breaking? Improving efficiency in school choice. *American Economic Review*, 98(3):669–689, 2008. doi:10.1257/aer.98.3.669.
- 5 David Gale and Lloyd S. Shapley. College Admissions and the Stability of Marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962. doi:10.2307/2312726.
- 6 David Gale and Marilda Sotomayor. Some remarks on the stable matching problem. *Discrete Applied Mathematics*, 11(3):223–232, 1985. doi:10.1016/0166-218X(85)90074-5.
- 7 Dan Gusfield and Robert W. Irving. *The Stable marriage problem - structure and algorithms*. Foundations of computing series. MIT Press, 1989.
- 8 Magnús M. Halldórsson, Kazuo Iwama, Shuichi Miyazaki, and Hiroki Yanagisawa. Improved approximation results for the stable marriage problem. *ACM Transactions on Algorithms*, 3(3):30, 2007. doi:10.1145/1273340.1273346.
- 9 Robert W. Irving and David F. Manlove. Approximation algorithms for hard variants of the stable marriage and hospitals/residents problems. *Journal of Combinatorial Optimization*, 16(3):279–292, 2008. doi:10.1007/s10878-007-9133-x.
- 10 Kazuo Iwama, David F. Manlove, Shuichi Miyazaki, and Yasufumi Morita. Stable Marriage with Incomplete Lists and Ties. In *Automata, Languages and Programming, 26th International Colloquium, ICALP'99, Prague, Czech Republic, July 11-15, 1999, Proceedings*, pages 443–452, 1999. doi:10.1007/3-540-48523-6_41.
- 11 Zoltán Király. Linear Time Local Approximation Algorithm for Maximum Stable Marriage. *Algorithms*, 6(3):471–484, 2013. doi:10.3390/a6030471.
- 12 Piotr Krysta, David F. Manlove, Baharak Rastegari, and Jinshan Zhang. Size Versus Truthfulness in the House Allocation Problem. *Algorithmica*, 81(9):3422–3463, 2019. doi:10.1007/s00453-019-00584-7.
- 13 Chi-Kit Lam and C. Gregory Plaxton. A $(1 + 1/e)$ -Approximation Algorithm for Maximum Stable Matching with One-Sided Ties and Incomplete Lists. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2823–2840, 2019. doi:10.1137/1.9781611975482.175.

- 14 David F. Manlove. *Algorithmics of Matching Under Preferences*, volume 2 of *Series on Theoretical Computer Science*. WorldScientific, 2013. doi:10.1142/8591.
- 15 David F. Manlove, Robert W. Irving, Kazuo Iwama, Shuichi Miyazaki, and Yasufumi Morita. Hard variants of stable marriage. *Theoretical Computer Science*, 276(1-2):261–279, 2002. doi:10.1016/S0304-3975(01)00206-7.
- 16 Eric McDermid. A 3/2-Approximation Algorithm for General Stable Marriage. In *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I*, pages 689–700, 2009. doi:10.1007/978-3-642-02927-1_57.
- 17 Katarzyna E. Paluch. Faster and Simpler Approximation of Stable Matchings. *Algorithms*, 7(2):189–202, 2014. doi:10.3390/a7020189.
- 18 Alvin E. Roth. The Economics of Matching: Stability and Incentives. *Mathematics of Operations Research*, 7(4):617–628, November 1982. doi:10.1287/moor.7.4.617.
- 19 Alvin E. Roth. The Evolution of the Labor Market for Medical Interns and Residents: A Case Study in Game Theory. *Journal of Political Economy*, 92(6):991–1016, 1984. doi:10.1086/261272.
- 20 Alvin E. Roth. On the Allocation of Residents to Rural Hospitals: A General Property of Two-Sided Matching Markets. *Econometrica*, 54(2):425–27, 1986. doi:10.2307/1913160.
- 21 Alvin E. Roth and Marilda Sotomayor. *Two-Sided Matching: A Study in Game-Theoretic Modeling and Analysis*. Cambridge University Press, New York, 1990.
- 22 Hiroki Yanagisawa. Approximation algorithms for stable marriage problems. *PhD thesis, Kyoto University, Graduate School of Informatics*, 2007.

A The Man-Oriented Gale-Shapley Algorithm

During the course of the algorithm, each person takes one of two states “free” and “engaged”. At the beginning, everyone is free and the matching M is initialized to the empty set. At one step of the algorithm, an arbitrary free man m proposes to the top woman w in his current list. If w is free, then m and w are provisionally matched and (m, w) is added to M . If w is engaged and matched with m' , then w compares m and m' , takes the preferred one, and rejects the other. The rejected man deletes w from the list and becomes (or remains) free. When there is no free man, the matching M is output. The pseudo-code is given in Algorithm 3.

B Non-Strategy-Proofness of Existing 1.5-approximation Algorithms for MAX SMTI-1TM

Király [11] presented a 1.5-approximation algorithm for general MAX SMTI (i.e., ties can appear on both sides), which is named “New Algorithm”. We modify it in the following two respects.

1. Men’s proposals do not get into the second round.
2. When there is arbitrariness, the person with the smallest index is prioritized.

Ideas behind these modifications are as follows: For item 1, since there is no ties in women’s preference lists, executing the second round does not change the result. The role of item 2 is to make the algorithm deterministic, so that the output is a function of an input (as we did in the proof of Lemma 5). For completeness, we give a pseudo-code of the algorithm, denoted M-KNA to stand for “Modified Király’s New Algorithm”, in Algorithm 4.

Each person takes one of three states, “free”, “engaged”, and “semi-engaged”. Initially, all the persons are free. At lines 5, 10, and 14, man m proposes to woman w . Basically, the procedure is exactly the same as that of MGS. If w is free, then we let $M := M \cup \{(m, w)\}$

■ **Algorithm 3** The man-oriented Gale-Shapley algorithm.

```

1: Let  $M := \emptyset$  and all people be free.
2: while there is a free man whose preference list is non-empty do
3:   Let  $m$  be any free man.
4:   Let  $w$  be the woman at the top of  $m$ 's current list.
5:   if  $w$  is free then
6:     Let  $M := M \cup \{(m, w)\}$ , and  $m$  and  $w$  be engaged.
7:   end if
8:   if  $w$  is engaged then
9:     Let  $m'$  be  $w$ 's partner.
10:    if  $w$  prefers  $m'$  to  $m$  then
11:      Delete  $w$  from  $m$ 's list.
12:    else
13:      Let  $M := M \cup \{(m, w)\} \setminus \{(m', w)\}$ .
14:      Let  $m'$  be free and  $m$  be engaged.
15:      Delete  $w$  from  $m'$ 's list.
16:    end if
17:  end if
18: end while
19: Output  $M$ .

```

and both m and w be engaged (we say w *accepts* m). If w is engaged to m' (i.e., $(m', w) \in M$) and if $m \succ_w m'$, then we let $M := M \cup \{(m, w)\} \setminus \{(m', w)\}$, m be engaged, and m' be free. We also delete w from m' 's preference list (we say w *accepts* m and *rejects* m'). If w is engaged to m' and $m' \succ_w m$, then we delete w from m 's preference list (we say w *rejects* m).

There is an exception in the acceptance/rejection rule of a woman, when she receives the first and second proposals. This is actually the key for guaranteeing 1.5-approximation, but this rule is not used in the subsequent counter-example so we omit it here. Readers may consult to the original paper [11] for the full description of the algorithm.

It is already proved that the (original) Király's algorithm always outputs a stable matching which is a 1.5-approximate solution, and it is not hard to see that the same results hold for the above M-KNA for MAX SMTI-1TM. However, as the example in Figures 5 and 6 shows, it is not a man-strategy-proof mechanism.

m_1 :	w_2	w_1	w_1 :	m_2	m_4	m_1
m_2 :	$(w_1$	$w_3)$	w_2 :	m_4	m_1	
m_3 :	w_3		w_3 :	m_2	m_3	
m_4 :	w_1	w_2	w_4 :			

■ **Figure 5** A counter-example (true lists).

m_1 :	w_1	w_2	w_1 :	m_2	m_4	m_1
m_2 :	$(w_1$	$w_3)$	w_2 :	m_4	m_1	
m_3 :	w_3		w_3 :	m_2	m_3	
m_4 :	w_1	w_2	w_4 :			

■ **Figure 6** A counter-example (manipulated by m_1).

■ **Algorithm 4** Modified Király’s New Algorithm (M-KNA) [11].

```

1: Let  $M := \emptyset$  and all people be free.
2: while there is a free man whose preference list is non-empty do
3:   Among those men, let  $m$  be the one with the smallest index.
4:   if the top of  $m$ ’s current preference list consists of only one woman  $w$  then
5:     Let  $m$  propose to  $w$ .
6:   end if
7:   if the top of  $m$ ’s current preference list is a tie then
8:     if all the women in the tie are engaged then
9:       Among those women, let  $w$  be the one with the smallest index.
10:      Let  $m$  propose to  $w$ .
11:     end if
12:     if there is a free woman in the tie then
13:       Among those free women, let  $w$  be the one with the smallest index.
14:       Let  $m$  propose to  $w$ .
15:     end if
16:   end if
17: end while
18: Output  $M$ .

```

If M-KNA is applied to the true preference lists in Figure 5, the obtained matching is $\{(m_2, w_1), (m_3, w_3), (m_4, w_2)\}$. Suppose that m_1 flips the order of w_1 and w_2 (Figure 6). This time, M-KNA outputs $\{(m_1, w_2), (m_2, w_3), (m_4, w_1)\}$ and m_1 successfully obtains a partner w_2 . By proposing to w_1 first, m_1 is able to let m_2 propose to w_3 . This allows m_4 to obtain w_1 , which prevents m_4 from proposing to w_2 . This eventually makes it possible for m_1 to obtain w_2 .

We finally remark that the same example shows that the other two 1.5-approximation algorithms [16, 17] (with the tie-breaking rule 2 above) are not man-strategy-proof mechanisms either.

Online Multidimensional Packing Problems in the Random-Order Model

David Naori

Computer Science Department, Technion, 32000 Haifa, Israel
dnaori@cs.technion.ac.il

Danny Raz

Computer Science Department, Technion, 32000 Haifa, Israel
danny@cs.technion.ac.il

Abstract

We study online multidimensional variants of the generalized assignment problem which are used to model prominent real-world applications, such as the assignment of virtual machines with multiple resource requirements to physical infrastructure in cloud computing. These problems can be seen as an extension of the well known secretary problem and thus the standard online worst-case model cannot provide any performance guarantee. The prevailing model in this case is the random-order model, which provides a useful realistic and robust alternative. Using this model, we study the d -dimensional generalized assignment problem, where we introduce a novel technique that achieves an $O(d)$ -competitive algorithms and prove a matching lower bound of $\Omega(d)$. Furthermore, our algorithm improves upon the best-known competitive-ratio for the online (one-dimensional) generalized assignment problem and the online knapsack problem.

2012 ACM Subject Classification Theory of computation → Packing and covering problems; Theory of computation → Online algorithms

Keywords and phrases Random Order, Generalized Assignment Problem, Knapsack Problem, Multidimensional Packing, Secretary Problem

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.10

1 Introduction

Online multidimensional packing problems appear in a wide verity of real-world applications [8]. A recent relevant example is the assignment of virtual elements to the physical infrastructure in Network Function Virtualization (NFV) and cloud computing (see [20, 21] for example). Typically, in these problems, we are given a set of bins, each with a certain capacity profile, then, items arrive one-by-one in an online fashion, each with a certain size and profit. Upon each arrival, one has to decide immediately and irrevocably whether and where to pack the current item. The goal is to find an assignment that maximizes the total profit without exceeding the capacity of any bin. These problems can be viewed as generalizations of the well-known secretary problem, in which we have a single bin, and every secretary consumes the capacity of the whole bin (see [6] for a formal definition of the secretary problem).

The common way of analyzing online algorithms is to use the worst-case model, where an adversary picks an instance along with the order in which items are revealed to the online algorithm. Despite its prevalence in the analysis of online algorithms, this setting is too pessimistic for the problem at hand. Indeed, no online algorithm can achieve any non-trivial worst-case competitive-ratio, even for the simple case of the secretary problem, as shown by Aggarwal et al. [1]. A more realistic model is the random-order model in which the power of choosing the arrival order of items is taken away from the adversary, instead, the



© David Naori and Danny Raz;

licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 10; pp. 10:1–10:15

Leibniz International Proceedings in Informatics



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

arrival order is chosen uniformly at random. In this model, we say that an algorithm ALG is c -competitive if for every input instance \mathcal{I} it holds that, $c \cdot \mathbb{E}[\text{ALG}(\mathcal{I})] \geq \text{OPT}(\mathcal{I})$, where the expectation is taken over the random arrival orders and the randomness of the algorithm.¹

Kesselheim et al. [13, 14] generalized the known optimal algorithm for the secretary problem to various packing problems in the random-order model. The outline of the generic algorithmic framework is as follows: it starts with a sampling phase in which the algorithm only observes the arriving items. Then, at every subsequent online round, the algorithm computes a local solution for the sub-instance consisting of all the items that arrived so far. If the bin in which the current item is packed in this local solution has enough free capacity (i.e., an assignment of the current item in this bin is feasible) the algorithm carries it out, otherwise, it leaves the item unpacked. Using this framework, Kesselheim et al. [14] presented an algorithm for the online generalized assignment problem (GAP) with the best-known competitive-ratio (prior to this work).

In GAP we have a set of bins and a set of items. Each bin has a certain non-negative capacity and each item has several packing options, one for each bin. Each packing option is associated with a certain consumption from the capacity of the bin and a certain profit it provides. The goal is to pack the items in the bins where each item can be packed at most once, maximizing the total profit without exceeding the capacity of any bin. A major challenge in online packing problems, and online GAP in particular, is to handle both items with high consumption of resources compared to a bin capacity, as well as items with low consumption. Kesselheim et al. handle this challenge by partitioning a GAP instance into two sub-instances: the first contains all “heavy” packing options of items, that is, packing options that occupy more than half of a bin capacity, the second is the complementary sub-instance that contains all “light” packing options. Their algorithm makes an initial random choice to operate on one of the sub-instances exclusively. Although it achieves the best-known results, this behaviour is undesirable for most applications, since it always leaves one type of items unpacked.

We use a similar algorithmic framework to design an online algorithm for the d -dimensional generalization of online GAP, or online Vector Generalized Assignment Problem (VGAP), in which the capacity profile of each bin, as well as the consumption of items from each bin, is described by a d -dimensional vector. The goal remains to maximize the profit, while the capacity of each bin must not be exceeded in any of its d dimensions. To the best of our knowledge, this is the first time the online version of this problem is studied. Our algorithm offers a preferable behaviour and improves upon the best-known competitive-ratio for online GAP. To achieve this, we take a different approach to overcome the challenge: instead of limiting the algorithm to either “heavy” or “light” packing options, our algorithm considers them both. It operates in three phases: a sampling phase, a phase for “heavy” packing options, and a phase for “light” packing options. To compute the tentative assignments, our algorithm in the second phase uses maximum-weight bipartite matching, and in the third phase, it uses an optimal fractional solution for the LP-relaxation of the local problem, and randomized rounding.

We also apply our technique to the $\{0, 1\}$ -VGAP in which every packing option of an item in every dimension must consume either the whole capacity of the bin or non of it. In one-dimension this problem is identical to weighted bipartite matching. For $\{0, 1\}$ -VGAP we partition the instance by a different criterion: the number of non-zero entries in the consumption vector of a packing option.

¹ We follow the definition used in [4, 5, 13] although it is also common to refer to such algorithm as $1/c$ -competitive.

Another interesting special case of VGAP is the Vector Multiple Knapsack Problem (VMKP), in which all bins are identical, and the packing options of each item are identical for all bins. That is, regardless of the bin's identity, the item consumes the same amount of capacity and raises the same profit. For instances of VMKP with at least two bins, we describe a simpler algorithm that avoids partitioning the instance. Here, our algorithm uses a fractional solution for the LP-relaxation of the local problem only to make a binary decision whether to pack the current item or not. For the actual packing, it exploits the fact that all packing options are identical and uses greedy First Fit approach, typically used for the Bin Packing problem.

Finally, we prove a lower bound for the online vector knapsack problem in the random-order model, which also applies to VMKP and VGAP, and indicates that our algorithms are asymptotically optimal. This lower bound is inspired by the work of Babaioff et al. [5] on the matroid secretary problem, which is based solely on the inherent uncertainty due to the online nature of the problem without any complexity assumptions.

Our main contributions are:

1. We describe an algorithm for online VGAP with a competitive-ratio of $\sqrt[4]{e}(4d+2) \approx 5.14d + 2.57$, where d is the dimension. For the VMKP with at least two bins we describe a $(4d+2)$ -competitive algorithm. To the best of our knowledge, these problems are studied for the first time.
2. We prove a matching lower bound of $\Omega(d)$ which is valid both for VGAP and VMKP.
3. Our method improves upon the best-known competitive-ratio for (one-dimensional) GAP from 8.1 to 6.99 (which is also the best-known competitive-ratio for online knapsack).

2 Related Work

Online packing problems in the random-order model have been studied extensively in recent years, most of them are generalizations of the secretary problem which has an optimal e -competitive algorithm [10, 17]. An immediate generalization is the multiple-choice secretary problem, in which one is allowed to pick up to k secretaries. It was studied by Kleinberg [15], where he presented an asymptotically optimal $\frac{\sqrt{k}}{\sqrt{k-5}}$ -competitive algorithm. Another related problem is the weighted-matching problem which has an optimal e -competitive algorithm by Kesselheim et al. [13].

The online knapsack problem, which generalizes the multiple-secretary problem, was studied by Babaioff et al. [4] who presented an $10e$ -competitive algorithm. It was later improved by the work of Kesselheim et al. [14] on online GAP, which generalizes all of the above problems. They presented an 8.1-competitive algorithm which is the best-known competitive-ratio for online GAP and the online knapsack problem. Our result for VGAP improves on that.

In their work, Kesselheim et al. also studied the online packing LPs problem with column sparsity d . The general online packing LPs problem was studied before by [2, 11, 19]. In this problem, there is a set of resources and a set of requests. Each request has several options to be served and each option is associated with a profit and a certain demand from each resource. For column sparsity d , each request may have a demand from at most d of the resources. This problem generalizes VGAP studied in this paper, however, to the best of our knowledge, the only known competitive online algorithms for this problem are for the special case of $B \geq 2$, where B is the capacity ratio, i.e., the minimal ratio between the capacity of a resource and the maximum demand for this resource. For this case they presented an $O(d^{1/(B-1)})$ -competitive algorithm which in case $B = \Omega(\log d/\epsilon^2)$ is $(1 + \epsilon)$ -competitive.

Dean et al. [9] showed that under the assumption of $\text{NP} \neq \text{ZPP}$, the packing integer programs problem (PIP, also known as vector knapsack) which is a special case of VMKP, cannot be approximated in polynomial time to within $d^{1-\epsilon}$ for any $\epsilon > 0$ even in the offline settings. Under the same assumptions, Chekuri et al. [7] showed that the $\{0, 1\}$ -case cannot be approximated to within $d^{1/2-\epsilon}$ for any $\epsilon > 0$. Their results are also applicable for the offline VMKP, VGAP and the $\{0, 1\}$ -VGAP. By using the results of Zuckerman [22], the same hardness result can be proved under the weaker assumption of $\text{P} \neq \text{NP}$ instead. As opposed to these results, our lower bound holds with no complexity assumptions, and applies even for algorithms with unbounded computational power.

Some related problems have competitive algorithms in the worst-case model too. One example is the AdWords problem which is a special case of GAP in which the profit of each item is equal to its size. Under the assumption that items are small compared to the capacity of the bins, Mehta et al. [18] presented an optimal $\frac{e}{e-1}$ -competitive algorithm. Without this assumption, the best known competitive-ratio is 2 [16]. Another example is the online vector bin packing problem, in which items arrive one-by-one, and the goal is to pack them all in the minimum number of unit sized d -dimensional bins. This problem was studied by Garey et al. [12] who showed that the First Fit algorithm has a worst-case competitive-ratio of $(d + 0.7)$. More recently, Azar et al. [3] showed that this algorithm is asymptotically optimal by proving a lower bound of $\Omega(d^{1-\epsilon})$.

3 Vector Generalized Assignment Problem

In the d -dimensional *Generalized Assignment Problem* (VGAP), we have a set of m d -dimensional bins and a set of n d -dimensional items that may be packed in the bins. Each bin j has a capacity $\mathbf{b}_j = (b_j^1, \dots, b_j^d) \in \mathbb{R}_{\geq 0}^d$. Packing item i in bin j consumes an amount of $\mathbf{w}_{i,j} = (w_{i,j}^1, \dots, w_{i,j}^d) \in \mathbb{R}_{\geq 0}^d$ from bin's j capacity and provides a profit of $p_{i,j} \geq 0$. Each item may be packed in at most one of the bins and the capacity of each bin must not be exceeded in any of its d dimensions. The goal is to find a feasible packing that maximizes the total profit. We use the following LP-formulation:

$$\begin{aligned} \max \quad & \sum_{i \in [n], j \in [m]} p_{i,j} x_{i,j} \\ \text{s.t.} \quad & \sum_{i \in [n]} w_{i,j}^t x_{i,j} \leq b_j^t, \quad j \in [m], t \in [d] \\ & \sum_{j \in [m]} x_{i,j} \leq 1, \quad i \in [n] \\ & x_{i,j} \in \{0, 1\}, \quad i \in [n], j \in [m]. \end{aligned}$$

We consider the online version of the problem in which the set of bins and their capacities are initially known, as well as the total number of items n . The items, however, arrive one by one in a random order. When item i arrives, we learn its *packing options*, i.e., its consumption on every bin $\mathbf{w}_{i,1}, \dots, \mathbf{w}_{i,m}$ (which we also call the *weight vectors* of i) along with the corresponding profits $p_{i,1}, \dots, p_{i,m}$. After every arrival, an immediate and irrevocable decision must be made: Assign the item to one of the available bins or leave the item unpacked.

Our algorithm is based on the technique presented by the authors of [14] with several critical improvements (see Algorithm 1). We call the packing option of item i in bin j *light* if $w_{i,j}^t \leq b_j^t/2$, $\forall t \in [d]$, otherwise, we call it *heavy*. Given a GAP instance \mathcal{I} we partition it

into two sub-instances \mathcal{I}_{heavy} and \mathcal{I}_{light} , both consist of the original items and bins, however, \mathcal{I}_{heavy} consists only of the heavy packing options of every item, while \mathcal{I}_{light} consists only of the light ones. In contrast to the algorithm presented in [14] that makes a random choice whether to operate on \mathcal{I}_{heavy} or \mathcal{I}_{light} exclusively, our algorithm considers them both. It is based on the intuition that heavy options may need a chance to be packed first, since any other packing decision might prevent them from being packed, while light options are more likely to fit in. Our algorithm operates in three phases: the *sampling phase* in which it only observes the arriving items, the *heavy phase* in which it considers only heavy options, and the *light phase* in which it considers only light options. In the heavy phase, our algorithm uses a matching in a weighted bipartite graph to make packing decisions, to this end, given an instance \mathcal{I} we define a weighted bipartite graph $G(\mathcal{I}) = (L, R, E)$, where L is the set of items of \mathcal{I} , R is the set of bins of \mathcal{I} , and there exists an edge $(i, j) \in E$ of weight $p_{i,j}$ if item i can be packed in bin j (i.e., $w_{i,j}^t \leq b_j^t, \forall t \in [d]$). Each phase takes place in a continuous fraction of the online rounds. To partition the rounds into phases, we use two parameters q_1 and q_2 that will be defined thereafter. For convenience of presentation and analysis, we represent a packing by a set $P \subseteq [n] \times [m]$ such that $P = \{(i, j) : i \text{ is packed in bin } j\}$. We also define $p_{i,0} = 0, \forall i \in [n]$. For an instance \mathcal{I} and a subset S of its items, we denote by $\mathcal{I}|_S$ the sub-instance that consists only of the items in S .

■ **Algorithm 1** Online VGAP.

```

 $S_0 \leftarrow \emptyset, P_0 \leftarrow \emptyset;$ 
for each item  $i_\ell$  that arrives at round  $\ell$  do
   $S_\ell \leftarrow S_{\ell-1} \cup \{i_\ell\};$ 
  if  $\ell \leq q_1 n$  then                                     /* sampling phase */
    | continue to the next round;
  else if  $q_1 n + 1 \leq \ell \leq q_2 n$  then                 /* heavy phase */
    | Let  $x^{(\ell)}$  be a maximum-weight matching in  $G(\mathcal{I}_{heavy}|_{S_\ell});$ 
    | // compute a tentative assignment  $(i_\ell, j_\ell)$ 
    | if  $i_\ell$  is matched in  $x^{(\ell)}$  then
    | | Let  $j_\ell$  be the bin to which  $i_\ell$  is matched;
    | else
    | |  $j_\ell \leftarrow 0$ 
    | if  $j_\ell \neq 0$  and  $j_\ell$  is empty in  $P_{\ell-1}$  then
    | |  $P_\ell \leftarrow P_{\ell-1} \cup \{(i_\ell, j_\ell)\};$ 
  else // ( $\ell \geq q_2 n + 1$ )                               /* light phase */
    | Let  $x^{(\ell)}$  be an optimal fractional solution for the LP-relaxation of  $\mathcal{I}_{light}|_{S_\ell};$ 
    | // compute a tentative assignment  $(i_\ell, j_\ell)$  by randomized rounding
    | Choose bin  $j_\ell$  randomly where  $\Pr[j_\ell = j] = x_{i_\ell, j}^{(\ell)}$  and
    |  $\Pr[j_\ell = 0] = 1 - \sum_{j \in [m]} x_{i_\ell, j}^{(\ell)};$ 
    | if  $j_\ell \neq 0$  and  $P_{\ell-1} \cup \{(i_\ell, j_\ell)\}$  is feasible then
    | |  $P_\ell \leftarrow P_{\ell-1} \cup \{(i_\ell, j_\ell)\};$ 
return  $P_n$ 

```

We now analyze the performance of Algorithm 1. Let $\text{OPT}(\mathcal{I})$ and $\text{ALG}(\mathcal{I})$ denote the overall profit of the optimal packing and the overall profit of the packing produced by Algorithm 1 on instance \mathcal{I} respectively. Let R_ℓ denote the profit raised by the algorithm

at round ℓ . In Lemma 1 and Lemma 2 below, we bound the expected profit raised at each round of the heavy phase and the light phase respectively. Similar claims are presented in [13] and [14].

► **Lemma 1.** *For $q_1n + 1 \leq \ell \leq q_2n$, we have $\mathbb{E}[R_\ell] \geq \frac{q_1}{\ell-1} \cdot \frac{1}{d} \text{OPT}(\mathcal{I}_{heavy})$.*

Proof. Let x^* be an optimal solution for \mathcal{I}_{heavy} , hence, $p^T x^* = \text{OPT}(\mathcal{I}_{heavy})$, and let $x^*|_{S_\ell}$ denote the projection of x^* onto the set of items S_ℓ , i.e., $(x^*|_{S_\ell})_{i,j} = x_{i,j}^*$ if $i \in S_\ell$ and $(x^*|_{S_\ell})_{i,j} = 0$ otherwise. Observe (by the definition of heavy) that in $x^*|_{S_\ell}$ every bin holds at most d items. Let x_ℓ^* be the solution obtained from $x^*|_{S_\ell}$ by leaving only the most profitable item in each bin. We get $p^T x_\ell^* \geq \frac{1}{d} \cdot p^T (x^*|_{S_\ell})$. Also, since x_ℓ^* is a feasible matching in $G(\mathcal{I}_{heavy}|_{S_\ell})$, we have $p^T x^{(\ell)} \geq p^T x_\ell^* \geq \frac{1}{d} \cdot p^T (x^*|_{S_\ell})$. Now since $S_\ell \subseteq [n]$ is a uniformly random subset of size ℓ , we have $\mathbb{E}[p^T (x^*|_{S_\ell})] = \frac{\ell}{n} \cdot \text{OPT}(\mathcal{I}_{heavy})$. Also, i_ℓ can be viewed as a uniformly random item of S_ℓ , and since $x^{(\ell)}$ is a matching we have $\mathbb{E}[p_{i_\ell, j_\ell}] = \mathbb{E}\left[\sum_{j \in [m]} x_{i_\ell, j}^{(\ell)} p_{i_\ell, j}\right] = \frac{1}{\ell} \mathbb{E}[p^T x^{(\ell)}]$. Combining the results together, we get

$$\mathbb{E}[p_{i_\ell, j_\ell}] = \frac{1}{\ell} \mathbb{E}[p^T x^{(\ell)}] \geq \frac{1}{\ell} \mathbb{E}\left[\frac{1}{d} \cdot p^T (x^*|_{S_\ell})\right] = \frac{1}{n \cdot d} \text{OPT}(\mathcal{I}_{heavy}).$$

The above expectation is taken only over the random choice of the subset $S_\ell \subseteq [n]$ and the random choice of $i_\ell \in S_\ell$, while the arrival order of items in previous rounds is irrelevant. We now bound the probability of successful assignment over the random arrival order of previous items. The assignment is successful if no item is packed in j_ℓ in rounds $q_1n, \dots, \ell - 1$. At round $\ell - 1$ the algorithm uses a maximum-weight matching in $G(\mathcal{I}_{heavy}|_{S_{\ell-1}})$ to compute a tentative assignment $(i_{\ell-1}, j_{\ell-1})$. In that matching at most one item is matched to j_ℓ . Since $i_{\ell-1}$ is a uniformly random item of $S_{\ell-1}$, the probability that $i_{\ell-1}$ is matched to j_ℓ is at most $1/(\ell - 1)$ regardless of the arrival order of the items in rounds $1, \dots, \ell - 2$, hence, we can treat subsequent events as independent and repeat the argument inductively from $\ell - 1$ to $q_1n + 1$ to get that the probability of successful assignment is at least $\prod_{k=q_1n+1}^{\ell-1} (1 - \frac{1}{k}) = \frac{q_1n}{\ell-1}$. By combining the expected profit with the probability of successful assignment, we get the lemma. ◀

► **Lemma 2.** *For $\ell \geq q_2n + 1$, we have $\mathbb{E}[R_\ell] \geq \frac{q_1}{q_2} \left(1 - 2d \sum_{k=q_2n+1}^{\ell-1} \frac{1}{k}\right) \frac{1}{n} \text{OPT}(\mathcal{I}_{light})$.*

Proof. Let x^* be an optimal solution for \mathcal{I}_{light} . At round $\ell \geq q_2n + 1$ the algorithm uses randomized rounding to determine the tentative assignment of i_ℓ from the fractional LP-solution $x^{(\ell)}$, therefore, $\mathbb{E}[p_{i_\ell, j_\ell}] = \mathbb{E}\left[\sum_{j \in [m]} x_{i_\ell, j}^{(\ell)} p_{i_\ell, j}\right]$. Using this observation, we can now follow a similar argument to that in the proof of Lemma 1 and get that for $\ell \geq q_2n + 1$, we have $\mathbb{E}[p_{i_\ell, j_\ell}] = \frac{1}{\ell} \mathbb{E}[p^T x^{(\ell)}] \geq \frac{1}{\ell} \mathbb{E}[p^T (x^*|_{S_\ell})] = \frac{1}{n} \text{OPT}(\mathcal{I}_{light})$, where the expectation is taken only over the random choice of the subset $S_\ell \subseteq [n]$, the random choice of $i_\ell \in S_\ell$ and the internal randomness of the algorithm at round ℓ . Here too, we bound the probability of successful assignment over the random arrival order of previous items and the internal randomness of the algorithm in previous rounds. Let us denote by $c(j, t, \ell)$ the total consumption of tentative assignments to bin j in dimension t during the light phase and before round ℓ . At round ℓ , the algorithm considers only light options, therefore, the assignment of i_ℓ to j_ℓ must be successful if the following conditions hold: (1) no item was packed in j_ℓ during the heavy phase, and (2) for every dimension $t \in [d]$, $c(j_\ell, t, \ell) \leq b_{j_\ell}^t/2$. Let us denote event (1) by H_ℓ , and the events described in (2) by L_ℓ^t for every dimension $t \in [d]$. We now bound $\mathbb{E}[c(j_\ell, t, \ell)]$ for every $t \in [d]$. Fix $t \in [d]$, at round $k < \ell$ of the light phase, the algorithm computes a tentative assignment based on a fractional optimal

solution for the LP-relaxation of $\mathcal{I}_{light}|_{S_k}$. In that solution, the total consumption of bin j_ℓ in dimension t is at most $b_{j_\ell}^t$. Since i_k can be viewed as a uniformly random item of S_k , the expected consumption of i_k from j_ℓ in dimension t is at most $b_{j_\ell}^t/k$, where the expectation is taken over the choice of $i_k \in S_k$ and the internal randomness of the algorithm at round k . Therefore, it is independent of the arrival order of items in rounds $1, \dots, k-1$, and the internal randomness used in those rounds. Hence, $\mathbb{E}[c(j_\ell, t, \ell)] \leq \sum_{k=q_2n+1}^{\ell-1} b_{j_\ell}^t/k$. We have

$$\begin{aligned} \Pr \left[\bigwedge_{t=1}^d L_\ell^t \right] &= 1 - \Pr \left[\bigvee_{t=1}^d \neg L_\ell^t \right] \geq 1 - \sum_{t=1}^d \Pr [\neg L_\ell^t] \\ &\geq 1 - \sum_{t=1}^d \frac{\sum_{k=q_2n+1}^{\ell-1} b_{j_\ell}^t/k}{b_{j_\ell}^t/2} \geq 1 - 2d \sum_{k=q_2n+1}^{\ell-1} \frac{1}{k}. \end{aligned}$$

The first inequality is due to a union bound, and the second is due to Markov's inequality. Since this event is independent of the arrival order of items in the heavy phase, we can follow the argument from the proof of the previous lemma and get that the probability of successful assignment is at least

$$\Pr \left[H_\ell \wedge \bigwedge_{t=1}^d L_\ell^t \right] \geq \prod_{k=q_1n+1}^{q_2n} \left(1 - \frac{1}{k} \right) \left(1 - 2d \sum_{k=q_2n+1}^{\ell-1} \frac{1}{k} \right) = \frac{q_1}{q_2} \left(1 - 2d \sum_{k=q_2n+1}^{\ell-1} \frac{1}{k} \right).$$

We can now combine the results of the expected profit and the success probability to get the lemma. \blacktriangleleft

► **Theorem 3.** For $q_2 = 2d/(2d+1)$ and $q_1 = q_2/\sqrt[4]{e}$, Algorithm 1 is $\sqrt[4]{e}(4d+2)$ -competitive.

Proof. The overall profit of the algorithm can be written as $\mathbb{E}[\text{ALG}] = \sum_{\ell=1}^n \mathbb{E}[R_\ell]$. We sum over the profit raised in each phase separately. For the heavy phase we have

$$\begin{aligned} \sum_{\ell=q_1n+1}^{q_2n} \mathbb{E}[R_\ell] &\geq \sum_{\ell=q_1n+1}^{q_2n} \frac{q_1}{\ell-1} \cdot \frac{1}{d} \text{OPT}(\mathcal{I}_{heavy}) \\ &= \text{OPT}(\mathcal{I}_{heavy}) \frac{q_1}{d} \sum_{\ell=q_1n}^{q_2n-1} \frac{1}{\ell} \geq \text{OPT}(\mathcal{I}_{heavy}) \frac{q_1}{d} \ln \left(\frac{q_2}{q_1} \right). \end{aligned}$$

The first inequality follows from Lemma 1 and the second inequality is due to the fact that $\sum_{\ell=q_1n}^{q_2n-1} \frac{1}{\ell} \geq \int_{q_1n}^{q_2n} \frac{1}{x} dx = \ln \left(\frac{q_2}{q_1} \right)$. For the light phase we have

$$\begin{aligned} \sum_{\ell=q_2n+1}^n \mathbb{E}[R_\ell] &\geq \sum_{\ell=q_2n+1}^n \frac{q_1}{q_2} \left(1 - 2d \sum_{k=q_2n+1}^{\ell-1} \frac{1}{k} \right) \frac{1}{n} \text{OPT}(\mathcal{I}_{light}) \\ &= \frac{1}{n} \text{OPT}(\mathcal{I}_{light}) \frac{q_1}{q_2} \left((1-q_2)n - 2d \sum_{k=q_2n+1}^n \left(\frac{n}{k} - 1 \right) \right) \\ &\geq \text{OPT}(\mathcal{I}_{light}) \frac{q_1}{q_2} \left((2d+1)(1-q_2) - 2d \ln \left(\frac{1}{q_2} \right) \right). \end{aligned}$$

The first inequality is due to Lemma 2 and the second inequality follows from the fact that $\sum_{k=q_2n+1}^n \frac{1}{k} \leq \int_{q_2n}^n \frac{1}{x} dx = \ln\left(\frac{1}{q_2}\right)$. Overall we get

$$\begin{aligned} \mathbb{E}[\text{ALG}] &\geq \text{OPT}(\mathcal{I}_{heavy}) \frac{q_1}{d} \ln\left(\frac{q_2}{q_1}\right) \\ &\quad + \text{OPT}(\mathcal{I}_{light}) \frac{q_1}{q_2} \left((2d+1)(1-q_2) - 2d \ln\left(\frac{1}{q_2}\right) \right). \end{aligned} \quad (1)$$

For the parameters $q_2 = 2d/(2d+1)$, $q_1 = q_2/\sqrt[4]{e}$, we have

$$\frac{q_1}{d} \ln\left(\frac{q_2}{q_1}\right) = \frac{1}{4\sqrt[4]{e}} \frac{2}{2d+1} = \frac{1}{\sqrt[4]{e}(4d+2)}.$$

Using the fact that for $x \geq 0$, $\ln(1+x) \leq x - \frac{1}{2}x^2 + \frac{1}{3}x^3$, we have

$$\frac{q_1}{q_2} \left((2d+1)(1-q_2) - 2d \ln\left(\frac{1}{q_2}\right) \right) = \frac{1}{\sqrt[4]{e}} \left(1 - 2d \ln\left(1 + \frac{1}{2d}\right) \right) \geq \frac{1}{\sqrt[4]{e}} \left(\frac{1}{4d} - \frac{1}{12d^2} \right).$$

It can be easily verified that $(1/4d - 1/12d^2) \geq 1/(4d+2)$ for $d \geq 1$. Now since $\text{OPT}(\mathcal{I}_{heavy}) + \text{OPT}(\mathcal{I}_{light}) \geq \text{OPT}(\mathcal{I})$, we get

$$\mathbb{E}[\text{ALG}] \geq \frac{1}{\sqrt[4]{e}(4d+2)} (\text{OPT}(\mathcal{I}_{heavy}) + \text{OPT}(\mathcal{I}_{light})) \geq \frac{1}{\sqrt[4]{e}(4d+2)} \text{OPT}. \quad \blacktriangleleft$$

It is important to note that for $d = 1$, the competitive-ratio can be improved by choosing $q_1 = 0.5256$ and $q_2 = 0.69$. Setting these parameters in (1) shows that Algorithm 1 is 6.99-competitive for the (one-dimensional) generalized assignment problem, which improves upon the best-known competitive-ratio of 8.1 achieved by Kesselheim et al. [14].

► **Remark 4.** Algorithm 1 can easily be extended to the case where each item has $K \geq 1$ different packing options in each bin in a similar way to the algorithm of Kesselheim et al. [14]. Therefore, the general online packing LPs problem with n requests and m resources can be viewed as a special case of VGAP with one m -dimensional bin and n items.

3.1 The $\{0,1\}$ -VGAP

The $\{0,1\}$ -VGAP is a special case of VGAP in which the consumption of item i from bin j in dimension t is either 0 or the whole capacity of bin j in dimension t . By scaling, we can assume without loss of generality that $\mathbf{b}_j = \mathbf{1}$ for all $j \in [m]$, and $\mathbf{w}_{i,j} \in \{0,1\}^d$, $\forall i \in [n]$, $\forall j \in [m]$.² Note that for $d = 1$ the problem is identical to weighted bipartite matching.

As for the general VGAP, given an instance \mathcal{I} we partition it into two sub-instances, however, we make the partition according to the density of the weight vectors: we call the packing option of item i in bin j *dense* if $|\text{supp}(\mathbf{w}_{i,j})| \geq \sqrt{d}$, otherwise, we call it *sparse*.³ We denote by \mathcal{I}_{dense} the sub-instance that consists only of the dense packing options of every item, and by \mathcal{I}_{sparse} the complementary sub-instance that consists only of the sparse packing options.

Our algorithm for this case, which we call Algorithm 3, is based on the simple observation that in \mathcal{I}_{dense} at most \sqrt{d} items can be packed in every bin, therefore, a maximum weight matching in $G(\mathcal{I}_{dense})$ has a weight of at least $\text{OPT}(\mathcal{I}_{dense})/\sqrt{d}$. Algorithm 3 is almost

² $\mathbf{1}$ denotes the all 1's vector.

³ $\text{supp}(\cdot)$ denotes the set of indices of non-zero entries of a vector.

identical to Algorithm 1, the only difference is that \mathcal{I}_{heavy} and \mathcal{I}_{light} are replaced with \mathcal{I}_{dense} and \mathcal{I}_{sparse} respectively (for a full description see Appendix A.1). The parameters q_1 and q_2 are defined in the analysis. Due to lack of space, we only state the result of our analysis in Theorem 5 and give the full proof in Appendix A.2.

► **Theorem 5.** For $q_2 = \sqrt{d}/(\sqrt{d} + 1)$ and $q_1 = q_2/\sqrt{e}$, Algorithm 3 is $2\sqrt{e}(\sqrt{d} + 2)$ -competitive.

4 Vector Multiple Knapsack Problem

The *Vector Multiple Knapsack Problem* (VMKP) is a special case of VGAP in which all bins have a capacity of $\mathbf{1}$, every packing option of item i consumes the same amount of capacity $\mathbf{w}_i \in [0, 1]^d$ and provides the same profit $p_i \geq 0$, i.e., $\mathbf{w}_{i,j} = \mathbf{w}_i$, $p_{i,j} = p_i$, $\forall i \in [n]$, $\forall j \in [m]$. We study the case where there are at least two bins, i.e., $m \geq 2$. For this special case we present an online algorithm that improves upon the competitive-ratio of Algorithm 1.

■ **Algorithm 2** Online VMKP.

```

 $S_0 \leftarrow \emptyset, P_0 \leftarrow \emptyset;$ 
for each item  $i_\ell$  that arrives at round  $\ell$  do
     $S_\ell \leftarrow S_{\ell-1} \cup \{i_\ell\};$ 
    if  $\ell \leq qn$  then                                     /* sampling phase */
        | continue to the next round;
    else //  $\ell \geq qn + 1$                                    /* packing phase */
        | Let  $x^{(\ell)}$  be an optimal fractional solution for the LP-relaxation of  $\mathcal{I}|_{S_\ell}$ ;
        | Choose  $j_\ell$  randomly where  $\Pr[j_\ell = j] = x_{i_\ell, j}^{(\ell)}$  and  $\Pr[j_\ell = 0] = 1 - \sum_{j \in [m]} x_{i_\ell, j}^{(\ell)}$ ;
        |
        | // First Fit
        | Let  $B_\ell = \{j \in [m] : P_\ell \cup \{(i_\ell, j)\} \text{ is feasible}\};$ 
        | if  $j_\ell \neq 0$  and  $B_\ell \neq \emptyset$  then
        | |  $P_\ell \leftarrow P_{\ell-1} \cup \{(i_\ell, \min B_\ell)\};$ 
    return  $P_n$ 

```

Algorithm 2 consists of two phases: a sampling phase and a packing phase. The packing phase is similar to the light phase of Algorithm 1, however, instead of using the LP-solution to compute a tentative assignment, it uses it only to make a binary decision whether to pack the current item or not. Still, we keep a randomized rounding step similar to Algorithm 1 in order to use observations made in Section 3. For the actual packing, Algorithm 2 exploits the fact that all packing options are identical and uses the First Fit algorithm [12].

We now analyze the performance of Algorithm 2. First we prove a simple observation due to the nature of First Fit.

► **Lemma 6.** For $\ell \geq qn + 1$ and $m \geq 2$, if i_ℓ cannot be packed in any bin, then $\sum_{(i,j) \in P_{\ell-1}} \sum_{t=1}^d w_i^t \geq m/2$.

Proof. Let $u(j, t, \ell)$ denote the total consumption of bin j in dimension t before round ℓ . Since i_ℓ cannot be packed in any bin, there is at least one item packed in each bin. Consider any two bins $j' > j$, and let i_k be the first item that was packed in j' . i_k could not be packed in bin j , therefore, for some $t' \in [d]$ we have $u(j, t', k) + w_{i_k}^{t'} > 1$. Since

10:10 Online Multidimensional Packing Problems in the Random-Order Model

the consumption is non-decreasing and $u(j', t', \ell) \geq w_{i_k}^{t'}$ we have $u(j, t, \ell) + u(j', t', \ell) > 1$, therefore, $\sum_{t=1}^d u(j, t, \ell) + u(j', t, \ell) > 1$. By summing the last inequality for all consecutive pairs of bins $(j+1, j)$ as well as $(m, 1)$ we get $2 \sum_{j=1}^m \sum_{t=1}^d u(j, t, \ell) > m$ and hence the lemma. \blacktriangleleft

Next, we follow the method of the previous section to bound the expected profit of the algorithm at each round.

► **Lemma 7.** *For $\ell \geq qn + 1$ and $m \geq 2$, we have $\mathbb{E}[R_\ell] \geq \left(1 - 2d \sum_{k=qn+1}^{\ell-1} \frac{1}{k}\right) \frac{1}{n} \text{OPT}$.*

Proof. By following a similar argument to that in Lemma 1, we get $\mathbb{E}[p_{i_\ell, j_\ell}] \geq \frac{1}{n} \text{OPT}$ for $\ell \geq qn + 1$. We now bound the probability that $\sum_{(i,j) \in P_{\ell-1}} \sum_{t=1}^d w_i^t < m/2$, by Lemma 6, this is a sufficient condition for the assignment of i_ℓ to be successful. At round $k < \ell$ the algorithm computes a tentative assignment based on an optimal fractional solution $x^{(k)}$ for the LP-relaxation of $\mathcal{I}|_{S_k}$, therefore, we have $\sum_{i \in S_k} \sum_{t=1}^d \sum_{j=1}^m x_{i,j}^{(k)} w_i^t \leq dm$. Since i_k is a uniformly random item of S_k , we have $\mathbb{E} \left[\sum_{t=1}^d \sum_{j=1}^m x_{i_k, j}^{(k)} w_{i_k}^t \right] \leq dm/k$, hence,

$$\begin{aligned} \mathbb{E} \left[\sum_{(i,j) \in P_{\ell-1}} \sum_{t=1}^d w_i^t \right] &\leq \mathbb{E} \left[\sum_{k=qn+1}^{\ell-1} \sum_{t=1}^d w_{i_k}^t \sum_{j=1}^m x_{i_k, j}^{(k)} \right] \\ &= \sum_{k=qn+1}^{\ell-1} \mathbb{E} \left[\sum_{t=1}^d \sum_{j=1}^m x_{i_k, j}^{(k)} w_{i_k}^t \right] \leq \sum_{k=qn+1}^{\ell-1} \frac{dm}{k}. \end{aligned}$$

As before, we can now use Markov's inequality to bound the probability of successful assignment and get the lemma. \blacktriangleleft

► **Theorem 8.** *For $q = 2d/(2d+1)$, Algorithm 2 is $(4d+2)$ -competitive.*

Proof. By Lemma 7, the overall profit of the algorithm is bounded by

$$\mathbb{E}[\text{ALG}] \geq \sum_{\ell=qn+1}^n \left(1 - 2d \sum_{k=qn+1}^{\ell-1} \frac{1}{k} \right) \frac{1}{n} \text{OPT} \geq \left((2d+1)(1-q) - 2d \ln \left(\frac{1}{q} \right) \right) \text{OPT}.$$

This bound is maximized for $q = 2d/(2d+1)$, and for this choice of parameter, using similar arguments as in the proof of Theorem 3, we get

$$\begin{aligned} \mathbb{E}[\text{ALG}] &\geq \left(1 - 2d \ln \left(1 + \frac{1}{2d} \right) \right) \text{OPT} \\ &\geq \left(\frac{1}{4d} + \frac{1}{12d^2} \right) \text{OPT} \geq \left(\frac{1}{4d+2} \right) \text{OPT}. \end{aligned} \quad \blacktriangleleft$$

Note that by setting $d = 1$ in (2) we get that Algorithm 2 is 5.29-competitive for the (one-dimensional) multiple knapsack problem with at least two bins.

► **Remark 9.** For the special case of $d = 1$, Algorithm 2 can be implemented in a more efficient way: instead of solving an LP-relaxation at every round of the packing phase, we can obtain an optimal fractional solution by using a simple greedy algorithm.

► **Remark 10.** Algorithm 2 can be extended to the case of variable-sized squared bins, that is, to the case where $\mathbf{b}_j = \mathbf{1} \cdot b_j, \forall j \in [m]$, under the assumption that every item fits into every bin, i.e., $w_i^t \leq b_j \forall i \in [n], \forall j \in [m], \forall t \in [d]$, through sorting the bins by their capacity in a non-increasing order.

5 Lower bound

We now prove a lower bound of $\Omega(d)$ for the vector knapsack problem (VMKP with a single bin). Since it is a special case of VGAP, it shows our $O(d)$ -competitive algorithm for VGAP from Section 3 is asymptotically optimal. Note that our lower bound holds without any complexity assumptions. In particular, it also applies to algorithms with unbounded computational power. The proof is inspired by the work of Babaioff et al. [5].

We construct an instance for the d -dimensional knapsack problem consisting of one bin of capacity $\mathbf{1}$ and $n = \delta d^{(\delta+1)d+1}$ items, where $\delta \in \mathbb{N}_+$. The weight vectors of the items are the columns of the following $d \times d$ matrices:

$$A_j = (1 - \epsilon j d^j) \cdot I + \epsilon j d^{j-1} \cdot (\mathbf{1}\mathbf{1}^T - I), \quad \forall j \in [\delta d^{(\delta+1)d}].$$

Where I is the $d \times d$ identity matrix, and $\epsilon < 1/(2nd^n)$. By the choice of ϵ it holds that $\epsilon j d^j < 1/2, \forall j \in [\delta d^{(\delta+1)d}]$.

Observe that for every matrix A_j , all the items that correspond to its columns fit together in the bin, that is, $A_j \cdot \mathbf{1} \leq \mathbf{1}$. Also, every two columns of different matrices cannot be packed together. This is true because for any two matrices A_i, A_j where $i > j$, and any two columns $k, \ell \in [d]$, we have

$$(A_j)_{k,k} + (A_i)_{k,\ell} \geq (1 - \epsilon j d^j) + \epsilon i d^{i-1} \geq 1 - \epsilon j d^j + \epsilon (j+1) d^j > 1.$$

The first inequality follows from the fact that $\epsilon i d^{i-1} < (1 - \epsilon i d^i)$, and the second inequality follows from the fact that $i \geq j+1$. Every item is independently assigned a profit of 1 with probability $1/d^{\delta+1}$ and 0 with probability $1 - 1/d^{\delta+1}$.

► **Theorem 11.** *Any online algorithm produces a packing with expected profit of at most $(1 + \frac{1}{d^\delta})$, while $\text{OPT} = d$ with probability of at least $(1 - \frac{1}{e^\delta})$.*

Proof. Let us observe the first item that the online algorithm packs. It corresponds to a column of one matrix A_j . All items that correspond to columns of different matrices cannot be packed along with it. The only items that can be added to the packing are the remaining columns of A_j . There are less than d such items left, each has an expected profit of $1/d^{\delta+1}$. Since the first item has a profit of at most 1, the expected profit of the packing produced by the algorithm is at most $1 + 1/d^\delta$.

With regard to the optimal packing, for a given matrix A_ℓ , the probability that all items are of profit 1 is $1/d^{(\delta+1)d}$, therefore, the probability that all matrices are of weight less than d is $(1 - 1/d^{(\delta+1)d})^{d^{(\delta+1)d\delta}} \leq 1/e^\delta$. ◀

Note that Theorem 11 can be easily modified to apply to the case of two identical bins, thus, it shows that Algorithm 2 for VMKP with at least two bins is also asymptotically optimal.

6 Conclusions

In this paper, we presented simple, asymptotically optimal, online algorithms for multidimensional variants of the generalized assignment problem in the random-order model, which has vast implications for real-world applications, like resource allocation in cloud computing.

Our bounds for VGAP are translated to a matching lower and upper bounds of $\Omega(m)$ and $O(m)$ for the general online packing LPs problem (as mentioned in Remark 4, where m is the number of resources).

For the one-dimensional case, the best lower bound for the online GAP is derived from the lower bound for the secretary problem of e [6]. An interesting open question is to close the gap between e and the upper bound of 6.99 presented in this paper. It is also very interesting to understand whether the new theoretical algorithm provides practical value for cloud resource allocation, where the value of d is a small constant (2 or 3).

References

- 1 Gagan Aggarwal, Gagan Goel, Chinmay Karande, and Aranyak Mehta. Online Vertex-Weighted Bipartite Matching and Single-bid Budgeted Allocations. In *SODA*, pages 1253–1264. SIAM, 2011.
- 2 Shipra Agrawal, Zizhuo Wang, and Yinyu Ye. A Dynamic Near-Optimal Algorithm for Online Linear Programming. *Operations Research*, 62(4):876–890, 2014.
- 3 Yossi Azar, Ilan Reuven Cohen, Seny Kamara, and F. Bruce Shepherd. Tight bounds for online vector bin packing. In *STOC*, pages 961–970. ACM, 2013.
- 4 Moshe Babaioff, Nicole Immorlica, David Kempe, and Robert Kleinberg. A Knapsack Secretary Problem with Applications. In *APPROX-RANDOM*, volume 4627 of *Lecture Notes in Computer Science*, pages 16–28. Springer, 2007.
- 5 Moshe Babaioff, Nicole Immorlica, and Robert Kleinberg. Matroids, secretary problems, and online mechanisms. In *SODA*, pages 434–443. SIAM, 2007.
- 6 Niv Buchbinder, Kamal Jain, and Mohit Singh. Secretary Problems via Linear Programming. *Math. Oper. Res.*, 39(1):190–206, 2014.
- 7 Chandra Chekuri and Sanjeev Khanna. On Multi-Dimensional Packing Problems. In *SODA*, pages 185–194. ACM/SIAM, 1999.
- 8 Henrik I Christensen, Arindam Khan, Sebastian Pokutta, and Prasad Tetali. Approximation and online algorithms for multidimensional bin packing: A survey. *Computer Science Review*, 24:63–79, 2017.
- 9 Brian C. Dean, Michel X. Goemans, and Jan Vondrák. Adaptivity and approximation for stochastic packing problems. In *SODA*, pages 395–404. SIAM, 2005.
- 10 Eugene B Dynkin. The optimum choice of the instant for stopping a Markov process. *Soviet Mathematics*, 4:627–629, 1963.
- 11 Jon Feldman, Monika Henzinger, Nitish Korula, Vahab S. Mirrokni, and Clifford Stein. Online Stochastic Packing Applied to Display Ad Allocation. In *ESA (1)*, volume 6346 of *Lecture Notes in Computer Science*, pages 182–194. Springer, 2010.
- 12 Michael R Garey, Ronald L Graham, David S Johnson, and Andrew Chi-Chih Yao. Resource constrained scheduling as generalized bin packing. *Journal of Combinatorial Theory, Series A*, 21(3):257–298, 1976.
- 13 Thomas Kesselheim, Klaus Radke, Andreas Tönnis, and Berthold Vöcking. An Optimal Online Algorithm for Weighted Bipartite Matching and Extensions to Combinatorial Auctions. In *ESA*, volume 8125 of *Lecture Notes in Computer Science*, pages 589–600. Springer, 2013.
- 14 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.
- 15 Robert D. Kleinberg. A multiple-choice secretary algorithm with applications to online auctions. In *SODA*, pages 630–631. SIAM, 2005.
- 16 Benny Lehmann, Daniel J. Lehmann, and Noam Nisan. Combinatorial auctions with decreasing marginal utilities. *Games and Economic Behavior*, 55(2):270–296, 2006.
- 17 Denis V Lindley. Dynamic programming and decision theory. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 10(1):39–51, 1961.
- 18 Aranyak Mehta, Amin Saberi, Umesh V. Vazirani, and Vijay V. Vazirani. AdWords and generalized online matching. *J. ACM*, 54(5):22, 2007.
- 19 Marco Molinaro and R. Ravi. Geometry of Online Packing Linear Programs. In *ICALP (1)*, volume 7391 of *Lecture Notes in Computer Science*, pages 701–713. Springer, 2012.

- 20 Danny Raz, Itai Segall, and Maayan Goldstein. Multidimensional resource allocation in practice. In *Proceedings of the 10th ACM International Systems and Storage Conference*, page 1. ACM, 2017.
- 21 Lei Shi, Bernard Butler, Dmitri Botvich, and Brendan Jennings. Provisioning of requests for virtual machine sets with placement constraints in IaaS clouds. In *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, pages 499–505. IEEE, 2013.
- 22 David Zuckerman. Linear Degree Extractors and the Inapproximability of Max Clique and Chromatic Number. *Theory of Computing*, 3(1):103–128, 2007.

A Omitted Details

A.1 Description of Algorithm 3

■ **Algorithm 3** Online $\{0,1\}$ -VGAP.

```

 $S_0 \leftarrow \emptyset, P_0 \leftarrow \emptyset;$ 
for each item  $i_\ell$  that arrives at round  $\ell$  do
   $S_\ell \leftarrow S_{\ell-1} \cup \{i_\ell\};$ 
  if  $\ell \leq q_1 n$  then /* sampling phase */
    | continue to the next round;
  else if  $q_1 n + 1 \leq \ell \leq q_2 n$  then /* dense phase */
    | Let  $x^{(\ell)}$  be a maximum-weight matching in  $G(\mathcal{I}_{dense}|_{S_\ell});$ 
    | // compute a tentative assignment  $(i_\ell, j_\ell)$ 
    | if  $i_\ell$  is matched in  $x^{(\ell)}$  then
    | | Let  $j_\ell$  be the bin to which  $i_\ell$  is matched;
    | else
    | |  $j_\ell \leftarrow 0$ 
    | if  $j_\ell \neq 0$  and  $j_\ell$  is empty in  $P_{\ell-1}$  then
    | |  $P_\ell \leftarrow P_{\ell-1} \cup \{(i_\ell, j_\ell)\};$ 
  else //  $(\ell \geq q_2 n + 1)$  /* sparse phase */
    | Let  $x^{(\ell)}$  be an optimal fractional solution for the LP-relaxation of  $\mathcal{I}_{sparse}|_{S_\ell};$ 
    | // compute a tentative assignment  $(i_\ell, j_\ell)$  by randomized rounding
    | Choose bin  $j_\ell$  randomly where  $\Pr[j_\ell = j] = x_{i_\ell, j}^{(\ell)}$  and
    |  $\Pr[j_\ell = 0] = 1 - \sum_{j \in [m]} x_{i_\ell, j}^{(\ell)};$ 
    | if  $j_\ell \neq 0$  and  $P_{\ell-1} \cup \{(i_\ell, j_\ell)\}$  is feasible then
    | |  $P_\ell \leftarrow P_{\ell-1} \cup \{(i_\ell, j_\ell)\};$ 
  return  $P_n$ 

```

A.2 Proof of Theorem 5

Proof. To prove the competitive-ratio of Algorithm 3, we bound the expected profit of the algorithm in round separately. In Lemma 12 below, we bound the expected profit in each round of the dense phase, and in Lemma 13, we bound the expected profit in each round of the sparse phase. Then we sum over the expected profit in both phases to get the total profit of the algorithm.

10:14 Online Multidimensional Packing Problems in the Random-Order Model

► **Lemma 12.** For $q_1n + 1 \leq \ell \leq q_2n$, we have $\mathbb{E}[R_\ell] \geq \frac{q_1}{\ell-1} \cdot \frac{1}{\sqrt{d}} \text{OPT}(\mathcal{I}_{dense})$.

The proof is similar to the proof of Lemma 1 by using the observation that in \mathcal{I}_{dense} at most \sqrt{d} items can be packed in every bin, therefore, a maximum weight matching in $G(\mathcal{I}_{dense})$ has a weight of at least $\text{OPT}(\mathcal{I}_{dense})/\sqrt{d}$.

► **Lemma 13.** For $q_1n + 1 \leq \ell \leq q_2n$, we have

$$\mathbb{E}[R_\ell] \geq \frac{q_1}{q_2} \left(1 - \sqrt{d} \sum_{k=q_2n+1}^{\ell-1} \frac{1}{k} \right) \frac{1}{n} \text{OPT}(\mathcal{I}_{sparse}).$$

Proof. As in Lemma 2, we have $\mathbb{E}[p_{i_\ell, j_\ell}] = \frac{1}{n} \text{OPT}(\mathcal{I}_{sparse})$ where the expectation is taken only over the random choice of the subset $S_\ell \subseteq [n]$, the random choice of $i_\ell \in S_\ell$ and the internal randomness of the algorithm at round ℓ . Once again we bound the probability of successful assignment over the random arrival order of previous items and the internal randomness of the algorithm in previous rounds. The assignment of i_ℓ to j_ℓ must be successful if the following conditions hold: (1) no item was packed in j_ℓ during the dense phase, and (2) no tentative assignments from previous rounds of the sparse phase occupy the entries in $\text{supp}(\mathbf{w}_{i_\ell, j_\ell})$ of j_ℓ . Let us denote event (1) by H_ℓ and the event described in (2) by L_ℓ . At round $q_2n \leq k \leq \ell$ the algorithm uses an optimal fractional solution $x^{(k)}$ for the LP-relaxation on \mathcal{I}_{S_k} to compute a tentative assignment (i_k, j_k) . In that solution we have $\sum_{i \in S_k} x_{i, j_\ell}^{(k)} w_{i, j_\ell}^t \leq 1, \forall t \in [d]$. Observe that by the randomized rounding at round k and the fact that $w_{i_k, j_\ell}^t \in \{0, 1\}$, the probability that the tentative assignment of i_k uses dimension t in j_ℓ is given by $\sum_{i \in S_k} \Pr[j_k = j_\ell \wedge w_{i, j_k}^t = 1 | i_k = i] \cdot \Pr[i_k = i] = \frac{1}{k} \sum_{i \in S_k} x_{i, j_\ell}^{(k)} w_{i, j_\ell}^t \leq \frac{1}{k}$. Using a union bound, since $|\text{supp}(\mathbf{w}_{i_\ell, j_\ell})| \leq \sqrt{d}$, the probability that i_k blocks i_ℓ from being packed is at most \sqrt{d}/k . Applying a union bound once again over all previous rounds of the sparse phase, we get $\Pr[L_\ell] \geq 1 - \sum_{k=q_2n+1}^{\ell-1} \sqrt{d}/k$. From here on we can follow a similar argument as in the proof of Lemma 2 and get that the probability of successful assignment is at least

$$\Pr[H_\ell \wedge L_\ell] \geq \prod_{k=q_1n+1}^{q_2n} \left(1 - \frac{1}{k} \right) \left(1 - \sum_{k=q_2n+1}^{\ell-1} \frac{\sqrt{d}}{k} \right) = \frac{q_1}{q_2} \left(1 - \sqrt{d} \sum_{k=q_2n+1}^{\ell-1} \frac{1}{k} \right).$$

Overall, we get the lemma. ◀

By using Lemma 12 and Lemma 13 to sum over the profit raised in each phase, we get

$$\begin{aligned} \mathbb{E}[\text{ALG}] &\geq \text{OPT}(\mathcal{I}_{dense}) \frac{q_1}{\sqrt{d}} \ln \left(\frac{q_2}{q_1} \right) \\ &\quad + \text{OPT}(\mathcal{I}_{sparse}) \frac{q_1}{q_2} \left((\sqrt{d} + 1)(1 - q_2) - \sqrt{d} \ln \left(\frac{1}{q_2} \right) \right). \end{aligned}$$

Setting $q_2 = \frac{\sqrt{d}}{\sqrt{d}+1}$, $q_1 = q_2/\sqrt{e}$, we get

$$\begin{aligned} \mathbb{E}[\text{ALG}] &\geq \text{OPT}(\mathcal{I}_{dense}) \frac{1}{2\sqrt{e}(\sqrt{d}+1)} \\ &\quad + \text{OPT}(\mathcal{I}_{sparse}) \frac{1}{\sqrt{e}} \left(1 - \sqrt{d} \ln \left(1 + \frac{1}{\sqrt{d}} \right) \right). \end{aligned} \tag{3}$$

To bound the second term we use the fact that for $x \geq 0$, $\ln(1+x) \leq x - \frac{1}{2}x^2 + \frac{1}{3}x^3$ and get

$$\frac{1}{\sqrt{e}} \left(1 - \sqrt{d} \ln \left(1 + \frac{1}{\sqrt{d}} \right) \right) \geq \frac{1}{\sqrt{e}} \left(\frac{1}{2\sqrt{d}} - \frac{1}{3d} \right) \geq \frac{1}{2\sqrt{e}(\sqrt{d}+1)}. \quad (4)$$

The second inequality can be easily verified to hold for $d \geq 1$. By substituting (4) in Equation (3) and using the fact that $\text{OPT}(\mathcal{I}_{dense}) + \text{OPT}(\mathcal{I}_{sparse}) \geq \text{OPT}(\mathcal{I})$, we get the theorem. \blacktriangleleft

Approximate Euclidean Shortest Paths in Polygonal Domains

R. Inkulu

Department of Computer Science & Engineering, IIT Guwahati, India

<http://www.iitg.ac.in/rinkulu/>

rinkulu@iitg.ac.in

Sanjiv Kapoor

Department of Computer Science & Engineering, IIT Chicago, USA

<http://www.cs.iit.edu/~kapoor/>

kapoor@iit.edu

Abstract

Given a set \mathcal{P} of h pairwise disjoint simple polygonal obstacles in \mathbb{R}^2 defined with n vertices, we compute a sketch Ω of \mathcal{P} whose size is independent of n , depending only on h and the input parameter ϵ . We utilize Ω to compute a $(1 + \epsilon)$ -approximate geodesic shortest path between the two given points in $O(n + h((\lg n) + (\lg h)^{1+\delta} + (\frac{1}{\epsilon} \lg \frac{h}{\epsilon})))$ time. Here, ϵ is a user parameter, and δ is a small positive constant (resulting from the time for triangulating the free space of \mathcal{P} using the algorithm in [3]). Moreover, we devise a $(2 + \epsilon)$ -approximation algorithm to answer two-point Euclidean distance queries for the case of convex polygonal obstacles.

2012 ACM Subject Classification Theory of computation \rightarrow Computational geometry

Keywords and phrases Computational Geometry, Geometric Shortest Paths, Approximation Algorithms

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.11

Related Version A full version of the paper is available at <https://arxiv.org/abs/1506.01769>.

Funding R. Inkulu: This research is supported in part by SERB MATRICS grant MTR/2017/000474.

1 Introduction

For any set \mathcal{Q} of pairwise-disjoint simple polygonal obstacles in \mathbb{R}^2 , the free space $\mathcal{F}(\mathcal{Q})$ is the closure of \mathbb{R}^2 without the union of the interior of all the polygons in \mathcal{Q} . Given a set $\mathcal{P} = \{P_1, P_2, \dots, P_h\}$ of pairwise-disjoint simple polygonal obstacles in \mathbb{R}^2 and two points s and t in $\mathcal{F}(\mathcal{P})$, the *Euclidean shortest path finding problem* seeks to compute a shortest path between s and t that lies in $\mathcal{F}(\mathcal{P})$. This problem is well-known in the computational geometry community. Mitchell [10, 27] provides an extensive survey of research accomplished in determining shortest paths in polygonal and polyhedral domains. The problem of finding shortest paths in graphs is quite popular and considered to be fundamental. Especially, several algorithms for efficiently computing single-source shortest paths and all-pairs shortest paths are presented in Cormen et al. [9] and Kleinberg and Tardos [25]) texts. And, the algorithms for approximate shortest paths are surveyed in [28]. In the following, we assume that n vertices together define the h polygonal obstacles of \mathcal{P} .

Given a polygonal domain \mathcal{P} as input, the following are three well-known variants of the Euclidean shortest path finding problem: (i) both s and t are given as input with \mathcal{P} , (ii) only s is provided as input with \mathcal{P} , and (iii) neither s nor t is given as input. The type (i) problem is a single-shot problem and involves no preprocessing. The preprocessing phase of the algorithm for a type (ii) problem constructs a shortest path map with s as the source so that a shortest path between s and any given query point t can be found efficiently. In the



© R. Inkulu and Sanjiv Kapoor;

licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 11; pp. 11:1–11:17

Leibniz International Proceedings in Informatics



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

third variation, which is known as a two-point shortest path query problem, the polygonal domain \mathcal{P} is preprocessed to construct data structures that facilitate in answering shortest path queries between any given pair of query points s and t .

In solving a type (i) or type (ii) problem, there are two fundamentally different approaches: the visibility graph method (see Ghosh [14] for both the survey and details of various visibility algorithms) and the continuous Dijkstra (wavefront propagation) method. The visibility graph method [5, 21, 22, 29] is based on constructing a graph G , termed visibility graph, whose nodes are the vertices of the obstacles (together with s and t) and edges are the pairs of mutually visible vertices. Once the visibility graph G is available, a shortest path between s and t in G is found using Dijkstra's algorithm. As the number of edges in the visibility graph is $O(n^2)$, this method has worst-case quadratic time complexity. In the continuous Dijkstra approach [17, 18, 19, 26], a wavefront is expanded from s till it reaches t . In specific, for the case of polygonal obstacles in plane, Hershberger and Suri devised an algorithm in [17] which computes a shortest path in $O(n \lg n)$ time and the algorithm in [18] (which extends the algorithm by Kapoor [19]) by Inkulu, Kapoor, and Maheshwari computes a shortest path in $O(n + h((\lg h)^\delta + (\lg n)(\lg h)))$ time. Here, δ is a small positive constant (resulting from the time for triangulating the $\mathcal{F}(\mathcal{P})$ using the algorithm in [3]). The continuous Dijkstra method typically constructs a shortest path map with respect to s so that for any query point t , a shortest path from s to t can be found efficiently.

The two-point shortest path query problem within a given simple polygon was addressed by Guibas and Hershberger [15]. It preprocessed the simple polygon in $O(n)$ time and constructed a data structure of size $O(n)$ and answers two-point shortest distance queries in $O(\lg n)$ time. Exact two-point shortest path queries in the polygonal domain were explored by Chiang and Mitchell [7]. One of the algorithms in [7] constructs data structures of size $O(n^5)$ and answers the query for any two-point distance in $O(h + \lg n)$ worst-case time. And, another algorithm in [7] builds data structures of size $O(n + h^5)$ and outputs any two-point distance query in $O(h \lg n)$ time. In both of these algorithms, a shortest path itself is found in additional time $O(k)$, where k is the number of edges in the output path. Guo et al. [16] preprocessed $\mathcal{F}(\mathcal{P})$ in $O(n^2 \lg n)$ time to compute data structures of size $O(n^2)$ for answering two-point distance queries for any given pair of query points in $O(h \lg n)$ time.

Because of the difficulty of exact two-point queries in polygonal domains, various approximation algorithms were devised. Clarkson first made such an attempt in [8]. Chen [4] used the techniques from [8] in constructing data structures of size $O(n \lg n + \frac{n}{\epsilon})$ in $o(n^{3/2}) + O(\frac{n}{\epsilon} \lg n)$ time to support $(6 + \epsilon)$ -approximate two-point distance queries in $O(\frac{1}{\epsilon} \lg n + \frac{1}{\epsilon^2})$ time, and a shortest path in additional $O(L)$ time, where L is the number of edges of the output path. Arikati et al. [2] devised a family of algorithms to answer two-point approximate shortest path queries. Their first algorithm outputs a $(\sqrt{2} + \epsilon)$ -approximate distance; depending on a parameter $1 \leq r \leq n$, in the worst-case, either the preprocessed data structures of this algorithm take $O(n^2)$ space or the query time is $O(\sqrt{n})$. Their second algorithm takes $O(n)$ query time to report the distance. The stretch of the third and fourth algorithms proposed in [2] are respectively $(2\sqrt{2} + \epsilon)$ and $(3\sqrt{2} + \epsilon)$. Agarwal et al. [1] computes a $(1 + \epsilon)$ -approximate geodesic shortest path in $O(n + \frac{h}{\sqrt{\epsilon}} \lg(\frac{h}{\epsilon}))$ time when the obstacles are convex.

Throughout this paper, to distinguish graph vertices from the vertices of the polygonal domain, we refer to vertices of a graph as nodes. The Euclidean distance between any two points p and q is denoted with $\|pq\|$. The obstacle-avoiding geodesic Euclidean shortest path distance between any two points p, q amid a set \mathcal{Q} of obstacles is denoted with $dist_{\mathcal{Q}}(p, q)$. The (shortest) distance between two nodes s and t in a graph G is denoted with $dist_G(s, t)$. Unless specified otherwise, distance is measured in Euclidean metric. We denote both the

convex hull of a set R of points and the convex hull of a simple polygon R with $CH(R)$. Let r' and r'' be two rays with origin at p . Let \vec{v}_1 and \vec{v}_2 be the unit vectors along the rays r' and r'' respectively. A *cone* $C_p(r', r'')$ is the set of points defined by rays r' and r'' such that a point $q \in C_p(r', r'')$ if and only if q can be expressed as a convex combination of the vectors \vec{v}_1 and \vec{v}_2 with positive coefficients. When the rays are evident from the context, we denote the cone with C_p . The counterclockwise angle from the positive x-axis to the line that bisects the cone angle of C_p is termed as the *orientation of the cone* C_p .

Our contributions

First, we describe the algorithm for the case in which \mathcal{P} comprises convex polygonal obstacles. We compute a sketch Ω from the polygonal domain \mathcal{P} . Essentially, each convex polygonal obstacle P_i in \mathcal{P} is approximated with another convex polygonal obstacle whose complexity depends only on the input parameter ϵ ; significantly, the size of the approximated polygon is independent of the size of P_i . In specific, when \mathcal{P} is comprised of h convex polygonal obstacles, the sketch Ω is comprised of h convex polygonal obstacles: for each $1 \leq i \leq h$, the convex polygon $P_i \in \mathcal{P}$ is approximated with another convex polygon $Q_i \in \Omega$. For each $P_i \in \mathcal{P}$, we identify a coreset S_i of vertices of P_i and form the core-polygon $Q_i \in \Omega$ using S_i . When P_i is convex, the corresponding core-polygon Q_i obtained through this procedure is convex; and, $Q_i \subseteq P_i$. Like in [1], the combinatorial complexity of Ω is independent of n ; it depends only on h and the input parameter ϵ . For two points $s, t \in \mathcal{F}(\mathcal{P})$, we compute an approximate Euclidean shortest path between s and t in $\mathcal{F}(\Omega)$ using an algorithm that is a variant of [8]. From this path, we compute a path R in $\mathcal{F}(\mathcal{P})$ and show that R is a $(1 + \epsilon)$ -approximate Euclidean shortest path between s and t amid polygonal obstacles in \mathcal{P} . When the obstacles in \mathcal{P} are not necessarily convex, we compute the sketch of \mathcal{P} using the convex chains (that bound the obstacles) as well as the corridor paths that result from the hourglass decomposition [19, 20, 22] of $\mathcal{F}(\mathcal{P})$. The main contributions and the major advantages in our approach are described in the following:

- When \mathcal{P} is comprised of disjoint simple polygonal obstacles, we compute a $(1 + \epsilon)$ -approximate geodesic Euclidean shortest path between the two given points belonging to $\mathcal{F}(\mathcal{P})$ in $O(n + h((\lg n) + (\lg h)^{1+\delta} + \frac{1}{\epsilon} \lg \frac{h}{\epsilon}))$ time. Here, δ is a small positive constant resulting from the triangulation of the free space using the algorithm from [3]. (Refer to Theorem 9.) Agarwal et al. [1] compute a $(1 + \epsilon)$ -approximate geodesic shortest path in $O(n + \frac{h}{\sqrt{\epsilon}} \lg(\frac{h}{\epsilon}))$ time when the obstacles are convex. In computing approximate shortest paths, our algorithm extends the notion of coresets in [1] to simple polygons. However, our approach is computing coresets, and an approximate shortest path using these coresets is quite different from [1]. Our algorithm to construct the sketch of \mathcal{P} is simpler.
- As part of devising the above algorithm, when \mathcal{P} is comprised of convex polygonal obstacles, our algorithm computes a $(1 + \epsilon)$ -approximate geodesic Euclidean distance between the two given points in $O(n + \frac{h}{\epsilon} \lg \frac{h}{\epsilon})$ time. Further, our algorithm computes a $(1 + \epsilon)$ -approximate shortest path in additional $O(h \lg n)$ time. (Refer to Theorem 7.)
- When \mathcal{P} is comprised of disjoint convex polygonal obstacles, we preprocess these polygons in $O(n + \frac{h}{\epsilon^2} (\lg \frac{h}{\epsilon}) + \frac{h}{\epsilon} (\lg \frac{h}{\epsilon})^2)$ time to construct data structures of size $O(\frac{h}{\epsilon})$ for answering any two-point $(2 + \epsilon)$ -approximate geodesic distance (length) query in $O(\frac{1}{\epsilon^6} (\lg \frac{h}{\epsilon})^2)$ time. (Refer to Theorem 12.) To compute an optimal geodesic shortest path amid simple polygonal obstacles, Chen and Wang [5] takes $O(n + h \lg h + k)$ time, where k is a parameter sensitive to the geometric structures of the input and is upper bounded by $O(h^2)$. Our algorithm to answer approximate two-point distance queries amid convex polygonal obstacles takes space close to linear in n whereas the preprocessed data

11:4 Approximate Euclidean Shortest Paths

structures of algorithms proposed in [7] occupy $\Omega(n^5)$ space in the worst-case. Also, our algorithm for two-point distance queries improves the stretch factor of [4] from $(6 + \epsilon)$ to $(2 + \epsilon)$ in case of convex polygonal obstacles.

- Furthermore, our algorithm to compute the coreset of simple polygons to obtain a sketch Ω of \mathcal{P} as well as the algorithm to compute an approximate geodesic Euclidean shortest path in \mathcal{P} using the sketch Ω may be of independent interest.

Section 2 describes an algorithm for computing a single-shot approximate shortest path when obstacles in \mathcal{P} are convex polygons. Section 3 extends this algorithm to compute an approximate Euclidean shortest path amid simple polygonal obstacles. The algorithm to answer two-point approximate Euclidean distance queries amid convex polygonal obstacles is described in Section 4. A table comparing earlier algorithms to ours is given in the Appendix.

2 Approximate shortest path amid convex polygons

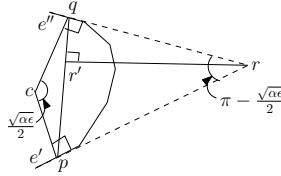
In this section, we consider the case in which every simple polygon in \mathcal{P} is convex. We use the following notation from Yao [30]. Let $\kappa \geq 2$, and define $\theta = 2\pi/\kappa$. Consider the set of κ rays: for $0 \leq i < \kappa$, the ray r_i passes through the origin and makes an angle $i\theta$ with the positive x -axis. Each pair of successive rays defines a cone whose apex is at the origin. This collection of κ cones is denoted by \mathcal{C} . It is clear that the cones of \mathcal{C} partition the plane. Also, the two bounding rays of any cone of \mathcal{C} make an angle θ . In our algorithm, the value of κ is chosen as a function of ϵ (refer to Subsection 2.2). When a cone $C \in \mathcal{C}$ is translated to have the apex at a point p , the translated cone is denoted with C_p . Each cone that we refer in this paper is a translated copy of some cone in \mathcal{C} . For each polygon P in \mathcal{P} , we choose a subset of $O(\frac{1}{\sqrt{\alpha\epsilon}})$ vertices from the vertices of P . At each such vertex p , we introduce a set of cones at p . In the algorithm to compute a single-shot s - t geodesic shortest path, the value of α is set to $\frac{\epsilon}{2}$. The algorithm for two-point approximate distance queries sets the value of α to $\frac{\epsilon}{12}$. The proof of Theorem 7 details the reasons for setting these specific values. As detailed below, these vertices and cones help in computing a spanner that approximates a Euclidean shortest path between the two given points in $\mathcal{F}(\mathcal{P})$.

2.1 Sketch of \mathcal{P}

In this subsection, we define and characterise the sketch of \mathcal{P} . For any $P_i \in \mathcal{P}$ and any two points p' and p'' on the boundary of P_i , the section of boundary of P_i that occurs while traversing from p' to p'' in counterclockwise order is termed a *patch* of P_i . In specific, we partition the boundary of each $P_i \in \mathcal{P}$ into a collection of patches Γ_i such that for any two points p', p'' belonging to any patch $\gamma \in \Gamma_i$, the angle between the outward (w.r.t. the centre of P_i) normals to respective edges at p' and p'' is upper bounded by $\frac{\sqrt{\alpha\epsilon}}{2}$. The maximum angle between the outward normals to any two edges that belong to a patch γ constructed in our algorithm is the *angle subtended by γ* . To facilitate in computing patches of any obstacle P_i , we partition the unit circle \mathbb{S}^2 centred at the origin into a minimum number of segments such that each circular segment is of length at most $\frac{\sqrt{\alpha\epsilon}}{2}$. For every such segment s of \mathbb{S}^2 , a patch (corresponding to s) comprises of the maximal set of the contiguous sequence of edges of P_i whose outward normals intersect s , when each of these normals is translated to the origin. (To avoid degeneracies, we assume each normal intersects a single segment.) Let Γ_i be a partition of the boundary of a convex polygon P_i into a collection of $O(\frac{1}{\sqrt{\alpha\epsilon}})$ patches. The lemma below shows that the geodesic distance between any two points belonging to any patch $\gamma \in \Gamma_i$ is a $(1 + \alpha\epsilon)$ -approximation to the Euclidean distance between them.

► **Lemma 1.** For any two points p and q that belong to any patch $\gamma \in \Gamma_j$, the geodesic distance between p and q along γ is upper bounded by $(1 + \alpha\epsilon)\|pq\|$ for $\alpha\epsilon < 1$.

Proof. Let e' be the edge on which p lies and let e'' be the edge on which q lies. Let c be the point of intersection of normal to e' at p and the normal to e'' at q . Since p and



■ **Figure 1** Illustrating the construction in proving the upper bound on the patch length.

q belong to the same patch, the angle between cp and cq is upper bounded by $\frac{\sqrt{\alpha\epsilon}}{2}$, when the value of $\alpha\epsilon$ is small. Let l' and l'' be the lines that respectively pass through e' and e'' . Also, let r be the point at which lines l' and l'' intersect. (Refer to Fig. 1.) For the small values of $\sqrt{\alpha\epsilon}$ and due to triangle inequality, the geodesic length of the patch between p and q is upper bounded by $\|pr\| + \|qr\|$. Let r' be the point of projection of r on to line segment pq . Suppose $\angle qrr' = \angle prr'$. (Analysis of other cases is similar.) Then $\|pr\| + \|qr\| \leq \frac{\|pr'\|}{\sin(\frac{\pi}{2} - \frac{\sqrt{\alpha\epsilon}}{4})} + \frac{\|r'q\|}{\sin(\frac{\pi}{2} - \frac{\sqrt{\alpha\epsilon}}{4})} = \frac{\|pq\|}{\cos \frac{\sqrt{\alpha\epsilon}}{4}} \leq (1 + \alpha\epsilon)\|pq\|$. The last inequality is valid when $\alpha\epsilon < 1$. ◀

For each obstacle P_i , the *coreset* S_i of P_i is comprised of two vertices chosen from each patch in Γ_i . In particular, for each patch $\gamma \in \Gamma_i$, the first and last vertices of γ that occur while traversing the boundary of P_i are chosen to be in the coreset S_i of P_i . The *coreset* \mathcal{S} of \mathcal{P} is then simply $\bigcup_i S_i$.

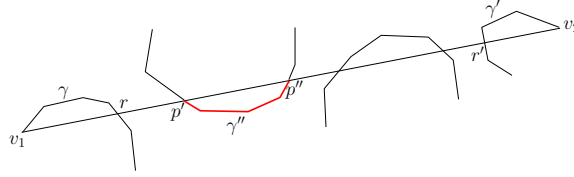
► **Observation 1.** The size of the coreset \mathcal{S} of \mathcal{P} is $O(\frac{h}{\sqrt{\alpha\epsilon}})$.

For every $1 \leq i \leq h$, our algorithm uses *core-polygon* $Q_i = CH(S_i)$ in place of P_i . For any single point obstacle P_i in \mathcal{P} , the core-polygon of P_i is that point itself. The patch construction procedure guarantees that each polygonal obstacle in \mathcal{P} is partitioned into patches such that the core-polygon that correspond to every obstacle in \mathcal{P} is valid. Let Ω be the set comprising of core-polygons corresponding to each of the polygons in \mathcal{P} . The set Ω is called the *sketch* of \mathcal{P} . The following lemmas show that Ω facilitates in computing a $(1 + \alpha\epsilon)$ -approximation of the geodesic distance between any two given points in $\mathcal{F}(\mathcal{P})$.

► **Lemma 2.** Let v', v'' be any two vertices of obstacles in Ω . Then, $dist_{\mathcal{P}}(v', v'')$ is upper bounded by $(1 + \alpha\epsilon)dist_{\Omega}(v', v'')$.

Proof. Let v_1, v_2 be any two successive vertices along a shortest path between v' and v'' in $\mathcal{F}(\Omega)$. Let $\mathcal{O} \subseteq \mathcal{P}$ be the set of obstacles intersected by the line segment v_1v_2 . Let v_1 and v_2 be respectively belonging to obstacles P_j and P_k . Also, let Γ_j (resp. Γ_k) be the set comprising the partition of boundary of P_j (resp. P_k) into patches. And, let S_j (resp. S_k) be the coreset of P_j (resp. P_k). Since the line segment v_1v_2 does not intersect the interior of the $CH(S_j)$ or $CH(S_k)$, it intersects at most one patch belonging to set Γ_j and at most one patch belonging to set Γ_k . Let v_1 and r be the points of intersection of line segment v_1v_2 with a patch $\gamma \in \Gamma_j$. (These points might as well be the endpoints of γ .) Then from Lemma 1, the geodesic distance between v_1 and r along γ is upper bounded by $(1 + \alpha\epsilon)\|v_1r\|$. (Refer Fig. 2.) Analogously, let v_2 and r' be the points of intersection of line segment v_1v_2

with a patch $\gamma' \in \Gamma_k$. Then the geodesic distance between v_2 and r' is upper bounded by $(1 + \alpha\epsilon)\|v_2 r'\|$. For any convex polygonal obstacle P_l in \mathcal{O} distinct from P_j and P_k , let p', p'' be the points of intersection of $v_1 v_2$ with the boundary of P_l . Since the line segment $v_1 v_2$ does not intersect the interior of the convex hull of coresets corresponding to P_l , both p' and p'' belong to the same patch, say $\gamma'' \in \Gamma_l$. Then again from Lemma 1, the geodesic distance between p' and p'' along patch γ'' is upper bounded by $(1 + \alpha\epsilon)\|p' p''\|$. We modify $v_1 v_2$ as follows: For every maximal subsection, say $p'_i p''_i$, of the line segment $v_1 v_2$ that is interior to a polygonal obstacle of \mathcal{P} , we replace that subsection with a geodesic Euclidean shortest path in $\mathcal{F}(\mathcal{P})$ between p'_i and p''_i .



■ **Figure 2** A line segment $v_1 v_2$ of a shortest path amid Ω intersecting three patches belonging to obstacles in \mathcal{P} .

Let $\gamma_1, \gamma_2, \dots, \gamma_k$ be the set Γ of patches intersected by the line segment $v_1 v_2$. Also, for every $1 \leq i \leq k$, let p'_i, p''_i be the points of intersections of $v_1 v_2$ with patch $\gamma_i \in \Gamma$ with p'_i closer to v_1 than v_2 along the line segment $v_1 v_2$. Then $\sum_{i=1}^k \text{dist}_{\mathcal{P}}(p'_i, p''_i)$ added with $\sum_{i=1}^{k-1} \|p''_i p'_{i+1}\|$ is upper bounded by $(1 + \alpha\epsilon)\|v_1 v_2\|$. Let v_1, \dots, v_l be the vertices of \mathcal{P} that occur in that order along a Euclidean shortest path in $\mathcal{F}(\Omega)$ between vertices $v', v'' \in \mathcal{P}$. Then $\text{dist}_{\mathcal{P}}(v_1, v_l) = \sum_{i=1}^{l-1} \text{dist}_{\mathcal{P}}(v_i, v_{i+1}) \leq (1 + \alpha\epsilon) \sum_{i=1}^{l-1} \text{dist}_{\Omega}(v_i, v_{i+1})$. Note that we do this transformation for each line segment of the shortest path that intersects any patch. ◀

Since $\mathcal{F}(\mathcal{P}) \subseteq \mathcal{F}(\Omega)$, every path that avoids convex polygonal obstacles in \mathcal{P} is also a path that avoids convex polygonal obstacles in Ω . This observation leads to the following:

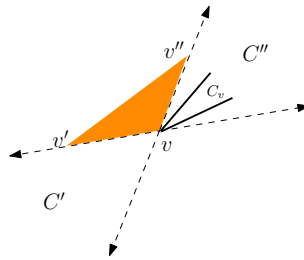
► **Lemma 3.** For any two vertices v', v'' of \mathcal{P} , $\text{dist}_{\Omega}(v', v'') \leq \text{dist}_{\mathcal{P}}(v', v'')$.

Considering the given two points $s, t \in \mathcal{F}(\mathcal{P})$ as degenerate obstacles, a $(1 + \alpha\epsilon)$ -approximation of the shortest distance between s and t amid polygonal obstacles in \mathcal{P} is computed.

► **Lemma 4.** For a set \mathcal{P} of h pairwise disjoint convex polygons in \mathbb{R}^2 and two points $s, t \in \mathcal{F}(\mathcal{P})$, the sketch \mathcal{S} of \mathcal{P} with cardinality $O(\frac{h}{\sqrt{\alpha\epsilon}})$ suffices to compute a $(1 + \alpha\epsilon)$ -approximate shortest path between s and t in $\mathcal{F}(\mathcal{P})$.

Proof. Immediate from Observation 1, Lemma 2, and Lemma 3. ◀

Our approach in computing coresets and an approximate shortest path using these coresets is quite different from [1]. As will be shown in Section 3, our sketch construction is extended to compute shortest paths even when \mathcal{P} comprises of polygon obstacles which are not necessarily convex. How our algorithm differs from [1] for the convex polygonal case is detailed herewith. Let \mathcal{P} be the polygonal domain defined with convex polygons P_1, P_2, \dots, P_h . In this algorithm as well as in [1], P_i is approximated with Q_i , for every $1 \leq i \leq h$. However, for every $1 \leq i \leq h$, in our algorithm $Q_i \subseteq P_i$ whereas in [1], $P_i \subseteq Q_i$. Let the new polygonal domain Ω be defined with simple polygons Q_1, Q_2, \dots, Q_h . Unlike [1], in computing Ω , our algorithm does not require using plane sweep algorithm to find pairwise vertically visible simple polygons of \mathcal{P} . As described above, our algorithm partitions the boundary of each convex polygon P into a set of patches.



■ **Figure 3** Illustrating an admissible cone C_v incident to a coreset vertex v of an obstacle.

2.2 Computing an approximate geodesic shortest path in $\mathcal{F}(\mathcal{P})$ using the sketch Ω of \mathcal{P}

Since we intend to compute an approximate shortest path, to keep our algorithm simpler, we do not want to use the algorithm from [17] to compute a shortest path amid convex polygonal obstacles in Ω . Instead, we use a spanner constructed with the conic Voronoi diagrams (CVDs) [8]. Further, in our algorithm, for any maximal line segment with endpoints r', r'' along the computed (approximate) shortest path amid obstacles in Ω , if the line segment $r'r''$ lies in $\mathcal{F}(\mathcal{Q}) - \mathcal{F}(\mathcal{P})$, we replace line segment $r'r''$ with the geodesic Euclidean shortest path between r' and r'' in $\mathcal{F}(\mathcal{P})$.

Since our algorithm relies on [8], we give a brief overview of that algorithm first. The algorithm in [8] constructs a spanner $G(V, E)$ for polygonal domain \mathcal{P} . Noting that the endpoints of line segments of a shortest path in $\mathcal{F}(\mathcal{P})$ are a subset of vertices of polygonal obstacles in \mathcal{P} , the node set V is defined as the vertex set of \mathcal{P} . Let \mathcal{C}' be the set of $O(\frac{1}{\epsilon})$ cones with apex at the origin of the coordinate system together partitioning \mathbb{R}^2 . (The cone angle of each cone in \mathcal{C}' except for one is set to ϵ and that one cone has $2\pi - \lfloor \frac{2\pi}{\epsilon} \rfloor \epsilon$ as the cone angle.) Let $C \in \mathcal{C}'$ be a cone with orientation θ and let $C' \in \mathcal{C}'$ be the cone with orientation $-\theta$. For each cone $C \in \mathcal{C}'$ and a set K of points, the set of cones resultant from introducing a cone C_p for every point $p \in K$, is the conic Voronoi diagram $CVD(C, K)$. (Note that as mentioned earlier, C_p is the cone resulted from translating cone C to have the apex at the point p .) For a given cone C_v , among all the points on the boundaries of polygons in \mathcal{P} that are visible from v , a point p whose projection onto the bisector of C_v is closest to v is said to be a *closest point in C_v to v* . If more than one point is closest in C_v to v , then we arbitrarily pick one of those points. For every vertex v of \mathcal{P} and for every cone C_v , if a closest point p in $C_v - \{v\}$ to v is not a vertex of \mathcal{P} , then the algorithm includes p as a node in V . Further, for every vertex v of \mathcal{P} and for every cone C_v , an edge e joining v and a closest point p in $C_v - \{v\}$ to v is introduced in E with its weight equal to the Euclidean distance between v and p . For every node v in G that corresponds to a point p on the boundary of $P \in \mathcal{P}$, if p is not a vertex of \mathcal{P} , then for every neighbor p' of p on the boundary of P which has a corresponding node v' in V , we introduce an edge e' between v and v' into E and set the weight of e' equal to the Euclidean distance between v and v' . These are the only edges included in E . The Theorem 2.5 in [8] proves that if d is the obstacle-avoiding geodesic Euclidean shortest path distance between any two vertices, say v' and v'' , of \mathcal{P} , then the distance between the corresponding nodes v' and v'' in G is upper bounded by $(1 + \epsilon)d$. The $CVD(C, K)$ is computed using the plane sweep in $O(|K| \lg |K|)$ time; and, the well-known planar point location data structure is used to locate the region in $CVD(C, K)$ to which a given query point belongs to.

As detailed below, apart from computing a sketch Ω of \mathcal{P} , as compared with [8], the number of cones per obstacle that participate in computing $CVDs$ amid $\mathcal{F}(\Omega)$ is further optimized by exploiting the convexity of obstacles together with the properties of shortest paths amid convex obstacles. By limiting the number of vertices of \mathcal{P} at which the cones are initiated to coreset \mathcal{S} of vertices, our algorithm improves the space complexity of the algorithm in [8]. Further, by exploiting the convexity of obstacles, we introduce $O(\frac{1}{\sqrt{\alpha\epsilon}})$ cones per obstacle, each with cone angle $O(\sqrt{\alpha\epsilon})$, and show that these are sufficient to achieve the claimed approximation factor.

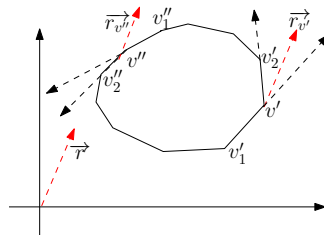
Let v be a vertex of \mathcal{P} that belongs to coreset S_i of convex polygon P_i . Let v', v, v'' be the vertices that respectively occur while traversing the boundary of P_i in counterclockwise order. Also, let C' be the cone defined by the pair of rays $(\overrightarrow{vv'}, -\overrightarrow{vv''})$ and let C'' be the cone defined by the pair of rays $(\overrightarrow{vv''}, -\overrightarrow{vv'})$. For a coreset vertex $v \in \mathcal{S}$, a cone $C \in \mathcal{C}$ is said to be *admissible* at v whenever $C_v \cap C'$ or $C_v \cap C''$ is non-empty. (See Fig. 3.) Let p and q be two points in $\mathcal{F}(\mathcal{P})$ such that p and q are not visible to each other due to polygonal obstacles in \mathcal{P} . Let v be a vertex of P_i through which a shortest path between p and q passes. Since any shortest path is convex at v with respect to P_i , there exists a shortest path between p and q where one of its line segment lies in C' , and another line segment of that path lies in C'' . Hence, in computing a Euclidean shortest path amid \mathcal{P} , it suffices to consider admissible cones at the vertices of \mathcal{P} .

Note that whenever two points s and t between which we intend to find a shortest path are visible to each other, the line segment st needs to be computed. To facilitate this, for every degenerate point obstacle p , every cone C with apex p is considered to be an admissible cone.

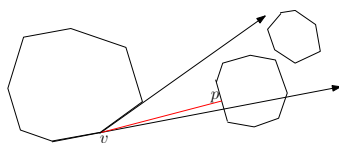
The same properties carry over to the polygonal domain Ω as well. For any two points p_1 and p_2 in $\mathcal{F}(\Omega)$, suppose that p_1 and p_2 are not visible to each other. Consider any shortest path τ between p_1 and p_2 . For any line segment ab in τ , ab is either an edge of a polygon in Ω or it is a tangent to an obstacle $O \in \Omega$. In the latter case, ab belongs to an admissible cone of O . When the polygonal domain is Ω , the following Lemma upper bounds the number of cones at the vertices of convex polygons in Ω .

► **Lemma 5.** *The number of cones introduced at all the obstacles of Ω is $O(\frac{h}{\sqrt{\alpha\epsilon}})$.*

Proof. Let O be the origin of the coordinate system. Let \vec{r} be a ray with origin at O . (See Fig. 4.) For any two distinct vertices v' and v'' of a convex polygon P , let $\vec{r}_{v'}$ be the ray parallel to \vec{r} with origin at v' and pointing in the same direction as \vec{r} and let $\vec{r}_{v''}$ be the ray parallel to \vec{r} with origin at v'' and point in the same direction as \vec{r} . Also, let v'_1 precede v' (resp. v''_1 precede v'') and v'_2 succeed v' (resp. v''_2 succeed v'') while traversing the boundary of P in counterclockwise order. Since P is a convex polygon, if every point of $\vec{r}_{v'}$ belongs to the cone defined by $\overrightarrow{v'_1v'}$ and $\overrightarrow{v'_2v'}$ then it is guaranteed that not every point of $\vec{r}_{v''}$ belongs to the cone defined by $\overrightarrow{v''_1v''}$ and $\overrightarrow{v''_2v''}$. Extending this argument, if a cone $C_{v'}$ is admissible



■ **Figure 4** Illustrating that a ray parallel to r can exist in only one admissible cone per obstacle.



■ **Figure 5** Illustrating an edge of the spanner.

at v' then the cone $C_{v''}$ cannot be admissible at v'' . Since the number of coreset vertices per obstacle is $O(\frac{1}{\sqrt{\alpha\epsilon}})$, the number of cones introduced per obstacle is $O(\frac{1}{\sqrt{\alpha\epsilon}})$. Further, since there are h convex polygonal obstacles, number of cones at all the obstacle vertices together is $O(\frac{h}{\sqrt{\alpha\epsilon}})$. ◀

Next, we describe the algorithm to compute the spanner $G(V = S \cup S', E)$. The set S comprises of nodes corresponding to coreset \mathcal{S} . The set S' is a set of Steiner points, as follows. For every $v \in S$ and every admissible cone C_v , let V' be the set of points on the boundaries of obstacles of Ω that are visible from v and belong to cone C_v . (See Fig. 5.) The point p in V' that is closest to v , termed the *closest Steiner point in C_v to v* , is determined and p is added to S' . An edge e between v and p is introduced in E while the Euclidean distance between v and p is set as the weight of e in G . Let p be located on a convex polygonal obstacle P . Further, for every Steiner point p , let v' (resp. v'') be the coreset vertex or Steiner point that lies on the boundary of P and occurs before (resp. after) p while traversing the boundary of P in counterclockwise order. Then an edge e' (resp. e'') between p and v' (resp. p and v'') is introduced in E while the geodesic distance between p and v' (resp. p and v'') along the boundary of P is set as the weight of e' (resp. e'') in G . Note that both $|V|$ and $|E|$ are $O(\frac{h}{\sqrt{\alpha\epsilon}})$. For any two points $s, t \in \mathcal{F}(\Omega)$, the following Lemma upper bounds the $dist_G(s, t)$ in terms of $dist_\Omega(s, t)$.

► **Lemma 6.** *Let G be the spanner constructed from Ω . Let $dist_G(p', p'')$ be the distance between p' and p'' in G . Then for any two points $s, t \in \mathcal{F}(\Omega)$, $dist_\Omega(s, t) \leq dist_G(s, t) \leq (1 + \sqrt{\alpha\epsilon})dist_\Omega(s, t)$.*

Proof. Theorem 2.5 of [8] concludes that to achieve $(1 + \alpha\epsilon)$ -approximation, $\sin \psi - \cos \psi \leq \frac{-1}{1 + \alpha\epsilon}$. Expanding *sine* and *cosine* functions for the first few terms yield $-1 + \psi + \frac{\psi^2}{2!} \leq \frac{-1}{1 + \alpha\epsilon}$. Solving the quadratic equation in ψ yields $\psi \leq \alpha\epsilon$. Since we are using cones with cone angle $\sqrt{\alpha\epsilon}$ in our algorithm, a $(1 + \sqrt{\alpha\epsilon})$ -approximation is achieved.

We claim that introducing a subset of cones (admissible cones) rather than all the cones as used in [8] does not affect the correctness. Let p and q be the vertices of two convex polygons P_i and P_j respectively. Suppose that pq is a line segment belonging to a shortest path R between vertices s and t of the spanner computed in [8]. Further, suppose that p occurs before q when R is traversed from s to t . If the line along pq supports P_i (resp. P_j) at p (resp. q), then the line segment pq belongs to an admissible cone at p (resp. q). Otherwise, there exists a line segment in the admissible cone with apex either at a vertex of P_i or at a vertex of P_j which would yield a shorter path from source s to q without using the line segment pq . ◀

Once we find a shortest path SP_Ω between s and t amid convex polygonal obstacles in Ω using the spanner G , following the proof of Lemma 3, we transform SP_Ω to a path amid obstacles in \mathcal{P} . Since there are $O(h)$ obstacles in Ω , SP_Ω contains $O(h)$ tangents between obstacles. Let this set of tangents be \mathcal{T} . We need to find points of intersection of convex polygons in \mathcal{P} with the line segments in \mathcal{T} . For any $l \in \mathcal{T}$ and $P_i \in \mathcal{P}$, by using the algorithm from Dobkin et al. [12], we compute the possible intersection between l and P_i . Whenever

11:10 Approximate Euclidean Shortest Paths

a line segment $l \in \mathcal{T}$ and a convex polygon $P_i \in \mathcal{P}$ intersect, say at points p' and p'' , we replace the line segment between p' and p'' with the geodesic shortest path between p' and p'' along the boundary of P_i . Analogously, for every line segment $l \in SP_\Omega - \mathcal{T}$ belonging to an obstacle $P_j \in \Omega$, we replace l with the corresponding geodesic path along the boundary of P_j . We use the plane sweep technique [11] to determine whichever line segments in \mathcal{T} could intersect with the convex obstacles in \mathcal{P} . Essentially, the event handling procedures of plane sweep algorithm replace every line segment in SP_Ω that intersects with any obstacle $P_j \in \mathcal{P}$ with the shortest geodesic shortest path along the boundary of P_j , so that the resulting shortest path $SP_{\mathcal{P}}$ after all such replacements belongs to $\mathcal{F}(\mathcal{P})$.

As part of the plane sweep, a vertical line is swept from left-to-right in the plane. Let L (resp. R) be the set of leftmost (resp. rightmost) vertices of convex polygons in \mathcal{P} . Initially, points in L and R together with the two endpoints of every line segment in \mathcal{T} are inserted into the priority queue Q . The event points are scheduled from Q using their respective distances from the initial sweep line position. As the events occur, the event points corresponding to L, R , and the endpoints of line segments in \mathcal{T} are handled and are deleted from Q . The algorithm terminates whenever Q is empty. As described below, the intersection points between the line segments in \mathcal{T} and the convex polygons in \mathcal{P} are added to Q with the traversal of the sweep line. The sweep line status is maintained as a balanced binary search tree B . We insert (resp. delete) a pointer to a line segment in \mathcal{T} or a pointer to a convex polygon in \mathcal{P} to B whenever leftmost (resp. rightmost) endpoint of it is popped from Q . We note that before a line segment $l \in \mathcal{T}$ and $P \in \mathcal{P}$ intersect, it is guaranteed that l and P occur adjacent along the sweep line. Hence, whenever l and P are adjacent in the sweep line status, we update the event-point schedule with the point of intersection between l and P that occurs first among all such points of intersection in traversing the sweep line from left to right. By using the algorithm from Dobkin et al. [12], we compute the possible intersection between l and P . If they do intersect, we push the leftmost point of their intersection to Q with the distance from the initial sweep line as the priority of that event point. Further, we store the rightmost intersection point between l and P with the leftmost point of intersection as satellite data. If the leftmost intersection point between l and P pops from Q , we compute the geodesic shortest path along the boundary of P between the leftmost intersection point and the corresponding rightmost intersection point. Further, whenever l and P become non-adjacent along the sweep line, we delete their leftmost point of intersection from Q .

► **Theorem 7.** *Given a set \mathcal{P} of pairwise disjoint convex polygons, two points $s, t \in \mathcal{F}(\mathcal{P})$, and $\epsilon \in (0, 0.6]$, computing a $(1 + \epsilon)$ -approximate geodesic distance between s and t takes $O(n + \frac{h}{\epsilon} \lg \frac{h}{\epsilon})$ time. Further, within an additional $O(h \lg n)$ time, a $(1 + \epsilon)$ -approximate shortest path is computed.*

Proof. From Lemma 4, we know that $dist_\Omega(s, t) \leq dist_{\mathcal{P}}(s, t) \leq (1 + \alpha\epsilon)dist_\Omega(s, t)$. Let G be the spanner constructed. From Lemma 6, we know that $dist_\Omega(s, t) \leq dist_G(s, t) \leq (1 + \sqrt{\alpha\epsilon})dist_\Omega(s, t)$. As detailed in Lemma 2, algorithm transforms a shortest path between s and t in G to a path p in $\mathcal{F}(\mathcal{P})$. Let $dist_{\mathcal{P}}^p(s, t)$ be the distance along p . From Lemma 2, $dist_{\mathcal{P}}^p(s, t) \leq (1 + \alpha\epsilon)dist_G(s, t)$. Hence, $dist_{\mathcal{P}}^p(s, t) \leq (1 + \alpha\epsilon)dist_G(s, t) \leq (1 + \alpha\epsilon)(1 + \sqrt{\alpha\epsilon})dist_\Omega(s, t) \leq (1 + \alpha\epsilon)(1 + \sqrt{\alpha\epsilon})dist_{\mathcal{P}}(s, t)$. Since p is a path in $\mathcal{F}(\mathcal{P})$, it is immediate to note that $dist_{\mathcal{P}}(s, t) \leq dist_{\mathcal{P}}^p(s, t)$. Therefore, $dist_{\mathcal{P}}(s, t) \leq dist_{\mathcal{P}}^p(s, t) \leq (1 + \alpha\epsilon)(1 + \sqrt{\alpha\epsilon})dist_{\mathcal{P}}(s, t)$. To achieve $(1 + \epsilon)$ -approximation, $(1 + \alpha\epsilon)(1 + \sqrt{\alpha\epsilon})$ needs to be less than or equal to $(1 + \epsilon)$. For small values of ϵ ($\epsilon \in (0, 0.6]$), choosing $\alpha = \frac{\epsilon}{2}$ satisfies this inequality.

From here on, we denote $\alpha\epsilon$ with ϵ' . Finding the coreset \mathcal{S} of vertices from the convex polygons in \mathcal{P} , and computing the set Ω of core-polygons together takes $O(n)$ time. The number of coreset vertices is $O(\frac{h}{\sqrt{\epsilon'}})$. The number of cones per obstacle is $O(\frac{1}{\sqrt{\epsilon'}})$. Therefore,

the total number of cones is $O(\frac{h}{\sqrt{\epsilon'}})$. For any cone $C \in \mathcal{C}$ and for any core-polygon $O \in \Omega$, at most a constant number of vertices of O are apexes to cones that have the orientation of C . Considering a sweep line in the orientation of C , the sweep line algorithm to find the closest Steiner point to the apex of each cone C (whenever an obstacle intersects with C) takes $O(h \lg h)$ time. Hence, computing the set of closest Steiner points corresponding to all the cone orientations in \mathcal{C} together take $O(\frac{h}{\sqrt{\epsilon'}} \lg h)$.

The number of nodes in the spanner G is $O(\frac{h}{\sqrt{\epsilon'}})$. These nodes include coresets vertices and at most one closest Steiner point per cone. As each cone introduces at most one edge into G , the number of edges in G is $O(\frac{h}{\sqrt{\epsilon'}})$. Using the Fredman-Tarjan algorithm [13], finding a shortest path between s and t in G takes $O(\frac{h}{\sqrt{\epsilon'}} \lg \frac{h}{\sqrt{\epsilon'}})$ time. Hence, computing the $(1 + \epsilon)$ -approximate distance between s and t takes $O(n + \frac{h}{\sqrt{\epsilon'}} \lg \frac{h}{\sqrt{\epsilon'}})$ time. For $\alpha = \frac{\epsilon}{2}$, the value of ϵ' is $O(\epsilon^2)$. Hence, the result stated in the theorem statement.

For the plane sweep, leftmost and rightmost extreme vertices of convex polygons in \mathcal{P} are found in $O(n)$ time. There are $O(h)$ line segments in \mathcal{T} , cardinality of Ω is $O(h)$, and $O(h)$ line segment-obstacle pairs (respectively from \mathcal{T} and \mathcal{P}) that intersect. The number of event points due to the endpoints in sets L, R , and the endpoints of line segments in \mathcal{T} is $O(h)$. If l and P become non-adjacent along the sweep line, deleting their point of intersection from Q is charged to the event that caused them non-adjacent. The sweep line status is updated if any of these $O(h)$ number of event points occur. Analogous to the analysis provided for line segment intersection [11], our plane sweep algorithm takes $O(n + h \lg h)$ time.

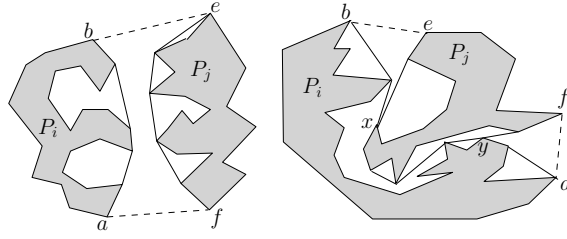
Due to Dobkin et al. [12], determining whether a line segment l in SP_Ω intersects with an obstacle P takes $O(\lg n)$ time, The preprocessing structures corresponding to [12] take $O(n)$ space and they are constructed in $O(n)$ time. Further, replacing every line segment between points of intersection with their respective geodesic shortest paths along the boundaries of obstacles together take $O(n)$ time. ◀

Note that the proof of the above theorem requires us to set the value of α to $\frac{\epsilon}{2}$.

3 Approximate shortest path amid simple polygons

In this section, we extend the approximation method from previous sections to the case of simple (not necessarily convex) polygons. This is accomplished by first decomposing $\mathcal{F}(\mathcal{P})$ into a set of corridors, funnels, hourglasses, and junctions [19, 20, 22]. In the following, we describe these geometric structures, and then we detail our algorithm.

For convenience, we assume a bounding box encloses the polygonal domain \mathcal{P} . In the following, we describe a coarser decomposition of $\mathcal{F}(\mathcal{P})$ as compared to the triangulation of $\mathcal{F}(\mathcal{P})$. In specific, this decomposition is used in our algorithm to achieve efficiency. Let $Tri(\mathcal{F})$ denote a triangulation of $\mathcal{F}(\mathcal{P})$. The line segments of $Tri(\mathcal{F})$ that are not the edges of obstacles in \mathcal{P} are referred to as *diagonals*. Let $G(\mathcal{F})$ denote the dual graph of $Tri(\mathcal{F})$, i.e., each node of $G(\mathcal{F})$ corresponds to a triangle of $Tri(\mathcal{F})$ and each edge connects two nodes corresponding to two triangles sharing a diagonal of $Tri(\mathcal{F})$. Based on $G(\mathcal{F})$, we compute a planar 3-regular graph, denoted by G_3 (the degree of every node in G_3 is three), possibly with loops and multi-edges, as follows. First, we remove each degree-one node from $G(\mathcal{F})$ along with its incident edge; repeat this process until no degree-one node remains in the graph. Second, remove every degree-two node from $G(\mathcal{F})$ and replace its two incident edges by a single edge; repeat this process until no degree-two node remains. The resultant graph G_3 is planar, which has $O(h)$ faces, nodes, and edges. Every node of G_3 corresponds to a triangle in $Tri(\mathcal{F})$, called a *junction triangle*. The removal of all junction triangles results in



■ **Figure 6** Illustrating an open hourglass (left) and a closed hourglass (right) with a corridor path connecting the apexes x and y of the two funnels. The dashed segments are diagonals.

$O(h)$ corridors. The points s and t between which a shortest path needs to be computed are placed in their own degenerate single point corridors. The boundary of each corridor C consists of four parts (see Fig. 6): (1) A boundary portion of an obstacle $P_i \in \mathcal{P}$, from a point a to a point b ; (2) a diagonal of a junction triangle from b to a point e on an obstacle $P_j \in \mathcal{P}$ ($P_i = P_j$ is possible); (3) a boundary portion of the obstacle P_j from e to a point f ; (4) a diagonal of a junction triangle from f to a . The corridor C is a simple polygon. Let $\tau(a, b)$ (resp., $\tau(e, f)$) be the Euclidean shortest path from a to b (resp., e to f) in C . The region H_C bounded by $\tau(a, b)$, $\tau(e, f)$, \bar{be} , and \bar{fa} is called an *hourglass*, which is *open* if $\tau(a, b) \cap \tau(e, f) = \emptyset$ and *closed* otherwise. (Refer Fig. 6.) If H_C is open, then both $\tau(a, b)$ and $\tau(e, f)$ are convex polygonal chains and are called the *sides* of H_C ; otherwise, H_C consists of two *funnels* and a path $\tau_C = \tau(a, b) \cap \tau(e, f)$ joining the two apexes of the two funnels, and τ_C is called the *corridor path* of C . Let x and y be the endpoints of τ_C . Also, let x be at a shorter distance from b as compared to y . The paths $\tau(b, x)$, $\tau(e, x)$, $\tau(a, y)$, and $\tau(f, y)$ are termed *sides of funnels* of hourglass H_C . We note that these paths are indeed convex polygonal chains. The apices x and y together is termed a *apex pair* of hourglass H_C . Further, the shortest path between x and y along the boundary of H_C is the *corridor path between apexes* of H_C .

We first give an overview of our algorithm for simple polygonal obstacles. A sketch of \mathcal{P} comprising of a sequence of convex polygonal (core-)chains is computed. Each such core-chain either corresponds to an approximation of a side of an open hourglass or a side of a funnel. If a simple polygon does not participate in any closed corridor, these polygonal chains together form a core-polygon. Similar to the convex polygon case, each such polygonal chain is partitioned into patches. Using these chains, we compute a spanner G . In addition, the following set of edges are included in G : for every closed hourglass H_C and for each obstacle P that participates in H_C , an edge representing the unique shortest path between the two apices of H_C (as detailed below). After we compute a shortest path p between s and t in the spanner, for every edge $e(r', r'') \in p$, if e is an edge that corresponds to the closed corridor path then we replace e with a shortest path (sequence of edges) between r' and r'' in $\mathcal{F}(\mathcal{P})$. The resultant path is the output of our algorithm. The scheme designed in Agarwal et al. [1] does not appear to extend easily to the case of simple polygons as they use the critical step of computing partitioning planes between pairs of convex polygonal obstacles from \mathcal{P} .

For every obstacle $P_j \in \mathcal{P}$, let \mathcal{R}_j be the union of the following: (i) the set comprising of open hourglass sides whose endpoints are incident to P_j , and (ii) the set comprising of sections of funnel sides whose non-apex endpoints incident to P_i . Note that the elements of sets in (i) and (ii) are polygonal convex chains. For every $R \in \mathcal{R}_j$, similar to the case of convex polygonal obstacles, we partition R into patches and the set comprising of the endpoints of these patches is the coreset of R . (For details, refer to Section 2.) For every

$R \in \mathcal{R}_j$, the *core-chain* of R is obtained by joining every two successive vertices that belong to the coreset of R with a line segment while traversing the boundary of R . We construct a spanner $G(V, E)$ that correspond to core-chains of \mathcal{P} using *CVDs*. For every admissible cone C_p at every vertex p of every core-chain, we consider C_p only if C_p has an intersection with $\mathcal{F}(\mathcal{P})$. While noting that Clarkson's method extends to core-chains defined as above, the shortest path determination algorithm for simple polygons is the same as for the convex polygons described in the previous section except for the following. For each apex pair $v'-v''$, an edge e is introduced into G between the vertices of G that correspond to v' and v'' with the weight of e equal to the geodesic distance between v' and v'' in the closed hourglass. For a shortest path p between any two nodes of G , for every edge $e \in p$ if both the endpoints of e correspond to an apex pair $a'-a''$ then we replace p with the shortest path between a' and a'' so that that path contains the corridor path of that closed hourglass; otherwise, as in Lemma 2, we replace the line segment l correspond to e with the sections of l together with the geodesic paths along the boundaries of patches that l intersects. Thus a shortest path between s and t in the spanner G is transformed to a path in the $\mathcal{F}(\mathcal{P})$. In addition, since the distance along the path that contains the corridor path between every pair of apexes is made as the weight of its corresponding edge in the spanner, and due to Lemma 6, the distance along the transformed path is a $(1 + \alpha\epsilon)$ -approximation to the distance between s and t amid obstacles in \mathcal{P} .

► **Lemma 8.** *For a set \mathcal{P} of h pairwise disjoint simple polygons in \mathbb{R}^2 and two points $s, t \in \mathcal{F}(\mathcal{P})$, the sketch of \mathcal{P} with cardinality $O(\frac{h}{\sqrt{\alpha\epsilon}})$ suffices to compute a $(1 + \alpha\epsilon)$ -approximate shortest path between s and t in $\mathcal{F}(\mathcal{P})$.*

Computing hourglasses of $\mathcal{F}(\mathcal{P})$ using [19, 20, 22] and determining the core-chains together takes $O(n + h(\lg h)^{1+\delta} + h \lg n)$ time (where δ is a small positive constant resulting from the triangulation of $\mathcal{F}(\mathcal{P})$ using the algorithm from [3]). Extending the proof of Theorem 7 leads to the following.

► **Theorem 9.** *Given a set \mathcal{P} of pairwise disjoint simple polygonal obstacles, two points $s, t \in \mathcal{F}(\mathcal{P})$, and $\epsilon \in (0, 0.6]$, a $(1 + \epsilon)$ -approximate geodesic shortest path between s and t is computed in $O(n + h((\lg n) + (\lg h)^{1+\delta} + (\frac{1}{\epsilon} \lg \frac{h}{\epsilon})))$ time. Here, δ is a small positive constant (resulting from the time involved in triangulating $\mathcal{F}(\mathcal{P})$ using [3]).*

Same as in Theorem 7, the proof of this theorem also needs the value of α to be equal to $\frac{\epsilon}{2}$.

4 Two-point approximate distance queries amid convex polygons

We preprocess the given set \mathcal{P} of convex polygons to output the approximate distance between any two query points located in $\mathcal{F}(\mathcal{P})$. Like in the previous section, our preprocessing algorithm relies on [8] and constructs a spanner G . Our query algorithm constructs an auxiliary graph from G . We compute the approximate distance between the two query points using a shortest path finding algorithm in the auxiliary graph.

4.1 Preprocessing

The graph G constructed as part of preprocessing in Section 2.2 is useful in finding an approximate Euclidean shortest path in $\mathcal{F}(\mathcal{P})$ between any two vertices in \mathcal{P} . Instead of finding a shortest path between two query nodes in G , to improve the query time complexity, we compute a planar graph $G^{pl}(V, E^{pl})$ from $G(V, E)$ using the result from Chew [6]. Chew's

11:14 Approximate Euclidean Shortest Paths

algorithm finds a set $E^{pl} \subseteq E$ in $O(|V| \lg |V|)$ time so that the distance between any two nodes of G^{pl} is a 2-approximation of the distance between the corresponding nodes in G . We use the algorithm from Kawarabayashi et al. [23] to efficiently answer $(1 + \epsilon)$ -approximate distance (length) queries in G^{pl} . More specifically, [23] takes $O(|V|(\lg |V|)^2)$ time to construct a data structure of size $O(|V|)$ so that any distance query is answered in $O((\frac{\lg |V|}{\epsilon})^2)$ time.

► **Lemma 10.** *Let G be the spanner computed for the polygonal domain Ω using the algorithm mentioned in Subsection 2.2. Let s and t be two points in $\mathcal{F}(\mathcal{P})$. Let G^{pl} be the planar graph constructed from G using [6]. Further, let $dist_K(s, t)$ be the distance between s and t in G^{pl} computed using the algorithm from [23]. By choosing $\alpha = \frac{\epsilon}{12}$, $dist_{\mathcal{P}}(s, t) \leq dist_K(s, t) \leq (2 + \epsilon)dist_{\mathcal{P}}(s, t)$.*

Proof. From Lemma 4, we know that $dist_{\Omega}(s, t) \leq dist_{\mathcal{P}}(s, t) \leq (1 + \alpha\epsilon)dist_{\Omega}(s, t)$. From Lemma 6, we know that $dist_{\Omega}(s, t) \leq dist_G(s, t) \leq (1 + \sqrt{\alpha\epsilon})dist_{\Omega}(s, t)$. Let $dist_{G^{pl}}(s, t)$ be the distance in G^{pl} between nodes s and t of G^{pl} . From [6], $dist_G(s, t) \leq dist_{G^{pl}}(s, t) \leq 2dist_G(s, t)$. Further, as mentioned above, $dist_{G^{pl}}(s, t) \leq dist_K(s, t) \leq (1 + \alpha\epsilon)dist_{G^{pl}}(s, t)$. As detailed in Lemma 2, algorithm transforms a shortest path between s and t in K to a path p in $\mathcal{F}(\mathcal{P})$. Let $dist_{\mathcal{P}}^p(s, t)$ be the distance along p . From Lemma 2, $dist_{\mathcal{P}}^p(s, t) \leq (1 + \alpha\epsilon)dist_K(s, t)$. Hence, $dist_{\mathcal{P}}^p(s, t) \leq (1 + \alpha\epsilon)dist_K(s, t) \leq (1 + \alpha\epsilon)^2dist_{G^{pl}}(s, t) \leq 2(1 + \alpha\epsilon)^2dist_G(s, t) \leq 2(1 + \alpha\epsilon)^2(1 + \sqrt{\alpha\epsilon})dist_{\Omega}(s, t) \leq 2(1 + \alpha\epsilon)^2(1 + \sqrt{\alpha\epsilon})dist_{\mathcal{P}}(s, t)$. Since p is a path in $\mathcal{F}(\mathcal{P})$, it is immediate to note that $dist_{\mathcal{P}}(s, t) \leq dist_{\mathcal{P}}^p(s, t)$. Therefore, $dist_{\mathcal{P}}(s, t) \leq dist_{\mathcal{P}}^p(s, t) \leq 2(1 + \alpha\epsilon)^2(1 + \sqrt{\alpha\epsilon})dist_{\mathcal{P}}(s, t)$. To achieve $(2 + \epsilon)$ -approximation, $2(1 + \alpha\epsilon)^2(1 + \sqrt{\alpha\epsilon})$ needs to be less than or equal to $(2 + \epsilon)$. For small values of ϵ ($\epsilon \in (0, 0.7)$), choosing $\alpha = \frac{\epsilon}{12}$ satisfies this inequality. ◀

We note that $\alpha\epsilon$ is $O(\epsilon^2)$. We suppose that there are $O(\frac{1}{\epsilon})$ cones in \mathcal{C} , each cone with a cone angle $O(\epsilon)$. It remains to describe data structures that need to be constructed during the preprocessing phase for obtaining the closest vertex of the query point s (resp. t) in a given cone C_s (resp. C_t). To efficiently determine all these $O(\frac{1}{\epsilon})$ neighbors to s and t during query time, we construct a set of $O(\frac{1}{\epsilon})$ *CVDs*: for every $C \in \mathcal{C}$, one *CVD* that corresponds to C . The *CVDs* are constructed similarly to the algorithm given in Subsection 2.2.

► **Lemma 11.** *The preprocessing phase takes $O(n + \frac{h}{\epsilon^2}(\lg \frac{h}{\epsilon}) + \frac{h}{\epsilon}(\lg \frac{h}{\epsilon})^2)$ time. The space complexity of the data structures constructed by the end of the preprocessing phase is $O(\frac{h}{\epsilon})$.*

Proof. Computing the sketch Ω from the given \mathcal{P} takes $O(n + \frac{h}{\epsilon})$ time. The number of cones in all the *CVDs* together is $O(\frac{h}{\epsilon})$. It takes $O(\frac{1}{\epsilon} \frac{h}{\epsilon} \lg \frac{h}{\epsilon})$ time to compute G which include computing *CVDs*. Due to [6], computing planar graph G^{pl} with $O(\frac{h}{\epsilon})$ nodes takes $O(\frac{h}{\epsilon} \lg \frac{h}{\epsilon})$ time. Computing space-efficient data structures using [23] takes $O(\frac{h}{\epsilon} (\lg \frac{h}{\epsilon})^2)$ time. Hence, the preprocessing phase takes $O(n + \frac{h}{\epsilon} \lg \frac{h}{\epsilon} + \frac{h}{\epsilon} ((\lg \frac{h}{\epsilon})^2))$ time. Further, data structures constructed using [23] by the end of preprocessing phase occupy $O(\frac{h}{\epsilon})$ space. The Kirkpatrick's point location [24] data structures for planar point location take $O(\frac{h}{\epsilon})$ space. ◀

4.2 Shortest distance query processing

The query algorithm finds the obstacle-avoiding Euclidean shortest path distance between any two given points $s, t \in \mathcal{F}(\mathcal{P})$. We construct a graph G_{st} from G^{pl} . (The graph G^{pl} is as defined in Subsection 4.1.) For every $C \in \mathcal{C}$, if the point s is located in the cell of a point p of *CVD* corresponding to C , then we introduce a node corresponding to p into a set V_s . (Essentially, p is the closest visible point in cone $-C_s$ to point s .) Analogously, we define the set V_t of nodes for t in G_{st} . The node set of G_{st} comprises of nodes in $V_s \cup V_t \cup \{s, t\}$.

The edges of this graph are of three kinds: $\{s\} \times V_s$, $V_s \times V_t$ and $\{t\} \times V_t$. Since there are $O(\frac{1}{\epsilon})$ CVDs, the number of nodes and edges of G_{st} are respectively $O(\frac{1}{\epsilon})$ and $O(\frac{1}{\epsilon^2})$. For every edge (s, s') (resp. (t, t')) with $s' \in V_s$ (resp. $t' \in V_t$), the weight of edge (s, s') (resp. (t, t')) is the Euclidean distance between s and s' (resp. t and t'). For every edge (s', t') with $s' \in V_s$ and $t' \in V_t$, the weight of (s', t') is the $(2 + \epsilon)$ -approximate distance between s' and t' . These weights are obtained from the data structures maintained as in [23]. We use Fredman-Tarjan algorithm [13] to find a shortest path between s and t in G_{st} . From the above, this distance is a $(2 + \epsilon)$ -approximate distance from s to t amid convex polygons in \mathcal{P} .

► **Theorem 12.** *Given a set \mathcal{P} of h pairwise disjoint convex polygonal obstacles in plane defined with n vertices and $\epsilon \in (0, 0.6]$, the polygons in \mathcal{P} are preprocessed in $O(n + \frac{h}{\epsilon^2}(\lg \frac{h}{\epsilon}) + \frac{h}{\epsilon}(\lg \frac{h}{\epsilon})^2)$ time to construct data structures of size $O(\frac{h}{\epsilon})$ for answering two point $(2 + \epsilon)$ -approximate distance query between any two given points belonging to $\mathcal{F}(\mathcal{P})$ in $O(\frac{1}{\epsilon^6}(\lg \frac{h}{\epsilon})^2)$ time.*

References

- 1 P. K. Agarwal, R. Sharathkumar, and H. Yu. Approximate Euclidean shortest paths amid convex obstacles. In *Proceedings of Symposium on Discrete Algorithms*, pages 283–292, 2009.
- 2 S. R. Arikati, D. Z. Chen, L. P. Chew, G. Das, M. H. M. Smid, and C. D. Zaroliagis. Planar spanners and approximate shortest path queries among obstacles in the plane. In *Proceedings of European Symposium on Algorithms*, pages 514–528, 1996.
- 3 R. Bar-Yehuda and B. Chazelle. Triangulating disjoint Jordan chains. *International Journal of Computational Geometry & Applications*, 4(4):475–481, 1994.
- 4 D. Z. Chen. On the all-pairs Euclidean short path problem. In *Proceedings of Symposium on Discrete Algorithms*, pages 292–301, 1995.
- 5 D. Z. Chen and H. Wang. Computing shortest paths among curved obstacles in the plane. *ACM Transactions on Algorithms*, 11(4):26:1–26:46, 2015.
- 6 L. P. Chew. There are planar graphs almost as good as the complete graph. *Journal of Computer and System Sciences*, 39(2):205–219, 1989.
- 7 Y.-J. Chiang and J. S. B. Mitchell. Two-point Euclidean shortest path queries in the plane. In *Proceedings of Symposium on Discrete Algorithms*, pages 215–224, 1999.
- 8 K. L. Clarkson. Approximation algorithms for shortest path motion planning. In *Proceedings of Symposium on Theory of Computing*, pages 56–65, 1987.
- 9 T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2009.
- 10 Toth C. D., O'Rourke J., and Goodman J. E. *Handbook of discrete and computational geometry*. CRC Press, 3rd ed. edition, 2017.
- 11 M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: algorithms and applications*. Springer-Verlag, 3rd ed. edition, 2008.
- 12 D. P. Dobkin and David G. Kirkpatrick. Fast detection of polyhedral intersection. *Theoretical Computer Science*, 27:241–253, 1983.
- 13 M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of ACM*, 34(3):596–615, 1987.
- 14 S. K. Ghosh. *Visibility algorithms in the plane*. Cambridge University Press, New York, USA, 2007.
- 15 L. J. Guibas and J. Hershberger. Optimal shortest path queries in a simple polygon. *Journal of Computer and System Sciences*, 39(2):126–152, 1989.
- 16 H. Guo, A. Maheshwari, and J-R. Sack. Shortest path queries in polygonal domains. In *Proceedings of Algorithmic Aspects in Information and Management*, pages 200–211, 2008.
- 17 J. Hershberger and S. Suri. An optimal algorithm for Euclidean shortest paths in the plane. *SIAM Journal on Computing*, 28(6):2215–2256, 1999.

- 18 R. Inkulu, S. Kapoor, and S. N. Maheshwari. A near optimal algorithm for finding Euclidean shortest path in polygonal domain. *CoRR*, abs/1011.6481, 2010. [arXiv:1011.6481](https://arxiv.org/abs/1011.6481).
- 19 S. Kapoor. Efficient computation of geodesic shortest paths. In *Proceedings of Symposium on Theory of Computing*, pages 770–779, 1999.
- 20 S. Kapoor and S. N. Maheshwari. Efficient algorithms for Euclidean shortest path and visibility problems with polygonal obstacles. In *Proceedings of Symposium on Computational Geometry*, pages 172–182, 1988.
- 21 S. Kapoor and S. N. Maheshwari. Efficiently constructing the visibility graph of a simple polygon with obstacles. *SIAM Journal on Computing*, 30(3):847–871, 2000.
- 22 S. Kapoor, S. N. Maheshwari, and J. S. B. Mitchell. An efficient algorithm for Euclidean shortest paths among polygonal obstacles in the plane. *Discrete & Computational Geometry*, 18(4):377–383, 1997.
- 23 K. Kawarabayashi, P. N. Klein, and C. Sommer. Linear-space approximate distance oracles for planar, bounded-genus and minor-free graphs. In *Proceedings of Colloquium on Automata, Languages and Programming*, pages 135–146, 2011.
- 24 D. G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, 12(1):28–35, 1983.
- 25 J. Kleinberg and E. Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., 2005.
- 26 J. S. B. Mitchell. Shortest paths among obstacles in the plane. *International Journal of Computational Geometry & Applications*, 6(3):309–332, 1996.
- 27 J. S. B. Mitchell. Geometric shortest paths and network optimization. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 633–701. North-Holland, 2000.
- 28 S. Sen. Approximating shortest paths in graphs. In *Proceedings of Workshop on Algorithms and Computation*, pages 32–43, 2009.
- 29 E. Welzl. Constructing the visibility graph for n -line segments in $O(n^2)$ time. *Information Processing Letters*, 20(4):167–171, 1985.
- 30 A. C. Yao. On constructing minimum spanning trees in k -dimensional spaces and related problems. *SIAM Journal on Computing*, 11(4):721–736, 1982.

A Appendix

Comparison with previous results is depicted in Table 1.

In Chiang and Mitchell [7], h_s (resp. h_t) is the number of vertices visible from s (resp. t). Both the Chen [4] and Arikati et al. [2] output a shortest path in additional $O(L)$ time, where L is the number of edges of the output path. The r in Arikati et al. [2] is an arbitrary integer such that $1 \leq r \leq n$.

Table 1 Comparison with previous results.

	Preprocessing time	Space	Query time	Time	Stretch	Comment
Our results	-	-	-	$O(n + h((\lg n)^{1+\delta} + \frac{1}{\epsilon} \lg \frac{h}{\epsilon}))$	$1 + \epsilon$	non-convex
	-	-	-	$O(n + \frac{h}{\epsilon} \lg \frac{h}{\epsilon})$	$1 + \epsilon$	convex
Agarwal et al. [1]	$O(n + \frac{h}{\epsilon} (\lg \frac{h}{\epsilon}) + \frac{h}{\epsilon} (\lg \frac{h}{\epsilon})^2)$	$O(\frac{h}{\epsilon})$	$O(\frac{1}{\epsilon^2} (\lg \frac{h}{\epsilon})^2)$	-	$2 + \epsilon$	convex
Chiang & Mitchell [7]	-	-	-	$O(n + \frac{h}{\sqrt{\epsilon}} \lg(\frac{h}{\epsilon}))$	$1 + \epsilon$	convex
	-	$O(n^{5+\epsilon})$	$o(n)$	-	optimal	non-convex
	-	$O(n^{5+10\delta+\epsilon})$	$O(n^{1-\delta} \lg n)$	-	optimal	non-convex
	-	$O(n^{10} \lg n)$	$O((\lg n)^2)$	-	optimal	non-convex
	-	$O(n^{11})$	$O(\lg n)$	-	optimal	non-convex
	-	$O(n^5)$	$O(\lg n + \min h_s, h_t)$	-	optimal	non-convex
	-	$O(n + h^5)$	$O(h \lg n)$	-	optimal	non-convex
Chen [4]	-	$O(n \lg n + \frac{n}{\epsilon})$	$O(\frac{\lg n}{\epsilon} + \frac{1}{\epsilon^2})$	-	$6 + \epsilon$	non-convex
Arikati et al. [2]	$O(\frac{n^2}{\sqrt{r}})$	$O(\frac{n^2}{\sqrt{r}})$	$O(\lg n + \sqrt{r})$	-	$\sqrt{2} + \epsilon$	non-convex
	$O(n \lg n)$	$O(n)$	$O(n)$	-	$\sqrt{2} + \epsilon$	non-convex
	$O(n^{3/2})$	$O(n^{3/2})$	$O(\lg n)$	-	$2\sqrt{2} + \epsilon$	non-convex
	$O(\frac{n^{3/2}}{\sqrt{\lg n}})$	$O(n \lg n)$	$O(\lg n)$	-	$3\sqrt{2} + \epsilon$	non-convex

Reachability in High Treewidth Graphs

Rahul Jain 

Indian Institute of Technology Kanpur, India
jain@iitk.ac.in

Raghunath Tewari

Indian Institute of Technology Kanpur, India
rtewari@iitk.ac.in

Abstract

Reachability is the problem of deciding whether there is a path from one vertex to the other in the graph. Standard graph traversal algorithms such as DFS and BFS take linear time to decide reachability; however, their space complexity is also linear. On the other hand, Savitch's algorithm takes quasipolynomial time although the space bound is $O(\log^2 n)$. Here, we study space efficient algorithms for deciding reachability that run in polynomial time.

In this paper, we show that given an n vertex directed graph of treewidth w along with its tree decomposition, there exists an algorithm running in polynomial time and $O(w \log n)$ space that solves the reachability problem.

2012 ACM Subject Classification Theory of computation

Keywords and phrases graph reachability, simultaneous time-space upper bound, tree decomposition

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.12

Funding *Rahul Jain*: Ministry of Human Resource Development, Government of India
Raghunath Tewari: DST Inspire Faculty Grant, Visvesvaraya Young Faculty Fellowship

1 Introduction

Given a graph G and two vertices u and v in G , the reachability problem is to decide if there exists a path from u to v in G . This problem is NL-complete for directed graphs and L-complete for undirected graphs [17]. Hence its study gives important insight into space bounded computations. We will henceforth refer to the problem of directed graph reachability as *Reach*. The famous open question $L \stackrel{?}{=} NL$ essentially asks if there is a deterministic logspace algorithm for *Reach* or not. *Reach* can be solved in $\Theta(n \log n)$ space and optimal time using standard graph traversal algorithms such as DFS and BFS. We also know, due to Savitch, that it can be solved in $\Theta(\log^2 n)$ space [18]. However, Savitch's algorithm requires $n^{\Theta(\log n)}$ time. Wigderson surveyed reachability problems in which he asked if there is an algorithm for *Reach* that runs simultaneously in $O(n^{1-\epsilon})$ space (for any $\epsilon > 0$) and polynomial time [19]. Here, we make some partial progress towards answering this question.

In 1998 Barnes et al. made progress in answering Wigderson's question for general graphs by presenting an algorithm for *Reach* that runs simultaneously in $n/2^{\Theta(\sqrt{\log n})}$ space and polynomial time [6]. For several other topologically restricted classes of graphs, there has been significant progress in giving polynomial time algorithms for *Reach* that run simultaneously in sublinear space. For grid graphs a space bound of $O(n^{1/2+\epsilon})$ was first achieved [4]. The same space bound was then extended to all planar graphs by Imai et al. [14]. Later for planar graphs, the space bound was improved to $\tilde{O}(n^{1/2})$ space by Asano et al. [5]. For graphs of higher genus, Chakraborty et al. gave an $\tilde{O}(n^{2/3}g^{1/3})$ space algorithm which additionally requires, as an input, an embedding of the graph on a surface of genus g [8]. They also gave an $\tilde{O}(n^{2/3})$ space algorithm for H minor-free graphs which requires tree decomposition of



© Rahul Jain and Raghunath Tewari;

licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 12; pp. 12:1–12:14

Leibniz International Proceedings in Informatics



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

the graph as an input and $O(n^{1/2+\epsilon})$ space algorithm for $K_{3,3}$ -free and K_5 -free graphs. For layered planar graphs, Chakraborty and Tewari showed that for every $\epsilon > 0$ there is an $O(n^\epsilon)$ space algorithm [9].

Treewidth is a well-studied property of graphs. The value of treewidth can range from 1 (for a tree) to $n - 1$ (for a complete graph on n vertices). The computational complexity of many difficult problems becomes easy for bounded treewidth graphs. We can solve classical problems such as the Hamiltonian circuit, vertex cover, Steiner tree, and vertex coloring in linear time for bounded treewidth [3]. The weighted independent set problem can be solved in $O(2^w n)$ time [7]. It is NP-complete to find on given input $\langle G, k \rangle$ if G has treewidth k [2]. However, an $O(\sqrt{\log n})$ -factor approximation algorithm is known [12]. Series-parallel graphs are equivalent to graphs of treewidth 2. For them, Jakoby and Tantau showed a logspace algorithm for Reach [15]. Das et al. extended the logspace bound to bounded treewidth graphs when the input contains the tree decomposition [10]. Elberfeld et al. showed a logspace algorithm for any monadic second order property of a logical structure of bounded treewidth [11].

1.1 Our Result

In this paper, we present a polynomial time algorithm with improved space bound for deciding reachability in directed graphs of treewidth w . In particular, we show the following result.

► **Theorem 1.** *Given a directed graph G , a tree decomposition T of G of treewidth w , and two vertices u and v in G , there is an $O(w \log n)$ space and polynomial time algorithm that decides if there is a path from u to v in G .*

Das et al. presented a logspace algorithm to solve reachability in constant treewidth graph given a tree decomposition as input [10]. Although they do not explicitly analyze the space and time bounds of their algorithm to show how it is dependent on the treewidth w of the graph, a naive analysis would show that their algorithm requires $\Omega(w^2 \log n)$ space and $n^{\Omega(w \log w)}$ time.

The graph reachability problem can also be expressed by a constant-size MSO formula. Hence the result by Elberfeld et al. [11] solves for a more general problem and implies a logspace algorithm for reachability in constant treewidth graphs. However, the space required by their algorithm is $\Omega(p(w) \log n)$ and time required is $\Omega(n^{q(w)})$ where p and q are super-linear polynomials.

It is worth noting here that both the algorithms of Elberfeld et al. [11] and Das et al. [10] cease to be polynomial time algorithms for classes of graphs whose treewidth w is not constant. These include a large number of interesting classes of graphs mentioned in the introduction. Our algorithm requires $O(w \log n)$ space and $O(\text{poly}(n, w))$ time and therefore has a better time and space complexity for solving the reachability problem when compared to the results of [11] and [10].

The notion of treewidth is intimately connected with a well-studied notion of vertex-separators in a graph. It is known that if a graph has treewidth w , then the graph contains vertex separators of size $w + 1$. To prove Theorem 1, we proceed in the following way:

- We first show that, using the input tree decomposition, vertex separators of size $w + 1$ of input graph can be constructed in $O(w \log n)$ space and polynomial time.
- Using the algorithm for vertex separator as a subroutine, we construct a new binary balanced tree decomposition of G having $O(w)$ treewidth and logarithmic depth.
- We use the new tree decomposition to solve the reachability problem.

To solve the reachability problem in the last step, we use the universal sequences of Asano et al. [5] to determine an appropriate order to process the vertices of the input graph.

Note that we use the input tree decomposition only to compute vertex separators in the graph. Hence, the method presented here gives a slightly stronger result when dealing with classes of graph where an $O(w \log n)$ space and polynomial time algorithm for finding a vertex separator of size $O(w)$ is known. For such graphs, we can waive the requirement of additional tree decomposition in the input and still get similar space and time complexity. We state this more formally in Theorem 2.

► **Theorem 2.** *Let \mathcal{G} be a class of graphs and $w : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ be a function such that for every graph $G \in \mathcal{G}$ with n vertices and m edges, treewidth of G is at most $w(n, m)$. If there exist an $O(w(n, m) \log n)$ space and polynomial time algorithm, that given a graph $G \in \mathcal{G}$ and a set U of $V(G)$, outputs a separator of U in G 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.*

For constant treewidth graphs, the result of Elberfeld et al. [11] is equipped with a logspace algorithm to construct a binary balanced tree decomposition of $O(w)$ treewidth whose depth is $O(w \log n)$. Since vertex separators can be constructed in logspace for constant treewidth graphs, our technique can be used to construct a binary balanced tree decomposition of $O(w)$ treewidth with $O(\log n)$ depth in logspace. The depth of our tree decomposition is independent of w .

1.2 Consequences of our Result

For graphs of treewidth $n^{1-\epsilon}$, for any $\epsilon > 0$, our algorithm gives an $O(n^{1-\delta})$ space and polynomial time algorithm (for some δ). For graphs of polylog treewidth, we show that reachability can be solved in polynomial time and polylog space.

Graphs which have genus g have treewidth $O((gn)^{1/2})$; hence our algorithm gives a $O((gn)^{1/2} \log n)$ space and polynomial time algorithm for it.

For planar graphs, our approach gives a $O(n^{1/2} \log n)$ space and polynomial time algorithm. Note that for planar graphs, a careful analysis of the separator construction algorithm of Imai et al. shows that one can construct a separator for planar graphs in polynomial time and $O(n^{1/2} \log n)$ space [14]. We can use this separator construction and waive the requirement of having the tree decomposition as input for planar graphs. As a result, we get the best known simultaneous time space bound for reachability in planar graphs.

Let H be a graph on h vertices. An H minor-free graph is also, by definition, K_h minor free where K_h is a complete graph on h vertices. Graphs which exclude a fixed minor K_h , have a treewidth of at most $hn^{1/2}$ [1][16]. Hence, for constant h , our approach results in $O(n^{1/2} \log n)$ space and polynomial time algorithm for H minor-free graphs.

A chordal graph on n vertices with m edges have a separator of size $O(m^{1/2})$ [13]. The proof of existence of such a separator in [13] directly leads to a $O(m^{1/2} \log n)$ space and polynomial time algorithm of constructing it. We can use this separator and obtain an $O(m^{1/2} \log n)$ space and polynomial time algorithm to solve the reachability problem in chordal graphs *without* the input tree decomposition.

1.3 Organization of the Paper

In Section 2 we give the definitions, notations and previously known results that we use in this paper. In Section 3 we show how to efficiently compute a logarithmic depth binary tree decomposition of G having a similar width from the input tree decomposition. In Section 4 we give the reachability algorithm and prove its correctness and complexity bounds.

2 Preliminaries

For a graph G on n vertices, we denote its vertex and edge sets as $V(G)$ and $E(G)$ respectively. Let W be a subset of $V(G)$. We denote the subgraph of G induced by the vertices in W by $G[W]$. Let $[n]$ denote the set $\{1, 2, \dots, n\}$ for $n \geq 1$. We assume that the vertices of an n vertex graph are indexed by integers from 1 to n .

We next define the terminology and notations related to tree decomposition that we use in this paper. For tree decomposition, we will treat the graph as an undirected graph by ignoring the direction of its edges.

For a graph G , a *tree decomposition* is a labeled tree T where the labeling function $B : V(T) \rightarrow \{X \mid X \subseteq V(G)\}$ has the following property: (i) $\bigcup_{t \in V(T)} B(t) = V(G)$, (ii) for every edge $\{v, w\}$ in $E(G)$, there exists t in $V(T)$ such that v and w are in $B(t)$, and (iii) if t_3 is on the path from t_1 to t_2 in T , then $B(t_1) \cap B(t_2) \subseteq B(t_3)$. The *treewidth* of a tree decomposition T is $\max_{t \in V(T)} (|B(t)| - 1)$. Finally the treewidth of a graph G is the minimum treewidth over all tree decompositions of G . We refer to an element t of $V(T)$ as a *node* and the set $B(t)$ to be the *bag* corresponding to t .

We assume that in a *binary tree*, every node has zero or two children. Moreover in a *balanced tree*, all paths from the root to leaves have the same length.

The next tool that we would be using is that of separators in graphs. For a subset W of $V(G)$, a *vertex separator* of W in G , is a subset S of $V(G)$ such that every component of the graph $G[V(G) \setminus S]$ has at most $|W|/2$ vertices of W .

We state in Lemma 3 the commonly known result about vertex separators in the form that we would be using it.

► **Lemma 3.** *Let G be a graph and T be a tree decomposition of G . For every subset U of $V(G)$, there exists a node t in $V(T)$ such that the bag $B(t)$ is a vertex separator of U in G .*

Proof. We root the tree arbitrarily. For a node t in $V(T)$, we denote its parent by $\text{parent}(t)$. Let $C(t) = B(t) \cap U$. We define weights on the nodes of T , such that each vertex of U is counted in exactly one of the weights. $\alpha(t) = |C(t) \setminus C(\text{parent}(t))|$. Thus, $\sum_{t \in V(T)} \alpha(t) = |U|$. In a weighted tree, there exists a node whose removal divides the tree into components whose weights are at most half the total weight of the tree. Let this node be t^* for the weight function α . We claim that $B(t^*)$ is the vertex separator for U in G . To prove this, we will prove that for a connected component H of $G[V(G) \setminus B(t^*)]$, H is a subset of $(\bigcup_{t \in T_i} B(t))$ for some subtree T_i of $T \setminus \{t^*\}$. Since the total weight of this subtree is at most half the total weight on T , it would follow that the number of vertices of $U \setminus B(t^*)$ contained in this set would be at most half, thus proving the lemma.

We will now prove that $H \subseteq (\bigcup_{t \in T_i} B(t))$. We first observe that a vertex $v \in V(G) \setminus B(t^*)$ can be in the bag of only one of the subtree, since otherwise, it would belong to $B(t^*)$ as well, due to the third property of tree decomposition. Now, let us assume that there are two vertices of H which belong to the bags of two different subtrees, say T_i and T_j . Since they are in a connected component, there will exist a path between them. In this path, there will exist an edge, whose endpoints v_1 and v_2 would belong to different subtrees. We thus get a contradiction to the second property of tree decomposition. ◀

We use a multitape Turing machine model to discuss the space-bounded polynomial time algorithms. A multitape Turing machine consists of a read-only input tape, a write-only output tape, and a constant number of work tapes. We measure the space complexity of a multitape Turing machine by the total number of bits used in the worktapes.

If we compose two polynomial time algorithms A_1 and A_2 , requiring space $S_1(n)$ and $S_2(n)$ respectively, such that the output of A_1 is used as an input to A_2 , then the total space used in composing A_1 and A_2 is $S(n) = O(S_1(n) + S_2(n))$. To see this note that whenever A_2 queries for an input bit, we simulate A_1 until it yields the desired bit, and then resume the simulation of A_2 . The total time would remain polynomial as it would be a product of two polynomials.

3 Finding a Tree Decomposition of Small Depth

In this section, we show how to compute a binary balanced tree decomposition (say T') with logarithmic depth and treewidth $O(w)$. We require such a tree decomposition because our main algorithm for reachability (Algorithm 4) might potentially store reachability information for all vertices corresponding to the bags of treenodes in a path from the root to a leaf. Once the depth is reduced to $O(\log n)$ with bag size being $O(w)$, the algorithm will only need to store reachability information of $O(w \log n)$ vertices.

Thus, we prove the following theorem.

► **Theorem 4.** *Given as input $\langle G, T \rangle$ where G is a graph and T is a tree decomposition of G with treewidth w , there exists an algorithm working simultaneously in $O(w \log n)$ space and polynomial time which outputs a binary tree decomposition T' of G which has treewidth $6w + 6$ and depth $O(\log n)$.*

We will now develop a framework that will help us to prove Theorem 4.

First, we show how to compute a vertex separator of a given set U in G in polynomial time and $O(w \log n)$ space. We cycle through every node in the tree T and store the set of vertices in $B(t)$. Doing this requires $O(w \log n)$ space. Then using Reingold's undirected reachability algorithm [17], we count the number of vertices of U in each of the components of $G[V(G) \setminus B(t)]$. By Lemma 3, at least one of these sets $B(t)$ would be a separator of U in G . Its size will be the size of $B(t)$ for some treenode t . Hence it can be at most $w + 1$. We summarise this procedure in Lemma 5.

► **Lemma 5.** *Given as input $\langle G, T, U \rangle$ where G is a graph, T is a tree decomposition of G with treewidth w , and U is a subset of $V(G)$, there exists an $O(w \log n)$ space and polynomial time algorithm that computes a vertex separator of U in G of size at most $w + 1$.*

When G and T are clear from the context, we will refer to the vertex separator of U in G that is returned by the algorithm of Lemma 5 as $\text{sep}(U)$.

3.1 Constructing a Recursive Decomposition

As an intermediate step, we construct a *recursive decomposition* of the graph which is a tree whose nodes represent a subgraph of G . The root node represents the entire G . We then remove a separator from it. We assume inductively that each of the connected components has its recursive decomposition and connect the root node to the roots of these recursive decompositions of connected components. We select a separator such that a small number of bits can encode each node. This recursive decomposition acts as an intermediate to our tree decomposition. Once we have a recursive decomposition of the graph, we add labels to each node such that it satisfies the properties of tree decomposition.

► **Definition 6.** *Let $Z \subseteq V(G)$ and a vertex $r \in (V(G) \setminus Z)$. Define $G_{(Z,r)}$ to be the subgraph of G induced by the set of vertices in the connected component of $G[V(G) \setminus Z]$ which contains r . Define the tree $\text{rdtree}(Z, r)$ which we call recursive decomposition as follows:*

12:6 Reachability in High Treewidth Graphs

- The root of $\text{rdtree}(Z, r)$ is $\langle Z, r \rangle$.
- Let $Z' = Z \cup \text{sep}(Z) \cup \text{sep}(V(G_{\langle Z, r \rangle}))$ and let r_1, \dots, r_k be the lowest indexed vertices in each of the connected components of $G[(V(G_{\langle Z, r \rangle}) \setminus Z)']$. The children of the root are roots of the recursive decompositions $\text{rdtree}(Z'_i, r_i)$ for each $i \in \{1, \dots, k\}$, where Z'_i is the set of vertices in Z' that are adjacent to at least one vertex of $V(G_{\langle Z', r_i \rangle})$ in G .

Observe that for the graph G the recursive decomposition tree structure has logarithmic depth, and we can encode a node $\langle Z, r \rangle$ using $O(|Z| \log n)$ bits.

► **Lemma 7.** *Let v_0 be a vertex in G . Then the depth of the recursive decomposition $\text{rdtree}(\emptyset, v_0)$ is at most $\log n$. Moreover, for a node $\langle Z, r \rangle$ in $\text{rdtree}(\emptyset, v_0)$, we have $|Z| \leq 4w + 4$.*

Proof. We prove a more general result that for any set of vertices $Z \subseteq V(G)$ and a vertex $r \in (V(G) \setminus Z)$, the depth of $\text{rdtree}(Z, r)$ is at most $\log n$. Let Z' be as in Definition 6. By Definition 6, the set $\text{sep}(V(G_{\langle Z, r \rangle}))$ is a subset of Z' . Hence removal of Z' divides the graph $G_{\langle Z, r \rangle}$ into components each of which is of size at most half that of the size of $G_{\langle Z, r \rangle}$. Since r_1, \dots, r_k are chosen from these components, it follows that the size of $G_{\langle Z', r_i \rangle}$ is at most half of $G_{\langle Z, r \rangle}$. Additionally, in Definition 6 the sets Z'_i are chosen in such a manner that the graphs $G_{\langle Z'_i, r_i \rangle}$ and $G_{\langle Z', r_i \rangle}$ are equivalent. This proves that the size of the graph $G_{\langle Z, r \rangle}$ halves at each level of the recursive decomposition. Hence $\text{rdtree}(Z, r)$ would have at most $\log n$ depth.

We prove the second part of the lemma by induction on the depth of the node $\langle Z, r \rangle$ in $\text{rdtree}(\emptyset, v_0)$. This is trivially true for the root. Now let $\langle Z'_i, r_i \rangle$ be a child of $\langle Z, r \rangle$. Let Z_i be the set of vertices of $Z \setminus \text{sep}(Z)$ which are adjacent to at least one of the vertices of $V(G_{\langle Z', r_i \rangle})$ in G , and let C_i be the unique connected component of $G[V(G) \setminus \text{sep}(Z)]$ whose intersection with $G_{\langle Z', r_i \rangle}$ is not empty. Since $\text{sep}(Z)$ is a separator of Z in G , C_i will contain at most $|Z|/2$ vertices of Z . This shows that $|Z_i| \leq |Z|/2$. By Definition 6, we know that $|Z'_i| \leq |Z_i| + |\text{sep}(Z)| + |\text{sep}(V(G_{\langle Z, r \rangle}))|$. The size of $\text{sep}(V(G_{\langle Z, r \rangle})) \leq w + 1$ and $\text{sep}(Z) \leq w + 1$ by Lemma 5. Lastly by induction $|Z|/2 \leq (4w + 4)/2$. Hence it follows that $|Z'_i| \leq 4w + 4$. ◀

We now show that the recursive decomposition tree corresponding to G can be computed efficiently as well. To prove this, we give procedures that, given a node in the recursive decomposition tree, can compute its parent and children efficiently.

■ **Algorithm 1** Computes the children of the node $\langle Z, r \rangle$ in $\text{rdtree}(\emptyset, v_0)$.

Input: $\langle G, T, v_0, Z, r \rangle$
Output: Children of the node $\langle Z, r \rangle$ in $\text{rdtree}(\emptyset, v_0)$

- 1 Compute $\text{sep}(Z)$ using Lemma 5
- 2 Compute $\text{sep}(V(G_{\langle Z, r \rangle}))$ using Lemma 5
- 3 Let $Z' := Z \cup \text{sep}(Z) \cup \text{sep}(V(G_{\langle Z, r \rangle}))$
- 4 **for** $v \in V(G)$ **do**
- 5 **if** $v \in V(G_{\langle Z, r \rangle})$ **and** v is smallest indexed vertex in $G_{\langle Z', v \rangle}$ **then**
- 6 Let $\widehat{Z} := \{u \in Z' \mid u \text{ is adjacent to } V(G_{\langle Z', v \rangle}) \text{ in } G\}$
- 7 Output $\langle \widehat{Z}, v \rangle$
- 8 **endif**
- 9 **endfor**

Algorithm 1 outputs the children of $\langle Z, r \rangle$ in $\text{rdtree}(\emptyset, v_0)$. Note that we don't explicitly store $V(G_{\langle Z, r \rangle})$ but compute it whenever required. That is, whenever we need to check if a vertex belongs to $V(G_{\langle Z, r \rangle})$, we check if it is connected to r in the underlying undirected graph

of $G \setminus Z$ using Reingold's algorithm. The separators in line 1 and 2 both have cardinality at most $w + 1$ and can be computed in $O(w \log n)$ space and polynomial time by Lemma 5. The cardinality of Z is at most $4w + 4$ by Lemma 7. Therefore $|Z'|$ is at most $6w + 6$. The size of \widehat{Z} computed is $4w + 4$ by Lemma 7. Thus the space required by Algorithm 1 is $O(w \log n)$.

■ **Algorithm 2** Computes the parent of the node $\langle Z, r \rangle$ in $\text{rdtree}(\emptyset, v_0)$.

Input: $\langle G, T, v_0, Z, r \rangle$
Output: parent of the node $\langle Z, r \rangle$ in $\text{rdtree}(\emptyset, v_0)$

- 1 Set $current := \langle \emptyset, v_0 \rangle$
- 2 **while** $\langle Z, r \rangle$ is not a child of $current$ **do**
- 3 Let $\langle Z', r' \rangle$ be the child of $current$ such that $G_{\langle Z', r' \rangle}$ contains r
- 4 Set $current := \langle Z', r' \rangle$
- 5 **end**
- 6 Output $current$

Algorithm 2 outputs the parent of $\langle Z, r \rangle$ in $\text{rdtree}(\emptyset, v_0)$. It uses Algorithm 1 as a subroutine to get the children of a node in $\text{rdtree}(\emptyset, v_0)$. Hence we can traverse the tree $\text{rdtree}(\emptyset, v_0)$ in $O(w \log n)$ space and polynomial time. We summarize the above in Lemma 8.

► **Lemma 8.** *Let G be a graph, T be a tree decomposition of G with treewidth w and v_0 be a vertex in G . Given $\langle G, T, v_0 \rangle$ and the node $\langle Z, r \rangle$ in $\text{rdtree}(\emptyset, v_0)$, there exist algorithms that use $O(w \log n)$ space and polynomial time, and output the children and parent of $\langle Z, r \rangle$ respectively. As a consequence $\text{rdtree}(\emptyset, v_0)$ can be traversed in $O(w \log n)$ space and polynomial time as well.*

3.2 Constructing a New Tree Decomposition

We now construct a new tree decomposition of G from the recursive decomposition defined earlier. The new tree decomposition will have the same tree structure as that of the recursive decomposition. However, we will assign it a labeling function. The subgraph that a node of the recursive decomposition represents is a connected component obtained after removing a set of separators from G . The corresponding label for this node in the new tree decomposition is simply the set of separator vertices in the boundary of this subgraph together with the separator required to subdivide this subgraph further. We formalize this in Definition 9.

► **Definition 9.** *Let \widehat{T} be the tree corresponding to the recursive decomposition $\text{rdtree}(\emptyset, v_0)$. For a node $\langle Z, r \rangle$ in $\text{rdtree}(\emptyset, v_0)$, we define the function $\widehat{B}(\langle Z, r \rangle)$ as, $\widehat{B}(\langle Z, r \rangle) := Z \cup ((\text{sep}(V(G_{\langle Z, r \rangle})) \cup \text{sep}(Z)) \cap V(G_{\langle Z, r \rangle}))$.*

We first show that \widehat{T} is a tree decomposition of G as well, with labeling function \widehat{B} .

► **Lemma 10.** *The tree \widehat{T} defined in Definition 9 along with the labeling function \widehat{B} , is a tree decomposition of G of width $6w + 6$. Moreover, the depth of \widehat{T} is at most $\log n$.*

Proof. We claim that for a node v in $G_{\langle Z, r \rangle}$, there exists a vertex $\langle Z', r' \rangle$ in $\text{rdtree}(Z, r)$ such that $\widehat{B}(\langle Z', r' \rangle)$ contains v . We prove this by induction on the depth of the recursive decomposition $\text{rdtree}(Z, r)$. If $\text{rdtree}(Z, r)$ is just a single node, then v is in $\text{sep}(V(G_{\langle Z, r \rangle}))$ by construction. Otherwise v is either in $(\text{sep}(Z) \cup \text{sep}(V(G_{\langle Z, r \rangle})))$ or in one of the connected components of $G[V(G_{\langle Z, r \rangle}) \setminus (\text{sep}(Z) \cup \text{sep}(V(G_{\langle Z, r \rangle})))]$. If v is in $(\text{sep}(Z) \cup \text{sep}(V(G_{\langle Z, r \rangle})))$, then v is in $\widehat{B}(\langle Z, r \rangle)$ and we are done. Otherwise one of the children of $\langle Z, r \rangle$ will be

$\langle \tilde{Z}, \tilde{r} \rangle$ such that v is in $G_{\langle \tilde{Z}, \tilde{r} \rangle}$. Now by induction hypothesis, there exists a vertex $\langle Z', r' \rangle$ in $\text{rdtree}(\tilde{Z}, \tilde{r})$ such that $\hat{B}(\langle Z', r' \rangle)$ contains v . It follows that every vertex v of $V(G)$ is contained in the label of at least one of the vertices of \hat{T} , satisfying the first property of tree decomposition.

We claim that for any edge (u, v) in G such that $\{u, v\} \subseteq V(G_{\langle Z, r \rangle}) \cup Z$, either both u and v are in $\hat{B}(\langle Z, r \rangle)$ or there exists a child $\langle Z'_i, r_i \rangle$ of $\langle Z, r \rangle$ such that $\{u, v\} \subseteq V(G_{\langle Z'_i, r_i \rangle}) \cup Z'_i$. Since u and v are connected by an edge, there cannot exist any set of vertices \hat{Z} such that u and v are in different connected components of $G[V(G) \setminus \hat{Z}]$. Let $Z' = Z \cup \text{sep}(Z) \cup \text{sep}(V(G_{\langle Z, r \rangle}))$. If both u and v are in Z' , then they are in $\hat{B}(\langle Z, r \rangle)$. Otherwise, let r_i be the lowest indexed vertex in the connected component of $G[(V(G_{\langle Z, r \rangle}) \setminus Z) \cup Z']$ which contains either of u or v . Let Z'_i be the set of vertices in Z' that are adjacent to at least one of the vertices of $V(G_{\langle Z'_i, r_i \rangle})$ in G . Now, if both u and v are not in $V(G_{\langle Z'_i, r_i \rangle})$, then one of them have to be in Z'_i . Hence in all cases, u and v are contained in $V(G_{\langle Z'_i, r_i \rangle}) \cup Z'_i$. Hence by induction on the depth of the tree decomposition \hat{T} we have that there exists a treenode in \hat{T} whose bag contains both u and v , satisfying the second property of tree decomposition.

To establish the third property of tree decomposition we first show that if v is not in $Z \cup V(G_{\langle Z, r \rangle})$, then for no descendant $\langle \tilde{Z}, \tilde{r} \rangle$ of $\langle Z, r \rangle$ will $\hat{B}(\langle \tilde{Z}, \tilde{r} \rangle)$ contain v . We show this by induction on the depth of the recursive decomposition. If there is only one node in $\text{rdtree}(Z, r)$, then $\hat{B}(\langle Z, r \rangle)$ does not contain v by definition. Otherwise, no connected component of $G[V(G_{\langle Z, r \rangle}) \setminus Z']$ contains v . Also Z'_i for any of its children will not contain v as claimed.

Now let $\langle Z, r \rangle$ be a treenode in \hat{T} . We claim that for any child $\langle Z'_i, r_i \rangle$ of $\langle Z, r \rangle$ if a vertex v is in $\hat{B}(\langle Z, r \rangle)$, then either v is also in $\hat{B}(\langle Z'_i, r_i \rangle)$ or no descendant of $\langle Z'_i, r_i \rangle$ has a bag corresponding to it which contains v . Since any connected component of $G[V(G_{\langle Z, r \rangle}) \setminus \hat{B}(\langle Z, r \rangle)]$ cannot contain v , v is not in $V(G_{\langle Z'_i, r_i \rangle})$ for any child $\langle Z'_i, r_i \rangle$ of $\langle Z, r \rangle$. Now if v is not in $\hat{B}(\langle Z'_i, r_i \rangle)$, then it implies that v is not in $Z'_i \cup V(G_{\langle Z'_i, r_i \rangle})$ as well. Hence the third property of tree decomposition is satisfied as well.

For a vertex $\langle Z, r \rangle$ in $\text{rdtree}(\emptyset, v_0)$, we have $|Z| \leq 4w + 4$, $\text{sep}(Z) \leq w + 1$ and $\text{sep}(V(G_{\langle Z, r \rangle})) \leq w + 1$ as well. Hence $\hat{B}(\langle Z, r \rangle) \leq 6w + 6$.

Since the tree \hat{T} and $\text{rdtree}(\emptyset, v_0)$ have the same structure, the bounds on their depths are the same. \blacktriangleleft

Next, we observe that given $\langle Z, r \rangle$, we can compute $\hat{B}(\langle Z, r \rangle)$ in $O(w \log n)$ space and polynomial time. Hence we have the following Lemma.

► **Lemma 11.** *Given a graph G and a tree decomposition T of G with treewidth w , there is an algorithm that can compute a new tree decomposition \hat{T} of G having treewidth at most $6w + 6$ and depth at most $\log n$, using $O(w \log n)$ space and polynomial time. Moreover, the tree \hat{T} can be traversed in $O(w \log n)$ space and polynomial time as well.*

Note that the tree \hat{T} might not be a binary tree since a separator might disconnect the graph into more than two components. However, to decide reachability in the later part of this paper, we require the tree decomposition to have bounded degree as well. We achieve this by using the following lemma from Elberfeld et al. to get the required tree decomposition T' .

► **Lemma 12** ([11]). *There is a logspace algorithm that on the input of any logarithmic depth tree decomposition of a graph G outputs a logarithmic depth, binary balanced tree decomposition of G having the same treewidth.*

Now combining Lemma 11 and Lemma 12 we get the proof of Theorem 4.

We observe here that the input tree decomposition T is used only to compute a vertex separator in G .

The requirement of input tree decomposition can be waived for those classes of graphs where vertex separators can be constructed in a space efficient manner. For example, in planar graphs [14] and chordal graphs [13], we can use their respective separator algorithms as subroutines instead of the algorithm of Lemma 5 in lines 1 and 2 of the Algorithm 1.

4 Deciding Reachability using a Binary Balanced Tree Decomposition

In this section, we show that given a graph G along with a binary balanced tree decomposition T whose depth is $O(\log n)$; there exists an efficient algorithm to decide reachability in G in $O(w \log n)$ space and polynomial time. In particular, we show the following theorem.

► **Theorem 13.** *Given $\langle G, T, u, v \rangle$ as input, where G is a graph on n vertices, u and v are two vertices of G , and T is a binary balanced tree decomposition of G having depth h and treewidth w , there exists an $O(wh + \log n)$ space and $O(\text{poly}(2^h, w, n))$ time algorithm that solves reachability in G .*

We first state the notation required to prove Theorem 13. This notation is commonly used to describe dynamic programming algorithms which use tree decomposition. Let T be a rooted binary tree. We denote $\text{root}(T)$ to be the root of T and for a node $t \in T$, we denote $\text{left}(t)$ and $\text{right}(t)$ to be the left and right child of t respectively (the value is NULL if a child does not exist). For two nodes t and t' in T , if t' lies in the path from $\text{root}(T)$ to t , then we say that t' is an *ancestor* of t and t is a *descendent* of t' . For a treenode t , let $B_e(t)$ denote the set of edges of G whose both endpoints are in $B(t)$. We define a subgraph of G with respect to the treenode t consisting of the ancestor vertices of t . Formally, the vertex set is $V_t^{\text{anc}} = \bigcup_{\{t' \text{ is an ancestor of } t\}} B(t')$, and the edge set is $E_t^{\text{anc}} = \bigcup_{\{t' \text{ is an ancestor of } t\}} B_e(t')$ and the graph $G_t^{\text{anc}} = (V_t^{\text{anc}}, E_t^{\text{anc}})$. Now, we define a subgraph of G with respect to the treenode t consisting of the ancestor as well as descendent vertices of t . Formally, the vertex set is $V_t = \bigcup_{\{t' \text{ is an ancestor or descendent of } t\}} B(t')$, the edge set is $E_t = \bigcup_{\{t' \text{ is an ancestor or descendent of } t\}} B_e(t')$ and the graph $G_t = (V_t, E_t)$.

We assume that the vertices u and v are in $\text{root}(T)$, for otherwise, we can add them in all of the bags of the given tree decomposition. Also, we assume that n is a power of 2.

We now explain our reachability algorithm. For a node t in the tree decomposition T consider the graph G_t . Let P be a path of length d from a vertex of V_t^{anc} to another (assume without loss of generality that d a power of 2). We define a sequence of leaves $\text{SEQ}_{t,d}$ of T (see Section 4.1). Each leaf f in this sequence corresponds to a set of at most wh vertices V_f . Now subdivide the path P into subpaths P_1, P_2, \dots, P_k such that each P_i completely lies either in $G_{\text{left}(t)}$ or in $G_{\text{right}(t)}$. We now use the sequence $\text{SEQ}_{t,d}$ to give an iterative procedure to combine the results of the subpaths P_i 's to determine the path P . In Algorithm 4 we show how to use the sequence $\text{SEQ}_{t,d}$ to simulate the described method. We show in Lemma 18 that processing $\text{SEQ}_{t,d}$ is sufficient to determine a path of length at most d between two vertices in the graph G_t .

4.1 Constructing the Sequence $\text{SEQ}_{t,d}$

We will be using *universal sequences* and the following lemma about it from Asano et al. to construct the sequence of leaves.

12:10 Reachability in High Treewidth Graphs

For every integer $s \geq 0$, a *universal sequence* σ_s of length $2^{s+1} - 1$ is defined as follows:

$$\sigma_s = \begin{cases} \langle 1 \rangle & s = 0 \\ \sigma_{s-1} \diamond \langle 2^s \rangle \diamond \sigma_{s-1} & s > 0 \end{cases}$$

where \diamond is the concatenation operation.

► **Lemma 14** ([5]). *The universal sequence σ_s satisfies the following properties:*

- Let $\sigma_s = \langle c_1, \dots, c_{2^{s+1}-1} \rangle$. Then for any positive integer sequence $\langle d_1, \dots, d_x \rangle$ such that $\sum d_i \leq 2^s$, there exists a subsequence $\langle c_{i_1}, \dots, c_{i_x} \rangle$ such that $d_j \leq c_{i_j}$ for all $j \in [x]$.
- The sequence σ_s contains exactly 2^{s-i} appearances of the integer 2^i and nothing else.
- The sequence σ_s is computable in $O(2^s)$ time and $O(s)$ space.

► **Definition 15.** Let T be a binary balanced tree. Let t be a node in T and d be a positive power of 2. We define a sequence $SEQ_{t,d}$ consisting of leaves of T in the following way: If t is not a leaf then $SEQ_{t,d} = SEQ_{\text{left}(t),c_1} \diamond SEQ_{\text{right}(t),c_1} \diamond SEQ_{\text{left}(t),c_2} \diamond SEQ_{\text{right}(t),c_2} \diamond \dots \diamond SEQ_{\text{right}(t),c_{2^d-1}}$ where c_i is the i -th integer in $\sigma_{\log d}$. Otherwise, if t is a leaf, $SEQ_{t,d}$ is $\langle t \rangle$ concatenated with itself d times. We also define $SEQ_{t,d}(r)$ to be the leaf at the index r in the sequence $SEQ_{t,d}$. The length of $SEQ_{t,d}$ is the number of leaves in $SEQ_{t,d}$.

We show in Algorithm 3 how to construct the sequence $SEQ_{\text{root}(T),d}$ in $O(h + \log d)$ space. In Lemma 16 we give a closed form expression for the length of the sequence.

Algorithm to Compute $SEQ_{t,d}$

■ **Algorithm 3** Computes the r -th element of the sequence $SEQ_{t,d}$.

Input: $\langle t, d, r \rangle$

- 1 **while** t is not a leaf **do**
- 2 Let m be the depth of the subtree of T rooted at t
- 3 Let i^* be the smallest integer such that $(r - 2 \sum_{i=1}^{i^*} L(m/2, c_i)) \leq 0$ where c_i is the i 'th integer in the sequence $\sigma_{\log d}$
- 4 **if** $r - 2 \sum_{i=1}^{i^*-1} L(m/2, c_i) - L(m/2, c_{i^*}) \leq 0$ **then**
- 5 $r \leftarrow r - 2 \sum_{i=1}^{i^*-1} L(m/2, c_i)$
- 6 $t \leftarrow \text{left}(t)$
- 7 **else**
- 8 $r \leftarrow r - 2 \sum_{i=1}^{i^*-1} L(m/2, c_i) - L(m/2, c_{i^*})$
- 9 $t \leftarrow \text{right}(t)$
- 10 **endif**
- 11 $d \leftarrow c_{i^*}$
- 12 **end**
- 13 **return** t

► **Lemma 16.** Let T be a binary balanced tree. Let t be a node in T , d be a positive power of 2 and h be the depth of subtree of T rooted at t . Then, the length of sequence $SEQ_{t,d}$ is $2^h d^{\binom{h+\log d}{\log d}}$.

Proof. Let $L(h, d)$ be the length of the sequence $SEQ_{t,d}$. By definition of $SEQ_{t,d}$, we have

$$L(h, d) = \begin{cases} 2 \sum_{c \in \sigma_{\log d}} L(h-1, c) & h > 0 \\ d & h = 0 \end{cases}$$

From lemma 14, we get that $\sigma_{\log d}$ contains exactly $\frac{d}{2^i}$ occurrences of the integer 2^i . Thus we have:

$$L(h, d) = \begin{cases} \sum_{i=0}^{\log d} \frac{d}{2^{i-1}} L(h-1, 2^i) & h > 0 \\ d & h = 0 \end{cases}$$

We claim that $L(h, d) = 2^h d \binom{h+\log d}{\log d}$ and we prove this by induction on h . For $h = 0$, we see that

$$\begin{aligned} 2^h d \binom{h+\log d}{\log d} &= d \binom{\log d}{\log d} \\ &= d \end{aligned}$$

Now, we assume the statement to be true for smaller values of h . We see that:

$$\begin{aligned} L(h, d) &= \sum_{i=0}^{\log d} \frac{d}{2^{i-1}} L(h-1, 2^i) \\ L(h, d) &= \sum_{i=0}^{\log d} \frac{d}{2^{i-1}} 2^{h-1} 2^i \binom{h+i-1}{i} \\ L(h, d) &= 2^h d \sum_{i=0}^{\log d} \binom{h+i-1}{i} \end{aligned}$$

using $\binom{a}{r} = \binom{a+1}{r} - \binom{a}{r-1}$

$$\begin{aligned} L(h, d) &= 2^h d \sum_{i=0}^{\log d} \left(\binom{h+i}{i} - \binom{h+i-1}{i-1} \right) \\ L(h, d) &= 2^h d \binom{h+\log d}{\log d} \end{aligned}$$

► **Lemma 17.** *Let T be a binary balanced tree of depth at most h . Let t be a node of T and d be a power of 2. The sequence $SEQ_{t,d}$ can be constructed in space $O(h + \log d)$.*

Proof. We see that $L(m, d)$ is bounded by a polynomial in m and d . For a given integer r , let i^* be the smallest integer such that $r - 2 \sum_{i=1}^{i^*} L(m/2, c_i) \leq 0$. By the definition, $SEQ_{t,d}(r) = SEQ_{\text{left}(t), c_{i^*}}(r - 2 \sum_{i=1}^{i^*-1} L(m/2, c_i))$ if $r - 2 \sum_{i=1}^{i^*-1} L(m/2, c_i) - L(m/2, c_{i^*}) \leq 0$ and $SEQ_{t,d}(r) = SEQ_{\text{right}(t), c_{i^*}}(r - 2 \sum_{i=1}^{i^*-1} L(m/2, c_i) - L(m/2, c_{i^*}))$ otherwise.

The length of the sequence $SEQ_{t,d}$ is at most $2^h d \binom{h+\log d}{\log d}$. Hence the number of bits required to store any index of the sequence is at most $\log(2^h d \binom{h+\log d}{\log d}) = O(h + \log d)$. This gives the space bound of Algorithm 3. ◀

4.2 Algorithm to Solve Reachability

For a leaf t of T and a vertex v of G we use $\text{pos}_t(v)$ for the position of v in an arbitrarily fixed ordering of the vertices of G_t .

► **Lemma 18.** *Let G be a graph and T be a binary tree decomposition of G of width w and depth h . Let t be a node of T and d be a power of 2. For each vertex $y \in V_t^{\text{anc}}$, y is marked after the execution of iterations in lines 4 to 13 of Algorithm 4 with values of f in $SEQ_{t,d}$ if there is a marked vertex x in V_t^{anc} and a path from x to y in G_t of length at most d .*

12:12 Reachability in High Treewidth Graphs

■ **Algorithm 4** $\text{Reach}(G, T, u, v)$.

Input: $\langle G, T, v, u \rangle$

- 1 Let R_0 be and R_1 be two wh bit-vectors
- 2 Initialize t_0 and t_1 by two arbitrary leaves of T
- 3 Initialize all the bits of R_0 with 0 and mark u (by setting the bit at position $\text{pos}_{t_0}(u)$ to 1)
- 4 **for** every leaf f in $\text{SEQ}_{\text{root}(T), n}$ in order **do**
- 5 Let i be the iteration number
- 6 Reset all the bits of $R_{i \bmod 2}$ to 0
- 7 Let $t_{i \bmod 2} \leftarrow f$
- 8 **for** all x marked in $R_{(i-1) \bmod 2}$ and all y in V_f **do**
- 9 **if** (x, y) is an edge in G OR $x = y$ **then**
- 10 Mark y in $R_{i \bmod 2}$ (by setting the bit at position $\text{pos}_{t_{i \bmod 2}}(y)$ to 1)
- 11 **endif**
- 12 **endfor**
- 13 **endfor**
- 14 If v is marked return 1; otherwise return 0.

Proof. We prove this by induction on the depth of subtree rooted at t . The base case is trivial. Let p be the path of length at most d from x to y such that x is marked and x, y is in V_t^{anc} . We see that the edges of path p will belong to either $E_{\text{left}(t)}$ or $E_{\text{right}(t)}$ (or both). We label an edge of p as 0 if it belongs to $E_{\text{left}(t)}$, else label it as 1. Break down p into subpaths p_1, \dots, p_k such that the edges in p_i all have same label and label of edges in p_{i+1} is different from label of p_i . The endpoints y_i of these subpaths will belong to V_t^{anc} , for otherwise y_i will not be in $B(t)$ but since y_i has edges of both labels incident on it, it will be in bags of both subtrees rooted at $\text{left}(t)$ and $\text{right}(t)$ contradicting the third property of tree decomposition. Let l_i be the length of path p_i . Since $l_1 + l_2 + \dots + l_k \leq d$, by Lemma 14, there exists a subsequence $\langle c_{i_1}, c_{i_2}, \dots, c_{i_k} \rangle$ of $\sigma_{\log d}$ such that $l_j \leq c_{i_j}$.

Consider the subsequence $\text{SEQ}_{\text{left}(t), c_{i_1}} \diamond \text{SEQ}_{\text{right}(t), c_{i_1}} \diamond \text{SEQ}_{\text{left}(t), c_{i_2}} \diamond \text{SEQ}_{\text{right}(t), c_{i_2}} \diamond \text{SEQ}_{\text{left}(t), c_{i_3}} \diamond \text{SEQ}_{\text{right}(t), c_{i_3}} \diamond \dots \diamond \text{SEQ}_{\text{left}(t), c_{i_k}} \diamond \text{SEQ}_{\text{right}(t), c_{i_k}}$ of $\text{SEQ}_{t, d}$. We claim that y_j is marked after the iterations with the value of f in $\text{SEQ}_{\text{left}(t), c_{i_j}} \diamond \text{SEQ}_{\text{right}(t), c_{i_j}}$. Since y_{j-1} is marked before the iterations and the path p_j is either the subgraph $G_{\text{left}(t)}$ or $G_{\text{right}(t)}$ having length at most c_{i_j} , y_j will be marked by induction hypothesis. We see that any vertex present in V_t^{anc} is present in $G_{t'}$ for all leaves t' that is present in $\text{SEQ}_{t, d}$. Therefore, once such a vertex is marked, it remains marked for the rest of these iterations. Hence, y_j is marked before the iterations with the value of f in $\text{SEQ}_{\text{left}(t), c_{i_{j+1}}} \diamond \text{SEQ}_{\text{right}(t), c_{i_{j+1}}}$ ◀

► **Lemma 19.** *On input of a graph G with n vertices and its tree decomposition T with treewidth w and depth h ; Algorithm 4 solves reachability in G and requires $O(wh + \log n)$ space and time polynomial in $2^h, n$ and w .*

Proof. Algorithm 4 marks a vertex only if it is reachable from u . The proof of correctness of the algorithm follows from Lemma 18 and the fact that u and v are both present in $B(\text{root}(T))$ and u is marked before the first iteration of the for-loop in line 4.

We first analyze the space required. The size of bit-vectors R_0 and R_1 is wh . t_0 and t_1 are indices of nodes of T . The space required to store index of a vertex of T is $O(h)$. Space required to store a vertex of G is $O(\log n)$, and $\text{pos}_t(x)$ for a node t and a vertex x can be found in $O(\log n + h)$ space. Hence the total space required is $O(wh + \log n)$.

We now analyze the time bound. By Lemma 16, the size of $\text{SEQ}_{t,d}$ is polynomial in 2^h and d , the number of iterations in the for-loop of line 4 is thus a polynomial. The other lines do trivial stuff, and hence, the total running time of the algorithm is polynomial. ◀

Theorem 13 follows from Lemma 19. Combining Theorem 13 and Theorem 4 we get the proof of Theorem 1. Theorem 2 follows in a similar way. We use the separator algorithm which exists due to the hypothesis of Theorem 2 as subroutines instead of the algorithm of Lemma 5 in lines 1 and 2 of the Algorithm 1. The rest of the analysis is similar.


References

- 1 Noga Alon, Paul Seymour, and Robin Thomas. A separator theorem for nonplanar graphs. *Journal of the American Mathematical Society*, 3(4):801–808, 1990.
- 2 Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of Finding Embeddings in a k-Tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.
- 3 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.
- 4 Tetsuo Asano and Benjamin Doerr. Memory-Constrained Algorithms for Shortest Path Problem. In *Proceedings of the 23rd Annual Canadian Conference on Computational Geometry (CCCG 2011)*, 2011.
- 5 Tetsuo Asano, David Kirkpatrick, Kotaro Nakagawa, and Osamu Watanabe. $\tilde{O}(\sqrt{n})$ -Space and Polynomial-Time Algorithm for Planar Directed Graph Reachability. In *Proceedings of the 39th International Symposium on Mathematical Foundations of Computer Science (MFCS 2014)*, pages 45–56, 2014.
- 6 Greg Barnes, Jonathan F. Buss, Walter L. Ruzzo, and Baruch Schieber. A Sublinear Space, Polynomial Time Algorithm for Directed s-t Connectivity. *SIAM Journal on Computing*, 27(5):1273–1282, 1998.
- 7 Hans L. Bodlaender and Arie M. C. A. Koster. Combinatorial Optimization on Graphs of Bounded Treewidth. *The Computer Journal*, 51(3):255–269, 2008.
- 8 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 *Proceedings of the 34th Annual Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS 2014)*, pages 585–595, 2014.
- 9 Diptarka Chakraborty and Raghunath Tewari. An $O(n^\epsilon)$ Space and Polynomial Time Algorithm for Reachability in Directed Layered Planar Graphs. *ACM Transactions on Computation Theory (TOCT)*, 9(4):19:1–19:11, 2017.
- 10 Bireswar Das, Samir Datta, and Prajakta Nimbhorkar. Log-Space Algorithms for Paths and Matchings in k-Trees. *Theory of Computing Systems*, 53(4):669–689, 2013.
- 11 Michael Elberfeld, Andreas Jakoby, and Till Tantau. Logspace Versions of the Theorems of Bodlaender and Courcelle. In *Proceedings of the 51st Annual Symposium on Foundations of Computer Science (FOCS 2010)*, pages 143–152. IEEE Computer Society, 2010.
- 12 Uriel Feige, MohammadTaghi Hajiaghayi, and James R. Lee. Improved Approximation Algorithms for Minimum Weight Vertex Separators. *SIAM Journal on Computing*, 38(2):629–657, 2008.
- 13 J. Gilbert, D. Rose, and A. Edenbrandt. A Separator Theorem for Chordal Graphs. *SIAM Journal on Algebraic Discrete Methods*, 5(3):306–313, 1984.
- 14 Tatsuya Imai, Kotaro Nakagawa, Aduri Pavan, N. V. Vinodchandran, and Osamu Watanabe. An $O(n^{\frac{1}{2}+\epsilon})$ -Space and Polynomial-Time Algorithm for Directed Planar Reachability. In *Proceedings of the 28th Conference on Computational Complexity, CCC 2013*, pages 277–286, 2013.
- 15 Andreas Jakoby and Till Tantau. Logspace Algorithms for Computing Shortest and Longest Paths in Series-Parallel Graphs. In *Proceedings of the 27th Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2007)*, pages 216–227, 2007.

12:14 Reachability in High Treewidth Graphs

- 16 Ken-ichi Kawarabayashi and Bruce Reed. A separator theorem in minor-closed classes. In *Proceedings of the 51st Annual Symposium on Foundations of Computer Science (FOCS 2010)*, pages 153–162. IEEE, 2010.
- 17 Omer Reingold. Undirected connectivity in log-space. *Journal of the ACM (JACM)*, 55(4):17, 2008.
- 18 Walter J Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.
- 19 Avi Wigderson. The complexity of graph connectivity. In *Proceedings of the 17th International Symposium on Mathematical Foundations of Computer Science (MFCS 1992)*, pages 112–132. Springer, 1992.

Approximate Pricing in Networks: How to Boost the Betweenness and Revenue of a Node

Ruben Brokkelkamp 

Centrum Wiskunde & Informatica (CWI), Amsterdam, Netherlands
Ruben.Brokkelkamp@cwi.nl

Sven Polak 

Korteweg-de Vries Institute for Mathematics, University of Amsterdam, Netherlands
sven_polak@hotmail.com

Guido Schäfer

Centrum Wiskunde & Informatica (CWI), Amsterdam, Netherlands
Vrije Universiteit Amsterdam, Netherlands
g.schaefer@cwi.nl

Yllka Velaj

ISI Foundation, Turin, Italy
yllka.velaj@isi.it

Abstract

We introduce and study two new pricing problems in networks: Suppose we are given a directed graph $G = (V, E)$ with non-negative edge costs $(c_e)_{e \in E}$, k commodities $(s_i, t_i, w_i)_{i \in [k]}$ and a designated node $u \in V$. Each commodity $i \in [k]$ is represented by a source-target pair $(s_i, t_i) \in V \times V$ and a demand $w_i > 0$, specifying that w_i units of flow are sent from s_i to t_i along shortest s_i, t_i -paths (with respect to $(c_e)_{e \in E}$). The demand of each commodity is split evenly over all shortest paths. Assume we can change the edge costs of some of the outgoing edges of u , while the costs of all other edges remain fixed; we also say that we *price* (or *tax*) the edges of u .

We study the problem of pricing the edges of u with respect to the following two natural objectives: (i) *max-flow*: maximize the total flow passing through u , and (ii) *max-revenue*: maximize the total revenue (flow times tax) through u . Both variants have various applications in practice. For example, the max flow objective is equivalent to maximizing the *betweenness centrality* of u , which is one of the most popular measures for the influence of a node in a (social) network. We prove that (except for some special cases) both problems are NP-hard and inapproximable in general and therefore resort to approximation algorithms. We derive approximation algorithms for both variants and show that the derived approximation guarantees are best possible.

2012 ACM Subject Classification Theory of computation \rightarrow Approximation algorithms analysis; Theory of computation \rightarrow Graph algorithms analysis; Theory of computation \rightarrow Shortest paths; Theory of computation \rightarrow Network flows

Keywords and phrases Network pricing, Stackelberg network pricing, betweenness centrality, revenue maximization

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.13

Funding Part of this research was funded by the NWO Gravitation Project NETWORKS, Grant Number 024.002.003.

1 Introduction

Background and motivation. Nowadays, complex networks are used to model many different real-world scenarios and the analysis of these networks has become an extremely active research area. One of the main issues in complex network analysis is to identify the most “important” nodes in a network. To this aim, researchers have defined several centrality



© Ruben Brokkelkamp, Sven Polak, Guido Schäfer, and Yllka Velaj;
licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 13; pp. 13:1–13:15

Leibniz International Proceedings in Informatics



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

measures to capture different notions of importance. One of the most popular measures is *betweenness centrality*, which ranks the nodes according to their frequency of occurrence on shortest paths between all possible pairs of nodes.

In several scenarios, having a high centrality can have a positive impact on the node itself. For example, in the context of social networks, Mahmoody et al. [22] show experimentally that nodes with high betweenness are also nodes that are highly influential when spreading information to other nodes in a social network. Valente and Fujimoto [33] claim that users with a high betweenness centrality (also called “brokers” or “bridging individuals”) “may be more effective at changing others, more open to change themselves, and intrinsically interesting to identify”. Moreover, in the field of transportation network analysis Malighetti et al. [23] analyze a network of 57 European airports and find that the betweenness centrality seems to be positively correlated to the efficiency of an airport. Also, increasing the betweenness of an airport would mean more traffic flowing through it and possibly attracting more customers for its shops.

The betweenness centrality notion can also be used to investigate problems arising in economics. For example, in the Netherlands it is an active debate whether the country is a tax haven for multinational enterprises. News articles headlined “*The Netherlands is a tax haven for many multinationals*” [14], “*The Netherlands is an attractive tax country*” [1], “*Dutch masters of tax avoidance*” [32] seem to lend support to the claim that this is indeed the case. To further investigate this question, the CPB (Netherlands Bureau for Economic Policy Analysis) has recently conducted some network analysis to identify important countries (using betweenness centrality) in the international tax treaty network [35, 36]; see also [30]. Among others, they conclude that companies mainly use the Netherlands as an intermediary country to send money through on a route from one country to another one. In this sense, the Netherlands is not a *tax haven*, i.e., a destination country where the money is stored (like the Bahamas or Bermuda), but a *conduit country*, i.e., an intermediary country on a route via which companies send their money.

In light of the above insight, a natural question that arises is how a country could maximize the amount of money that is sent through it. As a result, this would attract more jobs in the financial sector, or incentivize foreign companies to establish their businesses in the country (if only in the form of a letterbox). Another conceivable objective of a country might be to maximize the total amount of taxes that it obtains from the money transfers through it. These two questions constitute the main motivation for the network pricing problems studied in this paper.

Our contributions. In this paper, we introduce and study the following *Network Pricing Problem (NPP)*: We are given a directed graph $G = (V, E)$ with non-negative edge costs $(c_e)_{e \in E}$, k commodities $(s_i, t_i, w_i)_{i \in [k]}$, a designated node $u \in V$ and a number $\kappa \geq 1$. Each commodity $i \in [k]$ is represented by a source-target pair $(s_i, t_i) \in V \times V$ and a demand $w_i > 0$, specifying that w_i units of flow are sent from s_i to t_i along shortest s_i, t_i -paths (with respect to $(c_e)_{e \in E}$). The demand of each commodity i is split evenly over all shortest s_i, t_i -paths. Suppose we can change the costs of $\kappa \leq \Delta(u)$ outgoing edges of u , where $\Delta(u)$ is the outdegree of u , while the costs of all other edges remain fixed; we also say that we *price* (or *tax*) the edges of u . Our goal is to optimally price at most κ edges of u such that (i) the total flow passing through u is maximized (**FLOW-NPP**), or (ii) the total revenue (i.e., flow times tax) through u is maximized (**REV-NPP**).

As it turns out, the problems behave rather differently in terms of hardness and approximability, depending on the objective under consideration and the parameter κ . More specifically, our main findings in this paper are as follows:

1. We show that **FLOW-NPP** can be solved in polynomial time when a constant number of edges or almost all edges of u can be priced.
2. In contrast, we prove that **FLOW-NPP** is NP-hard and $(1 - 1/e)$ -inapproximable (even for the special case of unit demands) if κ is part of the input. Further, we show that a natural greedy algorithm achieves an approximation guarantee of $(1 - 1/e)$ (which is best possible).
3. We show that **REV-NPP** can be solved in polynomial time when only one edge can be priced. On the other hand, **REV-NPP** becomes NP-hard and $(1 - 1/e)$ -inapproximable if κ is part of the input. We also show that the greedy algorithm might perform arbitrarily bad in this case.
4. We prove that **REV-NPP** is highly (meaning $1/\Delta(u)^\epsilon$) inapproximable if all outgoing edges of u can be priced. We therefore focus on special cases of this problem.
 - First, we show that the single-commodity case is polynomial time solvable. This result also constitutes an important building block for our *uniform pricing algorithms* (i.e., all edges are priced the same).
 - Then, we focus on the unit demand setting and derive a (tight) H_k -approximate uniform pricing algorithm. We complement this result by showing that this problem is $1/\log^\epsilon(k)$ -inapproximable.
5. Finally, we show that our uniform pricing algorithm extends to the general setting and provides a $\max\{1/k, 1/\Delta(u)\}$ -approximation algorithm for **REV-NPP** (which is essentially best possible).

Our results for **FLOW-NPP** mostly follow by using standard arguments for submodular function maximization. In contrast, we need to establish several new ideas and exploit structural insights to derive our results for **REV-NPP** (which constitutes the main technical contributions of this work).

We conclude with some (preliminary) experimental findings on an international tax treaty network based on real data. Our experiments indicate that our uniform pricing algorithm computes tax rates that would significantly increase the current tax revenue of the Netherlands (by a factor 68) and is at least within 51% of the optimal revenue (which is much better than the worst-case approximation guarantee suggests).

Related work. The problem of increasing the centrality of a node in a network has been widely investigated for different centrality measures. For example, boosting the popularity of web pages by increasing their page rank has been studied intensively [2, 26] with a particular focus on “fooling” search engines (e.g., through link farming [37]). The problem has also been considered for other centrality measures such as closeness centrality [11, 12], betweenness centrality [4], coverage centrality [13], eccentricity [15, 29], average distance [24] and some measures related to the number of paths passing through a given node [20]. Below, we give a few representative references only; most of these works focus on edge additions to increase the centrality.

Meyerson and Tegiku [24] give a constant factor approximation algorithm for the problem of minimizing the average shortest-path distance between all pairs of nodes by adding shortcut edges. Several algorithms are proposed in [27, 28] and experimentally shown to perform well in practice. Bauer et al. [3] study the problem of minimizing the average number of hops in shortest paths. They prove that the problem cannot be approximated within a logarithmic factor and provide respective approximation algorithms. Bilò et al. [5] and Demaine and Zadimoghaddam [15] consider the problem of minimizing the diameter of a graph and provide constant factor approximation algorithms.

The problem of maximizing revenue by pricing the edges of a graph has been studied in several works. These problems are known under different names such as the *network (or highway) pricing problem* [21, 9], but also as *Stackelberg network pricing games* [31, 8].

Labbe et al. [21] use a bilevel optimization model for taxing a given subset of the edges in a network to maximize the revenue that the leader receives from the followers. Among other results, they prove that the problem is NP-hard for single-commodity instances, exploiting negative edge costs and lower bound restrictions on the taxes. In a subsequent work, Roch et al. [31] improve upon this result and show NP-hardness for non-negative edge costs and without lower bound restrictions. They also provide an approximation algorithm for the single-commodity case.

Briest et al. [8] consider the following Stackelberg setting: There are several buyers who are interested in buying certain (pre-determined) subgraphs of the network and a seller (network owner) who can price a given subset of the edges. Once the seller fixes the prices, the buyers purchase the cheapest subgraph they are interested in. The goal is to maximize the total revenue obtained from the buyers. The authors show that a uniform price for all edges guarantees the seller a revenue within logarithmic factor of the optimal revenue. A more specific problem was considered by Briest et al. [7], where each buyer i is interested in purchasing a subgraph that contains a shortest s_i, t_i -path. Other special cases were considered in [18, 17, 19].

In general, there is a vast literature on the problem of pricing multiple items so as to maximize the revenue obtained from (possibly budget-constrained) buyers. There is a close connection between our problem and the problem of determining *envy-free* prices [19], because envy-freeness naturally corresponds to choosing the cheapest available option. Especially, we exploit known hardness results for the special cases of the *unit-demand pricing problem* and the *single-minded pricing problem* (see [19, 6, 10]) to establish the inapproximability results of our (more restrictive) network pricing problem.

We emphasize that our problem differs from the ones mentioned above because (i) the seller corresponds to a given node u who can set the prices of its outgoing edges only, and (ii) the revenue that u obtains depends on the proportion of the demand of each commodity routed along shortest paths through u .

2 Preliminaries

We formally define the *Network Pricing Problems* considered in this paper: Suppose we are given a directed graph $G = (V, E)$ with non-negative edge costs $(c_e)_{e \in E}$, k commodities $(s_i, t_i, w_i)_{i \in [k]}$ ¹, and a designated node $u \in V$. Each commodity $i \in [k]$ is specified by a source-target pair $(s_i, t_i) \in V \times V$ with $s_i \neq t_i$ and a non-negative demand (or weight) $w_i > 0$. The interpretation here is that each commodity $i \in [k]$ sends a total of w_i units of flow from the source node s_i to the target node t_i . The demand w_i is split evenly along all (simple²) shortest s_i, t_i -paths with respect to the edge costs $(c_e)_{e \in E}$ (formal definitions are given below). We assume that for each commodity $i \in [k]$, $s_i, t_i \neq u$ and there is at least one s_i, t_i -path that passes through u . This assumption is without loss of generality as otherwise the commodity is irrelevant (as will become clear below) and can be removed.

¹ Given an integer $k \geq 1$, we define $[k] = \{1, \dots, k\}$.

² Recall that a path is said to be *simple* if it does not contain any cycles. Throughout the paper, whenever we refer to a shortest path we implicitly mean a simple shortest path.

We introduce some more notation. Let n and m be the the number of nodes and edges of G , respectively. We use the standard notation $\delta^+(u)$ to refer to the set of all outgoing edges of u , i.e., $\delta^+(u) = \{(u, v) \in E\}$, and define $\Delta(u) = |\delta^+(u)|$ as the outdegree of u . Given a pair of nodes $(x, y) \in V \times V$ with $x \neq y$, we denote by $\pi(x, y)$ the number of shortest x, y -paths with respect to $(c_e)_{e \in E}$. Similarly, we use $d(x, y)$ to refer to the total cost of a shortest x, y -path; we also say that $d(x, y)$ is the *distance* between x and y . For a set $A \subseteq E$, we denote by $d_{E \setminus A}(x, y)$ the distance between x and y in the graph $G = (V, E \setminus A)$, where the edges in A are removed. Below, we often omit the explicit reference to the respective edge costs if they are clear from the context.

For ease of notation, for every commodity $i \in [k]$, we use $\pi^i = \pi(s_i, t_i)$ to refer to the number of shortest s_i, t_i -paths. Further, we define π_u^i as the number of shortest s_i, t_i -paths that pass through node $u \in V$, where $s_i, t_i \neq u$. Given an outgoing edge $e = (u, v) \in E$ of u , we denote by π_e^i the number of shortest s_i, t_i -paths that pass through e . Observe that $\pi_u^i = \sum_{e \in \delta^+(u)} \pi_e^i$.

We can now define the flow that passes through the outgoing edges of u : Recall that the demand w_i of each commodity $i \in [k]$ is assumed to be split evenly over all shortest s_i, t_i -paths. Formally, the flow f_e^i of an outgoing edge $e = (u, v)$ of commodity i is defined as $f_e^i = w_i \cdot \pi_e^i / \pi^i$. The total flow passing through node u with respect to commodity i is then

$$f_u^i = \sum_{e \in \delta^+(u)} f_e^i = \sum_{e \in \delta^+(u)} w_i \cdot \frac{\pi_e^i}{\pi^i} = w_i \cdot \frac{\pi_u^i}{\pi^i}.$$

Further, we define $f_e = \sum_{i \in [k]} f_e^i$ as the total flow on edge e . The *total flow of node u* is then defined as

$$f_u = \sum_{e \in \delta^+(u)} f_e = \sum_{i \in [k]} \sum_{e \in \delta^+(u)} f_e^i = \sum_{i \in [k]} w_i \cdot \frac{\pi_u^i}{\pi^i} = \sum_{i \in [k]} f_u^i.$$

Another notion that is of interest in this paper is the following one: The *total revenue of node u* is defined as

$$r_u = \sum_{e \in \delta^+(u)} f_e \cdot c_e = \sum_{i \in [k]} \sum_{e \in \delta^+(u)} f_e^i \cdot c_e = \sum_{i \in [k]} \sum_{e \in \delta^+(u)} w_i \cdot \frac{\pi_e^i}{\pi^i} \cdot c_e.$$

Suppose we can change the costs of $\kappa \in [\Delta(u)]$ outgoing edges of u . How would we set the edge costs such that the total flow (or revenue, respectively) of u is maximized? More precisely, our goal is to determine a set $S \subseteq \delta^+(u)$ with $|S| \leq \kappa$ and non-negative costs $\bar{c}_S = (\bar{c}_e)_{e \in S}$ for the edges in S such that f_u (or r_u , respectively) with respect to the combined edge costs (\bar{c}_S, c_{-S}) is maximized, where we use $c_{-S} = (c_e)_{e \in E \setminus S}$ to refer to the (original) costs of the edges in $E \setminus S$ that remain unchanged. For convenience, we write $\bar{c}_e = \bar{c}_{\{e\}}$, we also write p_S when we set the cost of all edges in S to $p \in \mathbb{R} \cup \{\infty\}$. We use $f_u(\bar{c}_S)$ and $r_u(\bar{c}_S)$ to refer to the total flow and revenue of u , respectively, with respect to (\bar{c}_S, c_{-S}) .

This gives rise to the following two optimization problems:

NETWORK PRICING PROBLEM (NPP)

Given: A directed graph $G = (V, E)$ with non-negative edge costs $(c_e)_{e \in E}$, k commodities $(s_i, t_i, w_i)_{i \in [k]}$, a designated node $u \in V$ and a number $\kappa \in [\Delta(u)]$.

Goal: Determine a set $S \subseteq \delta^+(u)$ with $|S| \leq \kappa$ and edge costs $\bar{c}_S = (\bar{c}_e)_{e \in S}$ such that $f_u(\bar{c}_S)$ is maximized (**FLOW-NPP**), or $r_u(\bar{c}_S)$ is maximized (**REV-NPP**)

13:6 Approximate Pricing in Networks

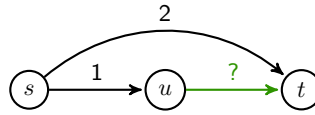
Note that if the commodities correspond to all possible node pairs of the graph (not involving u as a source or target node), then the flow through u is precisely the betweenness centrality of u (as introduced above). In particular, in this case **FLOW-NPP** can be interpreted as the problem of maximizing the betweenness centrality of u .

In our discussion below, we distinguish the following three cases:

- (C1) $\kappa = 1$: We are allowed to change the cost of only one outgoing edge of u .
- (C2) $1 < \kappa < \Delta(u)$: We are allowed to change the cost of κ outgoing edges of u .
- (C3) $\kappa = \Delta(u)$: We are allowed to change the cost of all the outgoing edges of u .

We continue with some basic observations. A pathological case we want to avoid in **REV-NPP** is that we can charge arbitrarily high costs.

► **Assumption 2.1.** For every commodity $i \in [k]$ there is at least one s_i, t_i -path that does not pass through u .



■ **Figure 1** Example graph.

Throughout the paper, we assume that the edge costs are non-negative integers (as they may correspond to monetary units, percentages of a fixed precision, etc.).³ The following example shows that this assumption is needed if one wants to be able to determine edge costs that realize the optimal revenue. Consider the instance depicted in Figure 1 and assume that there is a unit demand to be sent from s to t . Suppose we can impose an arbitrary non-negative rational cost $c_e \in \mathbb{Q}_{\geq 0}$ on the edge $e = (u, t)$. If we set $c_e = 1$, then the revenue of u becomes $\frac{1}{2}$. Otherwise, if we set $c_e = 1 - \varepsilon$ for a small rational $\varepsilon > 0$, then the revenue of u is $1 - \varepsilon$. It follows that **REV-NPP** does not admit an optimal solution.

Finally, we need to be able to efficiently compute how the flow splits. If there are zero cost cycles this may become infeasible [34]. We thus make the following assumption:

► **Assumption 2.2.** The edge costs $(c_e)_{e \in E}$ are non-negative integers and the graph does not contain any zero cost cycles, even if all outgoing edges of u are set to zero.

Using Assumption 2.2, it is possible to compute all relevant flows (as defined above) in polynomial time.⁴ Throughout the paper we use this fact without stating it explicitly.

Due to space restrictions, some proofs are omitted from this extended abstract and can be found in the full version of the paper.

3 Flow Maximization Problem

In this section, we consider the problem **FLOW-NPP**. We first prove the following intuitive monotonicity property for the flow f_u through u : If the cost of a single outgoing edge of u decreases, then the flow through u does not decrease.

³ All our results continue to hold if the edge costs are of the form $p \cdot \mathbb{Z}_{\geq 0}$ for some real number $p > 0$. In particular, this covers most practically relevant scenarios where one is bound to a finite number of decimals.

⁴ This can be done by running for every commodity $i \in [k]$ an adapted version of Dijkstra's shortest path algorithm [16] which also counts the number of shortest paths passing through the edges.

► **Lemma 1.** Consider an edge $e = (u, v) \in \delta^+(u)$ and assume that the edge cost c_e is decreased to $\bar{c}_e < c_e$. Then $f_u(\bar{c}_e) \geq f_u(c_e)$.

Using Lemma 1, it is clear what we should do if we can price a subset $S \subseteq \delta^+(u)$ of edges: Simply set the cost of each edge $e \in S$ to zero to maximize the flow through u .

► **Corollary 2.** Suppose we can change the costs of the edges in $S \subseteq \delta^+(u)$. Then setting $\bar{c}_e = 0$ for every $e \in S$ maximizes the flow f_u of u .

Note that this takes away the difficulty of determining optimal costs for the edges in S . What remains is how to find the right subset of edges S to be priced. This is easy in cases **(C1)** and **(C3)**: It is not hard to see that by using complete enumeration over all possible subsets and Corollary 2, **FLOW-NPP** can be solved efficiently if $\kappa = \mathcal{O}(1)$ or $\kappa = \Delta(u) - \mathcal{O}(1)$.

► **Theorem 3.** **FLOW-NPP** can be solved optimally in polynomial time for $\kappa = \mathcal{O}(1)$ and $\kappa = \Delta(u) - \mathcal{O}(1)$.

We consider the cases of **(C2)** which are not captured by Theorem 3. Then the approach above fails. In fact, we show that **FLOW-NPP** is NP-hard to approximate within a factor $1 - 1/e$, even in the unit demand setting (i.e., $w_i = 1$ for all $i \in [k]$).

► **Theorem 4.** Assuming $P \neq NP$, there is no α -approximation algorithm with $\alpha > 1 - 1/e$ for **FLOW-NPP** with $\mathcal{O}(1) < \kappa < \Delta(u) - \mathcal{O}(1)$, even in the unit demand setting.

We derive a $(1 - 1/e)$ -approximation algorithm for **FLOW-NPP**, which is best possible by Theorem 4. We use a well-known result due to Nemhauser et al. [25] for the following submodular function maximization problem: Given a finite set N , a function $z : 2^N \rightarrow \mathbb{R}$ and an integer k' , find a set $S \subseteq N$ such that $|S| \leq k'$ and $z(S)$ is maximum. If z is non-negative, monotone and submodular⁵, then the following natural greedy algorithm exhibits an approximation ratio of $1 - 1/e$ [25]: Start with the empty set and repeatedly add an element that gives the maximal *marginal gain*, i.e., if S is a partial solution, choose the element $j \in N \setminus S$ that maximizes $z(S \cup \{j\}) - z(S)$.

We show that $f_u(\bar{c}_S)$ (if considered as a set function) is non-negative, monotone and submodular.

► **Lemma 5.** Define $z(S) = f_u(0_S)$ for every $S \subseteq \delta^+(u)$. The function z is non-negative, monotone and submodular.

■ **Algorithm 1** Greedy algorithm for **FLOW-NPP**.

```

1  $S = \emptyset$ 
2 for  $i = 1, \dots, \kappa$  do
3    $e_{\max} = \arg \max \{f_u(0_{S \cup \{e\}}) : e \in \delta^+(u) \setminus S\}$ 
4    $S = S \cup \{e_{\max}\}$ 
5 return  $S$ 

```

Applied to our setting, the greedy algorithm proceeds as described in Algorithm 1.

► **Theorem 6.** The greedy algorithm provides a $(1 - 1/e)$ -approximation for **FLOW-NPP**.

⁵ Let N be a finite set and let $z : 2^N \rightarrow \mathbb{R}$ be a function. Then z is (i) *non-negative* if $z(S) \geq 0$ for every $S \subseteq N$, (ii) *monotone* if $z(S) \leq z(T)$ for every $S \subseteq T \subseteq N$, and (iii) *submodular* if for all sets $S \subseteq T \subseteq N$ and every element $e \in N \setminus T$, $z(S \cup \{e\}) - z(S) \geq z(T \cup \{e\}) - z(T)$.

4 Revenue Maximization Problem

We turn to the problem **REV-NPP**. As it turns out, this problem is much more challenging than **FLOW-NPP**. In fact, even if we can change the costs of all outgoing edges of u it remains non-trivial to find good approximation algorithms (see Section 4.3).

4.1 Changing the cost of one edge

We consider case **(C1)** of **REV-NPP**, i.e., we can change the cost of one outgoing edge. We first show that we can efficiently compute the optimal cost if the edge is *given*.

► **Lemma 7.** *Fix an outgoing edge $e = (u, v)$ of u . We can then determine the cost \bar{c}_e of e maximizing the revenue $r_u(\bar{c}_e)$ of u in polynomial time.*

Proof. Let \bar{c}_e^* be some optimal cost which maximizes $r_u(\bar{c}_e^*)$. We first claim that there exists some optimal cost \bar{c}_e with $\bar{c}_e \in T$, where

$$T = \left(\bigcup_{i \in [k]} \{T_i - 1, T_i\} \right) \cup \{\infty\} \quad \text{and} \quad T_i := d_{E \setminus \{e\}}(s_i, t_i) - d(s_i, u) - d(v, t_i) \quad \forall i \in [k].$$

If $\bar{c}_e^* > \max\{T_i : i \in [k]\}$ there is no flow passing through e . We obtain the same by setting $\bar{c}_e = \infty \in T$ and thus $r_u(\bar{c}_e) = r_u(\bar{c}_e^*)$, which is optimal. Suppose now that $\bar{c}_e^* \leq \max\{T_i : i \in [k]\}$ and $\bar{c}_e^* \notin T$. Let $L = \{i \in [k] : T_i < \bar{c}_e^*\}$ and $U = \{i \in [k] : T_i - 1 > \bar{c}_e^*\}$. For the commodities in L there is no flow passing through e , while for the commodities in U the entire flow passes through e . By setting $\bar{c}_e = \min\{T_i - 1 : i \in U\}$ the flows do not change while $\bar{c}_e > \bar{c}_e^*$. Because $U \neq \emptyset$ we have $r_u(\bar{c}_e) > r_u(\bar{c}_e^*)$, contradicting the optimality of \bar{c}_e^* . Hence there is an optimal cost \bar{c}_e in T .

Determining T takes at most $3k$ shortest path calculations. If all costs are fixed, we can compute the revenue by k shortest path calculations. Exploiting that $|T| \leq 2k + 1$, we can thus simply try all values in T and choose \bar{c}_e as the cost that gives the largest revenue. ◀

By iterating over all edges $e = (u, v)$ of u and using Lemma 7 to determine the maximum revenue $r_u(\bar{c}_e)$, we can determine the optimal cost among all these edges. We obtain:

► **Theorem 8. REV-NPP(C1)** *can be solved optimally in polynomial time.*

4.2 Changing the costs of κ edges

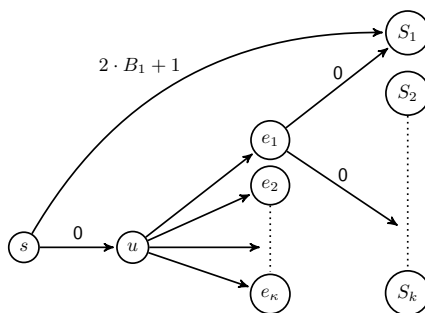
We turn to case **(C2)** of **REV-NPP**. As we show, this problem is hard to approximate:

► **Theorem 9.** *Assuming $P \neq NP$, there is no α -approximation algorithm with $\alpha > 1 - 1/e$ for **REV-NPP(C2)** with $1 < \kappa < \Delta(u)$, even in the unit demand setting.*

One could hope that a greedy approach similar to the one used for **FLOW-NPP(C2)** would work here as well. Unfortunately, this is not the case. In fact, the greedy algorithm can perform arbitrarily bad. Further, the objective function is not submodular (even if the original costs are assumed to be $c_{\delta+(u)} = \infty_{\delta+(u)}$).

4.3 Changing the costs of all edges

We come to case **(C3)** of **REV-NPP**, where we are allowed to change the costs of all outgoing edges of u , i.e., $\kappa = \Delta(u)$. We start by proving some inapproximability results, both for the general and the unit demand setting, and then turn to our approximation algorithms.



■ **Figure 2** Illustration of the instance used in the proof of Theorem 10.

Inapproximability. Under reasonable hardness assumptions, case **REV-NPP(C3)** is hard to approximate within a factor of $\Omega(1/\log^\varepsilon(k))$ when considering unit demands and of $\Omega(1/\Delta(u)^\varepsilon)$ when considering arbitrary demands.

► **Theorem 10.** **REV-NPP(C3)** is $\Omega(1/\log^\varepsilon(k))$ -inapproximable for some $\varepsilon > 0$ in the unit demand setting, assuming that no polynomial-time algorithm can approximate constant-degree Balanced Bipartite Independent Set⁶ to within arbitrarily small constant factors. **REV-NPP(C3)** is $\Omega(1/\Delta(u)^\varepsilon)$ -inapproximable for some $\varepsilon > 0$ for arbitrary demands, assuming $NP \not\subseteq \cap_{\delta > 0} BPTIME(2^{O(n^\delta)})$.

We use a reduction from the *Unit-Demand Min-Buying Pricing Problem* (**UDP_{min}**) [6]: We are given a set of N items $\mathcal{I} = \{e_1, \dots, e_N\}$ and a set of k consumers $\mathcal{C} = \{c_1, \dots, c_k\}$. Every consumer $c_i \in \mathcal{C}$ has some budget $B_i \in \mathbb{Z}_{\geq 0}$ and a set $S_i \subseteq \mathcal{I}$ of items she is interested in. Given prices $p : \mathcal{I} \rightarrow \mathbb{Z}_{\geq 0}$ for the items, consumer c_i will buy an item $e \in S_i$ with $p(e)$ minimum, but only if $p(e) \leq B_i$. The goal is to find prices that maximize the total revenue, i.e.,

$$\sum_{c_i \in \mathcal{C}} \min\{p(e) : e \in S_i \wedge p(e) \leq B_i\},$$

where we define the minimum of an empty set to be zero. In the so-called *economist's version* of **UDP_{min}** (**EUDP_{min}**) we are additionally given a (discrete) probability distribution $\mathbb{P} : \mathcal{C} \rightarrow [0, 1]$ over the consumers which is then incorporated in the objective function by multiplying the revenue gained from a consumer with her probability. Note that we can think of this probability distribution as having weights on the consumers.

Proof of Theorem 10. We give a reduction from **EUDP_{min}** to **REV-NPP** with arbitrary demands. The same reduction also provides the hardness result for uniform demands because we can see **UDP_{min}** as a special case of **EUDP_{min}**, where all consumers have equal probabilities, and in what follows **UDP_{min}** is then reduced to **REV-NPP** with uniform demands.

We reduce an instance I of **EUDP_{min}** to an instance I' of **REV-NPP** such that any solution of I' can be converted into a solution of I losing at most a factor 2 in objective value. As a consequence, an α -approximation algorithm for **REV-NPP** with $\alpha = \Omega(1/\Delta(u)^\varepsilon)$ (respectively,

⁶ In this problem, we are given a bipartite graph $G = (V, W, E)$ and we want to find maximum cardinality subsets of vertices $V' \subseteq V, W' \subseteq W$ with $|V'| = |W'|$, such that $\{v, w\} \notin E$ for all $v \in V', w \in W'$; see [6] for more details.

13:10 Approximate Pricing in Networks

$\alpha = \Omega(1/\log^\varepsilon(k))$ provides an $\alpha/2$ -approximation algorithm for **EUDP**_{min} (respectively, **UDP**_{min}). Briest [6] showed that the latter is not possible (under the assumptions stated in Theorem 10).⁷

Let $(\mathcal{I}, \mathcal{C}, (S_c)_{c \in \mathcal{C}}, (B_c)_{c \in \mathcal{C}}, \mathbb{P})$ be an instance of **EUDP**_{min}. We construct an instance $I' = (G, (c_e)_{e \in E}, (s_i, t_i, w_i)_{i \in [k]}, u, \kappa)$ of **REV-NPP** as follows: Let the set of vertices of G be $V = \{s, u, S_1, \dots, S_k, e_1, \dots, e_N\}$, where each S_i , $i \in [k]$, and $e_j \in \mathcal{I}$ correspond to their counterpart in I . The set of edges E and the respective edge costs $(c_e)_{e \in E}$ are defined as follows (see Figure 2 for an illustration): There is an edge (s, u) of cost 0. For every S_j , $j \in [k]$, there is an edge which needs to be priced. For every $e_i \in \mathcal{I}$ and S_j , $j \in [k]$, such that $e_i \in S_j$ there is an edge (e_i, S_j) of cost 0. For every S_j , $j \in [k]$, there is an edge (s, S_j) of cost $2 \cdot B_j + 1$. Finally, we have k commodities (s, S_j, w_j) with demand $w_j = \mathbb{P}(c_j)$ for every $j \in [k]$. Note that $\Delta(u) = N$. Clearly, this reduction can be done in polynomial time.

First note that $\text{OPT}(I') \geq 2\text{OPT}(I)$ since taking the optimal prices p in I and using the prices $\bar{c}_{(u, e_i)} = 2p(e_i)$ for all $i \in [N]$ in I' will give a revenue of $2\text{OPT}(I)$ for I' .

Consider a solution \bar{c} for I' with some revenue Z' . We will convert this into a solution Z for I with value at least $Z'/4$. Note that we may assume that $B_j \geq 1$ for all $j \in [k]$ and therefore that $Z' \geq \sum_{i \in [k]} w_i \cdot 2 \min_{j \in [k]} \{B_j\} \geq 2 \sum_{i \in [k]} w_i$ which is the revenue we would get by setting all prices to $2 \min_{j \in [k]} \{B_j\}$. Now, modify \bar{c} by subtracting 1 from \bar{c}_e for all $e \in E$ if \bar{c}_e is odd. This will cost us at most $\sum_{i \in [k]} w_i$ revenue. Thus we have $Z' - \sum_{i \in [k]} w_i \geq Z'/2$ revenue remaining. Observe that all prices are even and that $f_u^i/w_i \in \{0, 1\}$ for all $i \in [k]$. Using prices $p(e_i) = \bar{c}_{(u, e_i)}/2$ for $i \in [k]$ in I yields a revenue of at least $Z'/4$.

To conclude, if $Z' \geq \alpha \text{OPT}(I')$ then $4Z \geq Z' \geq \alpha \text{OPT}(I') \geq 2\alpha \text{OPT}(I)$ implying $Z \geq \alpha/2 \text{OPT}(I)$ which proves the theorem. \blacktriangleleft

Special case: single commodity. We next consider the problem of **REV-NPP(C3)** for a single commodity only, i.e., $k = 1$. In this case, we can assume without loss of generality that $w_1 = 1$. Our goal is thus to determine $\bar{c}_{\delta^+(u)} = (\bar{c}_e)_{e \in \delta^+(u)}$ to maximize the revenue

$$r_u(\bar{c}_{\delta^+(u)}) = \sum_{e \in \delta^+(u)} f_e^1 \cdot \bar{c}_e = w_1 \sum_{e \in \delta^+(u)} \frac{\pi_e^1}{\pi_1} \cdot \bar{c}_e = \sum_{e \in \delta^+(u)} \frac{\pi_e^1}{\pi_1} \cdot \bar{c}_e.$$

► Theorem 11. REV-NPP(C3) *with a single commodity only (i.e., $k = 1$) can be solved optimally in polynomial time.*

Proof. For every edge $(u, v) \in \delta^+(u)$, we compute the value $h(v) := d_{E \setminus \delta^+(u)}(s_1, t_1) - d(s_1, u) - d(v, t_1)$. Let $T = \max_{(u, v) \in \delta^+(u)} h(v)$. If $T \leq 0$, then no revenue can be obtained and we stop. If $T > 0$, we compute the revenue obtained by setting all costs uniformly to either $T - 1$ or T :

$$\begin{aligned} r_u((T - 1)_{\delta^+(u)}) &= T - 1 \\ r_u(T_{\delta^+(u)}) &= T \cdot \left(\sum_{e=(u, v) \in \delta^+(u): h(v)=T} \pi_e^1 \right) / \pi_1. \end{aligned}$$

⁷ In fact, Briest [6] established the corresponding inapproximability results, where the prices and budgets are assumed to be reals. However, it is not hard to see that multiplying all budgets in the proof of Theorem 2 in [6] by a factor 2^k results in integer budgets. Then we can still assume that the prices are powers of 2, but now these powers are positive making also the prices integer. That is, the results in [6] also go through for integer values and budgets. We use this for our reduction here. Based on different assumptions, Chalermsook et al. [10] provide a stronger inapproximability result for the unit demand setting. If the same trick can be applied to the reduction presented in [10], our proof shows that the unit demand setting is $\log^{1-\epsilon}(k)$ -inapproximable for every $\epsilon > 0$.

Algorithm 2 Uniform Price Algorithm.

```

1  $C = \emptyset$ 
2 foreach commodity  $i \in [k]$  do
3   Compute prices  $T_i - 1$  and  $T_i$  as in Theorem 11 (considering only commodity  $i$ )
4   Let  $T_i^* \in \{T_i - 1, T_i\}$  be the price achieving higher revenue
5    $C = C \cup \{T_i^*\}$ 
6 return  $\arg \max\{r_u(p_{\delta^+(u)}) : p \in C\}$ 

```

We argue that the maximum of the two is the optimal revenue. When $\bar{c}_e > T$ for some $e \in \delta^+(u)$ it holds that $f_e = 0$, so we can assume that there is an optimum where $\bar{c}_e \leq T$ for all $e \in \delta^+(u)$. If there is an optimum for which $\bar{c}_e \leq T - 1$ for all $e \in \delta^+(u)$ then the maximum revenue we can get is $T - 1$ which is actually attained by setting all \bar{c}_e to $T - 1$. Suppose the optimum is larger than $T - 1$ then there must be some $e' \in \delta^+(u)$ with $\bar{c}_{e'} = T$ and strictly positive flow. If there is some other edge for which $\bar{c}_e < T$ which gets flow then this contradicts $\bar{c}_{e'}$ getting flow, thus it cannot have flow in which case we could also set it to T . Thus then there must also be an optimum where $\bar{c}_e = T$ for all $e \in \delta^+(u)$. So, the maximum of $r_u(T_{\delta^+(u)})$ and $r_u((T - 1)_{\delta^+(u)})$ is indeed the optimum. The values of $h(v)$ and $r_u(T_{\delta^+(u)})$ and $r_u((T - 1)_{\delta^+(u)})$ can all be computed in polynomial time. ◀

Uniform pricing. We exploit the fact that for a single commodity we are able to find optimal uniform costs in polynomial time. Consider the *uniform price algorithm* described in Algorithm 2. First, we consider the case where all demands are uniform.

► **Theorem 12.** *Algorithm 2 is a $1/H_k$ -approximation algorithm for REV-NPP(C3) when all demands are uniform and this is tight.*

Proof. We can assume without loss of generality that all demands are 1. Let T_i^* be the optimal price for commodity $i \in [k]$ as determined in the proof of Theorem 11 and let f_u^{i*} be the flow of commodity i going through u when using prices $\bar{c}_{\delta^+(u)} = (T_i^*)_{\delta^+(u)}$.

Assume that the commodities are ordered such that $T_1^* \geq T_2^* \geq \dots \geq T_k^*$ and if $i < j$ and $T_i^* = T_j^*$ then $f_u^{i*} \geq f_u^{j*}$. So, first we order on T_i^* and if the T_i^* are equal then we order on f_u^{i*} . Let s be the number of unique values among the T_i^* . Let $i_1 = 1$ and define i_j for $2 \leq j \leq s$ recursively as the first entry that is strictly smaller than $T_{i_{j-1}}^*$. For convenience let $i_{s+1} = k + 1$.

Let P be the output of Algorithm 2. The algorithm tries prices T_i^* and because we have unit demands and by the ordering of the commodities we know that for $i \in \{i_j, \dots, i_{j+1} - 1\}$, it holds that

$$P \geq T_i^* \cdot \left((i_j - 1) + \sum_{\ell=i_j}^{i_{j+1}-1} f_u^{\ell*} \right), \quad \text{which implies} \quad T_i^* \leq \frac{P}{(i_j - 1) + \sum_{\ell=i_j}^{i_{j+1}-1} f_u^{\ell*}}. \quad (1)$$

Let OPT be the maximum attainable revenue. If we single out the income from one commodity we cannot expect to do better than when we just consider that commodity. Hence,

13:12 Approximate Pricing in Networks

$$\begin{aligned}
 \text{OPT} &\leq \sum_{i=1}^k T_i^* f_u^{i*} = \sum_{j=1}^s \sum_{i=i_j}^{i_{j+1}-1} T_i^* f_u^{i*} \leq \sum_{j=1}^s \sum_{i=i_j}^{i_{j+1}-1} \frac{P \cdot f_u^{i*}}{(i_j - 1) + \sum_{\ell=i_j}^{i_{j+1}-1} f_u^{\ell*}} \\
 &\leq \sum_{j=1}^s \sum_{i=i_j}^{i_{j+1}-1} \frac{P}{(i_j - 1) + (i - (i_j - 1))} = \sum_{j=1}^s \sum_{i=i_j}^{i_{j+1}-1} \frac{P}{i} = \sum_{i=1}^k \frac{P}{i} = H_k \cdot P
 \end{aligned} \tag{2}$$

The second inequality follows from (1). For the third inequality we make use of the fact that $f_u^{i*} \leq 1$ and that we sorted the commodities in such a way that if $i < j$ and $T_i^* = T_j^*$ we have $f_u^{i*} \geq f_u^{j*}$. Thus there are at least $i - (i_j - 1)$ terms for which the T^* -value is equal but the f -value is at least as large. ◀

We turn to the general demand case. We modify Algorithm 2 by replacing line 5 with $C = C \cup \{T_i - 1, T_i\}$.

► **Theorem 13.** *The modified version of Algorithm 2 is a $\max\{1/k, 1/\Delta(u)\}$ -approximation algorithm for **REV-NPP(C3)** and this is tight.*

Proof. To show that it is a $1/k$ -approximation algorithm we only need to consider the prices used in the original version of Algorithm 2. We follow the same reasoning as in Theorem 12. Let T_i^* and f_u^{i*} , $i \in [k]$, be as in the proof of Theorem 12. We note that $P \geq T_i^* \sum_{\ell=1}^i f_u^{\ell*}$, which implies that $T_i^* \leq P / \sum_{\ell=1}^i f_u^{\ell*}$. Thus,

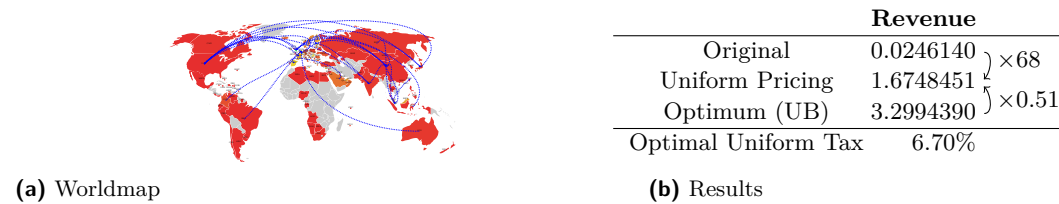
$$\text{OPT} \leq \sum_{i=1}^k T_i^* \cdot f_u^{i*} \leq \sum_{i=1}^k \frac{P \cdot f_u^{i*}}{\sum_{\ell=1}^i f_u^{\ell*}} \leq \sum_{i=1}^k P = k \cdot P. \tag{3}$$

Next we show that the modified version of Algorithm 2 is also a $1/\Delta(u)$ -approximation algorithm. Let \bar{c}^* be the optimal prices giving a revenue of OPT. If we consider the revenue that is contributed by each $e \in \delta^+(u)$, there is at least one $e^* \in \delta^+(u)$ which contributed at least $\text{OPT}/\Delta(u)$, i.e., $f_{e^*} \cdot \bar{c}_{e^*}^* \geq \text{OPT}/\Delta(u)$. Consider $\bar{c}_e = \bar{c}_{e^*}^*$ for all $e \in \delta^+(u)$. The flow f_{e^*} will not go down because of $e \in \delta^+(u)$ such that $\bar{c}_e^* < \bar{c}_{e^*}^*$. Some of f_{e^*} may go to $e \in \delta^+(u)$ such that $\bar{c}_e^* \geq \bar{c}_{e^*}^*$ but if this happens we will still earn at least $\bar{c}_{e^*}^*$ on it. Hence the revenue for \bar{c}_e is at least $f_{e^*} \cdot \bar{c}_{e^*}^* \geq \text{OPT}/\Delta(u)$.

Fix $\bar{c}_e = \bar{c}_{e^*}^*$ for all $e \in \delta^+(u)$. Let $\mathcal{F} = \{i \in [k] : f_u^i > 0\}$, i.e., all commodities that have some positive flow going through u and so we earn some revenue on them. Let T_i be the T corresponding to commodity i as in Theorem 11. Note that $\bar{c}_e^* \leq \min\{T_i : i \in \mathcal{F}\}$. If $\bar{c}_e^* \geq \min\{T_i - 1 : i \in \mathcal{F}\}$ then we are done because then the approximation algorithm will try a price which yields at least $\text{OPT}/\Delta(u)$ revenue. Suppose $\bar{c}_e^* < \min\{T_i - 1 : i \in \mathcal{F}\}$. Then $f_u^i/w_i = 1$ for all $i \in \mathcal{F}$ and when raising \bar{c}_e to $\min\{T_i - 1 : i \in \mathcal{F}\}$ for all $e \in \Delta(u)$ the flows for commodities $i \in \mathcal{F}$ will not change while the revenue increases. Hence the approximation algorithm tries a price which yields a revenue of at least $\text{OPT}/\Delta(u)$. We conclude that Algorithm 2 is a $1/\Delta(u)$ -approximation algorithm. ◀

5 Conclusion

A motivating scenario for this research was figuring out how a country should change its tax rates in order to maximize its revenue. Computing the optimum is an intractable problem, but we can use our results to compute an optimal uniform tax. We used tax data from [35, 36], which also provides estimates of the volumes that are sent from one country to another (based



■ **Figure 3** Outcome of experiments.

on the sizes of their economies). The data contains 108 countries (nodes), 8777 tax treaties (edges) and 11342 commodities. In this scenario, we need to find “money-transfer” paths such that the total tax paid by the companies is as low as possible. We run our experiments with “The Netherlands” as node u . The results are summarized in Figure 3. If the Netherlands would change its outgoing tax rate to 6.7% for all treaties, it would potentially increase its revenue by a factor 68. Further, the optimal uniform tax revenue is even within 51% of the optimum (upper bound as in (3)) and thus much better as suggested by Theorem 13.

We settle most cases of **FLOW-NPP** and **REV-NPP** in this paper but a case which is not completely settled is **REV-NPP(C2)**. Although we show that it is inapproximable within a factor $1 - 1/e$, case **(C3)** seems to suggest that it may even be harder.

An interesting way to look at our problem is from a game theory perspective. Now that we know what one node will do (approximately), what will happen if the nodes correspond to strategic players? Will they settle in a stable scenario where everybody gets some revenue, or will it end in a “price war” where the revenue of each player becomes zero?

References

- 1 Nederland is een aantrekkelijk belastingland. NOS, Nov. 6, 2014.
- 2 Konstantin Avrachenkov and Nelly Litvak. The Effect of New Links on Google Pagerank. *Stoc. Models*, 2006.
- 3 Reinhard Bauer, Gianlorenzo D’Angelo, Daniel Delling, Andrea Schumm, and Dorothea Wagner. The Shortcut Problem - Complexity and Algorithms. *J. Graph Algorithms Appl.*, 16(2):447–481, 2012.
- 4 Elisabetta Bergamini, Pierluigi Crescenzi, Gianlorenzo D’Angelo, Henning Meyerhenke, Lorenzo Severini, and Yllka Velaj. Improving the Betweenness Centrality of a Node by Adding Links. *ACM J. of Experimental Algorithmics*, 23, 2018.
- 5 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.
- 6 Patrick Briest. Uniform Budgets and the Envy-Free Pricing Problem. In *ICALP*, pages 808–819, 2008.
- 7 Patrick Briest, Parinya Chalermsook, Sanjeev Khanna, Bundit Laekhanukit, and Danupon Nanongkai. Improved Hardness of Approximation for Stackelberg Shortest-Path Pricing. In *Internet and Network Economics*, 2010.
- 8 Patrick Briest, Martin Hoefer, and Piotr Krysta. Stackelberg Network Pricing Games. *Algorithmica*, 62(3), 2012.
- 9 Luce Brotcorne, F. Cirinei, Patrice Marcotte, and Gilles Savard. An exact algorithm for the network pricing problem. *Discrete Optimization*, 8(2):246–258, 2011.
- 10 Parinya Chalermsook, Julia Chuzhoy, Sampath Kannan, and Sanjeev Khanna. Improved Hardness Results for Profit Maximization Pricing Problems with Unlimited Supply. In *APPROX-RANDOM*, pages 73–84, 2012.

13:14 Approximate Pricing in Networks


- 11 Pierluigi Crescenzi, Gianlorenzo D'Angelo, Lorenzo Severini, and Yllka Velaj. Greedily improving our own centrality in a network. In *SEA 2015*, volume 9125 of *Lecture Notes in Computer Science*, pages 43–55, 2015.
- 12 Pierluigi Crescenzi, Gianlorenzo D'Angelo, Lorenzo Severini, and Yllka Velaj. Greedily Improving Our Own Closeness Centrality in a Network. *ACM Trans. Knowl. Discov. Data*, 11(1):9:1–9:32, 2016.
- 13 Gianlorenzo D'Angelo, Martin Olsen, and Lorenzo Severini. Coverage Centrality Maximization in Undirected Networks. *CoRR*, abs/1811.04331, 2019. [arXiv:1811.04331](https://arxiv.org/abs/1811.04331).
- 14 P. de Waard. Nederland belastingparadijs voor veel multinationals. *De Volkskrant*, Oct. 14, 2011.
- 15 Erik D. Demaine and Morteza Zadimoghaddam. Minimizing the Diameter of a Network Using Shortcut Edges. In *SWAT*, volume 6139 of *Lecture Notes in Computer Science*, pages 420–431, 2010.
- 16 Edsger W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numer. Math.*, 1(1):269–271, 1959.
- 17 Iftah Gamzu and Danny Segev. A Sublogarithmic Approximation for Highway and Tollbooth Pricing. In *Proc. 37th International Colloquium Conference on Automata, Languages and Programming, ICALP'10*, pages 582–593, 2010.
- 18 Fabrizio Grandoni and Thomas Rothvoss. Pricing on Paths: A PTAS for the Highway Problem. *SIAM J. Comput.*, 45(2):216–231, 2016.
- 19 Venkatesan Guruswami, Jason D. Hartline, Anna R. Karlin, David Kempe, Claire Kenyon, and Frank McSherry. On Profit-maximizing Envy-free Pricing. In *SODA*, pages 1164–1173, 2005.
- 20 Vatche Ishakian, Dóra Erdős, Evimaria Terzi, and Azer Bestavros. A Framework for the Evaluation and Management of Network Centrality. In *Proc. 12th SIAM Int. Conf. on Data Mining (SDM)*, pages 427–438, 2012.
- 21 Martine Labbe, Patrice Marcotte, and Gilles Savard. A Bilevel Model of Taxation and Its Application to Optimal Highway Pricing. *Management Science*, 44(12):1608–1622, December 1998.
- 22 Ahmad Mahmoody, Charalampos E. Tsourakakis, and Eli Upfal. Scalable Betweenness Centrality Maximization via Sampling. In *Proc. 22nd ACM SIGKDD Int. Conf. on KDD*, pages 1765–1773, 2016.
- 23 Paolo Malighetti, Gianmaria Martini, Stefano Paleari, and Renato Redondi. The Impacts of Airport Centrality in the EU Network and Inter-Airport Competition on Airport Efficiency. Technical Report MPRA-7673, University Library of Munich, Germany, 2009.
- 24 Adam Meyerson and Brian Tagiku. Minimizing Average Shortest Path Distances via Shortcut Edge Addition. In *APPROX*, volume 5687, 2009.
- 25 George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. An analysis of approximations for maximizing submodular set functions—I. *Math. Program.*, 14(1):265–294, 1978.
- 26 Martin Olsen and Anastasios Viglas. On the approximability of the link building problem. *TCS*, 518:96–116, 2014.
- 27 Manos Papagelis, Francesco Bonchi, and Aristides Gionis. Suggesting Ghost Edges for a Smaller World. In *Proc. 20th ACM Int. Conf. on Information and Knowledge Management, CIKM '11*, pages 2305–2308. ACM, 2011.
- 28 Nikos Parotsidis, Evaggelia Pitoura, and Panayiotis Tsaparas. Selecting Shortcuts for a Smaller World. In *Proc. SIAM Int. Conf. on Data Mining*, pages 28–36. SIAM, 2015.
- 29 Senni Perumal, Prithwish Basu, and Ziyu Guan. Minimizing Eccentricity in Composite Networks via Constrained Edge Additions. In *Military Communications Conference, MILCOM 2013 - 2013 IEEE*, pages 1894–1899, 2013.
- 30 Sven C. Polak. Algorithms for the Network Analysis of Bilateral Tax Treaties. MSc. thesis, University of Amsterdam, The Netherlands, 2014.

- 31 Sébastien Roch, Gilles Savard, and Patrice Marcotte. An approximation algorithm for Stackelberg network pricing. *Networks*, 46(1):57–67, 2005.
- 32 D. Milmo S. Goodley. Dutch masters of tax avoidance. *The Guardian*, Oct. 19, 2011.
- 33 Thomas W. Valente and Kayo Fujimoto. Bridging: Locating critical connectors in a network. *Soc. Netw.*, 32(3):212–220, 2010.
- 34 Leslie Valiant. The Complexity of Enumeration and Reliability Problems. *SICOMP*, 8(3):410–421, 1979.
- 35 Maarten van ‘t Riet and Arjan M. Lejour. Ranking the Stars: Network Analysis of Bilateral Tax Treaties. CPB Discussion Paper, 2014.
- 36 Maarten van ‘t Riet and Arjan M. Lejour. Optimal tax routing: network analysis of FDI diversion. *International Tax and Public Finance*, 25(5):1321–1371, 2018.
- 37 Baoning Wu and Brian D. Davison. Identifying link farm spam pages. In *Proc. 14th Int. Conf. on World Wide Web, WWW 2005*, pages 820–829, 2005.

Slaying Hydrae: Improved Bounds for Generalized k -Server in Uniform Metrics

Marcin Bienkowski 

Institute of Computer Science, University of Wrocław, Poland
marcin.bienkowski@cs.uni.wroc.pl

Łukasz Jeż 

Institute of Computer Science, University of Wrocław, Poland
lukasz.jez@cs.uni.wroc.pl

Paweł Schmidt

Institute of Computer Science, University of Wrocław, Poland
pawel.schmidt@cs.uni.wroc.pl

Abstract

The generalized k -server problem is an extension of the weighted k -server problem, which in turn extends the classic k -server problem. In the generalized k -server problem, each of k servers s_1, \dots, s_k remains in its own metric space M_i . A request is a tuple (r_1, \dots, r_k) , where $r_i \in M_i$, and to service it, an algorithm needs to move at least one server s_i to the point r_i . The objective is to minimize the total distance traveled by all servers.

In this paper, we focus on the generalized k -server problem for the case where all M_i are uniform metrics. We show an $O(k^2 \cdot \log k)$ -competitive randomized algorithm improving over a recent result by Bansal et al. [SODA 2018], who gave an $O(k^3 \cdot \log k)$ -competitive algorithm. To this end, we define an abstract online problem, called Hydra game, and we show that a randomized solution of low cost to this game implies a randomized algorithm to the generalized k -server problem with low competitive ratio.

We also show that no randomized algorithm can achieve competitive ratio lower than $\Omega(k)$, thus improving the lower bound of $\Omega(k/\log^2 k)$ by Bansal et al.

2012 ACM Subject Classification Theory of computation \rightarrow Online algorithms

Keywords and phrases k -server, generalized k -server, competitive analysis

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.14

Funding Supported by Polish National Science Centre grant 2016/22/E/ST6/00499.

1 Introduction

The k -server problem, introduced by Manasse et al. [18], is one of the most well-studied and influential cornerstones of online analysis. The problem definition is deceptively simple: There are k servers, starting at a fixed set of k points of a metric space M . An input is a sequence of requests (points of M) and to service a request, an algorithm needs to move servers, so that at least one server ends at the request position. As typical for online problems, the k -server problem is sequential in nature: an online algorithm ALG learns a new request only after it services the current one. The cost of ALG, defined as the total distance traveled by all its servers, is then compared to the cost of an *offline* solution OPT; the ratio between them, called *competitive ratio*, is subject to minimization.

In a natural extension of the k -server problem, called the *generalized k -server problem* [16, 20], each server s_i remains in its own metric space M_i . The request is a k -tuple (r_1, \dots, r_k) , where $r_i \in M_i$, and to service it, an algorithm needs to move servers, so that *at least one* server s_i ends at the request position r_i . The original k -server problem corresponds to the



© Marcin Bienkowski, Łukasz Jeż, and Paweł Schmidt;
licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 14; pp. 14:1–14:14

Leibniz International Proceedings in Informatics



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

case where all metric spaces M_i are identical and each request is of the form (r, \dots, r) . The generalized k -server problem contains many known online problems, such as the weighted k -server problem [1, 7, 11, 12] or the CNN problem [8, 16, 19, 20] as special cases.

So far, the existence of an $f(k)$ -competitive algorithm for the generalized k -server problem in arbitrary metric spaces remains open. Furthermore, even for specific spaces, such as the line [16] or uniform metrics [1, 2, 16], the generalized k -server problem requires techniques substantially different from those used to tackle the classic k -server problems. For these reasons, studying this problem could lead to new techniques for designing online algorithms.

1.1 Previous Work

After almost three decades of extensive research counted in dozens of publications (see, e.g., a slightly dated survey by Koutsoupias [14]), we are closer to understanding the nature of the classic k -server problem. The competitive ratio achievable by deterministic algorithms is between k [18] and $2k - 1$ [15] with k -competitive algorithms known for special cases, such as uniform metrics [21], lines and trees [9, 10], or metrics of $k + 1$ points [18]. Less is known about competitive ratios for randomized algorithms: the best known lower bound holding for an arbitrary metric space is $\Omega(\log k / \log \log k)$ [4] and the currently best upper bound of $O(\log^6 k)$ has been recently obtained in a breakthrough result [6, 17].

In comparison, little is known about the generalized k -server problem. In particular, algorithms attaining competitive ratios that are functions of k exist only in a few special cases. The case of $k = 2$ has been solved by Sitters and Stougie [20, 19], who gave constant competitive algorithms for this setting. Results for $k \geq 3$ are known only for simpler metric spaces, as described below.

A uniform metric case describes a scenario where all metrics M_i are uniform with pairwise distances between different points equal to 1. For this case, Bansal et al. [2] recently presented an $O(k \cdot 2^k)$ -competitive deterministic algorithm and an $O(k^3 \cdot \log k)$ -competitive randomized one. The deterministic competitive ratio is at least $2^k - 1$ already when metrics M_i have two points [16]. Furthermore, using a straightforward reduction to the metrical task system (MTS) problem [5], they show that the randomized competitive ratio is at least $\Omega(k / \log k)$ [2].¹

A weighted uniform metric case describes a scenario where each metric M_i is uniform, but they have different scales, i.e., the pairwise distances between points of M_i are equal to some values $w_i > 0$. For this setting, Bansal et al. [2] gave an $2^{2^{O(k)}}$ -competitive deterministic algorithm extending an $2^{2^{O(k)}}$ -competitive algorithm for the weighted k -server problem in uniform metrics [12]. (The latter problem corresponds to the case where all requests are of the form (r, \dots, r) .) This matches a lower bound of $2^{2^{\Omega(k)}}$ [1] (which also holds already for the weighted k -server problem).

1.2 Our Results and Paper Organization

In this paper, we study the uniform metric case of the generalized k -server problem. We give a randomized $O(k^2 \cdot \log k)$ -competitive algorithm improving over the $O(k^3 \cdot \log k)$ bound by Bansal et al. [2].

¹ In fact, for the generalized k -server problem in uniform metrics, the paper by Bansal et al. [2] claims only the randomized lower bound of $\Omega(k / \log^2 k)$. To obtain it, they reduce the problem to the n -state metrical task system (MTS) problem and apply a lower bound of $\Omega(\log n / (\log \log n)^2)$ for MTS [3]. By using their reduction and a stronger lower bound of $\Omega(\log n / \log \log n)$ for n -state MTS [4], one could immediately obtain a lower bound of $\Omega(k / \log k)$ for the generalized k -server problem.

To this end, we first define an elegant abstract online problem: a *Hydra game* played by an online algorithm against an adversary on an unweighted tree. We present the problem along with a randomized, low-cost online algorithm HERC in Section 2. We defer a formal definition of the generalized k -server problem to Section 3.1. Later, in Section 3.2 and Section 3.3, we briefly sketch the structural claims concerning the generalized k -server problem given by Bansal et al. [2]. Using this structural information, in Section 3.4, we link the generalized k -server problem to the Hydra game: we show that a (randomized) algorithm of total cost R for the Hydra game on a specific tree (called *factorial tree*) implies a (randomized) $(R + 1)$ -competitive solution for the generalized k -server problem. This, along with the performance guarantees of HERC given in Section 2, yields the desired competitiveness bound. We remark that while the explicit definition of the Hydra game is new, the algorithm of Bansal et al. [2] easily extends to its framework.

Finally, in Section 4, we give an explicit lower bound construction for the generalized k -server problem, which does not use a reduction to the metrical task system problem, hereby improving the bound from $\Omega(k/\log k)$ to $\Omega(k)$.

2 Hydra Game

The Hydra game² is played between an online algorithm and an adversary on a fixed unweighted tree T , known to the algorithm in advance. The nodes of T have states which change throughout the game: Each node can be either *asleep*, *alive* or *dead*. Initially, the root r_T is alive and all other nodes are asleep. At all times, the following invariant is preserved: all ancestors of alive nodes are dead and all their descendants are asleep. In a single step, the adversary picks a single alive node w , kills it (changes its state to dead) and makes all its (asleep) children alive. Note that such adversarial move preserves the invariant above.

An algorithm must remain at some alive node (initially, it is at the root r_T). If an algorithm is at a node w that has just been killed, it has to move to any still alive node w' of its choice. For such movement it pays $\text{dist}(w, w')$, the length of the shortest path between w and w' in the tree T . The game ends when all nodes except one (due to the invariant, it has to be an alive leaf) are dead. Unlike many online problems, here our sole goal is to minimize the total (movement) cost of an online algorithm (i.e., without comparing it to the cost of the offline optimum).

This game is not particularly interesting in the deterministic setting: As an adversary can always kill the node where a deterministic algorithm resides, the algorithm has to visit all but one nodes of tree T , thus paying $\Omega(|T|)$. On the other hand, a trivial DFS traversal of tree T has the cost of $O(|T|)$. Therefore, we focus on randomized algorithms and assume that the adversary is oblivious: it knows an online algorithm, but not the random choices made by it thus far.

2.1 Randomized Algorithm Definition

It is convenient to describe our randomized algorithm HERC as maintaining a probability distribution η over set of nodes, where for any node u , $\eta(u)$ denotes the probability that HERC is at u . We require that $\eta(u) = 0$ for any non-alive node u . Whenever HERC decreases

² This is a work of science. Any resemblance of the process to the decapitation of a mythical many-headed serpent-shaped monster is purely coincidental.

the probability at a given node u by p and increases it at another node w by the same amount, we charge cost $p \cdot \text{dist}(u, w)$ to HERC. By a straightforward argument, one can convert such description into a more standard, “behavioral” one, which describes randomized actions conditioned on the current state of an algorithm, and show that the costs of both descriptions coincide. We present the argument in Appendix A for completeness.

At any time during the game, for any node u from tree T , $\text{rank}(u)$ denotes the number of non-dead (i.e., alive or asleep) leaves in the subtree rooted at u . As HERC knows tree T in advance, it knows node ranks as well. Algorithm HERC maintains η that is distributed over all alive nodes proportionally to their ranks. As all ancestors of an alive node are dead and all its descendants are asleep, we have $\eta(u) = \text{rank}(u)/\text{rank}(r_T)$ if u is alive and $\eta(u) = 0$ otherwise. In particular, at the beginning η is 1 at the root and 0 everywhere else.

While this already defines the algorithm, we still discuss its behavior when an alive node u is killed by the adversary. By HERC definition, we can think of the new probability distribution η' as obtained from η in the following way. First, HERC sets $\eta'(u) = 0$. Next, the probability distribution at other nodes is modified as follows.

Case 1. Node u is not a leaf. HERC distributes the probability of u among all (now alive) children of u proportionally to their ranks, i.e., sets $\eta'(w) = (\text{rank}(w)/\text{rank}(u)) \cdot \eta(u)$ for each child w of u .

Case 2. Node u is a leaf. Note that there were some other non-dead leaves, as otherwise the game would have ended before this step, and therefore $\eta(u) \leq 1/2$. HERC distributes $\eta(u)$ among all other nodes, scaling the probabilities of the remaining nodes up by a factor of $1/(1 - \eta(u))$. That is, it sets $\eta'(w) = \eta(w)/(1 - \eta(u))$ for any node w .

Note that in either case, η' is a valid probability distribution, i.e., all probabilities are non-negative and sum to 1. Moreover, η' is distributed over alive nodes proportionally to their new ranks, and is equal to zero at non-alive nodes.

► **Observation 1.** *At any time, the probability of an alive leaf u is exactly $\eta(u) = 1/\text{rank}(r_T)$.*

2.2 Analysis

For the analysis, we need a few more definitions. We denote the height and the number of the leaves of tree T by h_T and L_T , respectively. Let $\text{level}(u)$ denote the height of the subtree rooted at u , where leaves are at level 0. Note that $h_T = \text{level}(r_T)$.

To bound the cost of HERC, we define a potential Φ , which is a function of the current state of all nodes of T and the current probability distribution η of HERC. We show that Φ is initially $O(h_T \cdot (1 + \log L_T))$, is always non-negative, and the cost of each HERC’s action can be covered by the decrease of Φ . This will show that the total cost of HERC is at most the initial value of Φ , i.e., $O(h_T \cdot (1 + \log L_T))$.

Recall that $\eta(w) = 0$ for any non-alive node w and that $\text{rank}(u)$ is the number of non-dead leaves in the subtree rooted at u . Specifically, $\text{rank}(r_T)$ is the total number of non-dead leaves in T . The potential is defined as

$$\Phi = 4 \cdot h_T \cdot H(\text{rank}(r_T)) + \sum_{w \in T} \eta(w) \cdot \text{level}(w), \quad (1)$$

where $H(n) = \sum_{i=1}^n 1/i$ is the n -th harmonic number.

► **Lemma 2.** *At any time, $\Phi = O(h_T \cdot (1 + \log L_T))$.*

Proof. Since $\text{rank}(r_T) \leq L_T$ at all times, the first summand of Φ is $O(h_T \cdot \log L_T)$. The second summand of Φ is a convex combination of node levels, which range from 0 to h_T , and is thus bounded by h_T . ◀

► **Lemma 3.** Fix any step in which an adversary kills a node u and in result HERC changes the probability distribution from η to η' . Let ΔHERC be the cost incurred in this step by HERC and let $\Delta\Phi$ be the resulting change in the potential Φ . Then, $\Delta\Phi \leq -\Delta\text{HERC}$.

Proof. We denote the ranks before and after the adversarial event by rank and rank' , respectively. We consider two cases depending on the type of u .

Case 1. The killed node u is an internal node. In this case, $\Delta\text{HERC} = \eta(u)$ as HERC simply moves the total probability of $\eta(u)$ along a distance of one (from u to its children). As $\text{rank}'(r_T) = \text{rank}(r_T)$, the first summand of Φ remains unchanged. Let $C(u)$ be the set of children of u . Then,

$$\begin{aligned} \Delta\Phi &= \sum_{w \in T} (\eta'(w) - \eta(w)) \cdot \text{level}(w) = -\eta(u) \cdot \text{level}(u) + \sum_{w \in C(u)} \eta'(w) \cdot \text{level}(w) \\ &\leq -\eta(u) \cdot \text{level}(u) + \sum_{w \in C(u)} \eta'(w) \cdot (\text{level}(u) - 1) \\ &= -\eta(u) \cdot \text{level}(u) + \eta(u) \cdot (\text{level}(u) - 1) = -\Delta\text{HERC}, \end{aligned}$$

where the inequality holds as level of a node is smaller than the level of its parent and the penultimate equality follows as the whole probability mass at u is distributed to its children.

Case 2. The killed node u is a leaf. It is not the last alive node, as in such case the game would have ended before, i.e., it holds that $\text{rank}(r_T) \geq 2$. HERC moves the probability of $\eta(u) = 1/\text{rank}(r_T)$ (cf. Observation 1) along a distance of at most $2 \cdot h_T$, and thus $\Delta\text{HERC} \leq 2 \cdot h_T/\text{rank}(r_T)$.

Furthermore, for any $w \neq u$, $\eta'(w) = \eta(w)/(1 - \eta(u))$. Using $\eta(u) = 1/\text{rank}(r_T)$, we infer that the probability at a node $w \neq u$ increases by

$$\begin{aligned} \eta'(w) - \eta(w) &= \left(\frac{1}{1 - \eta(u)} - 1 \right) \cdot \eta(w) = \frac{\eta(u)}{1 - \eta(u)} \cdot \eta(w) \\ &= \frac{1}{\text{rank}(r_T) - 1} \cdot \eta(w) \leq \frac{2}{\text{rank}(r_T)} \cdot \eta(w), \end{aligned} \tag{2}$$

where the last inequality follows as $\text{rank}(r_T) \geq 2$.

Using (2) and the relation $\text{rank}'(r_T) = \text{rank}(r_T) - 1$ (the number of non-dead leaves decreases by 1), we compute the change of the potential:

$$\begin{aligned} \Delta\Phi &= 4 \cdot h_T \cdot (H(\text{rank}'(r_T)) - H(\text{rank}(r_T))) + \sum_{w \in T} (\eta'(w) - \eta(w)) \cdot \text{level}(w) \\ &= -\frac{4 \cdot h_T}{\text{rank}(r_T)} + (\eta'(u) - \eta(u)) \cdot \text{level}(u) + \sum_{w \neq u} (\eta'(w) - \eta(w)) \cdot \text{level}(w) \\ &\leq -\frac{4 \cdot h_T}{\text{rank}(r_T)} + \sum_{w \neq u} \frac{2}{\text{rank}(r_T)} \cdot \eta(w) \cdot h_T \leq -\frac{2 \cdot h_T}{\text{rank}(r_T)} \leq -\Delta\text{HERC}. \end{aligned}$$

In the first inequality, we used that $\text{level}(u) = 0$ and $\text{level}(w) \leq h_T$ for any w . Summing up, we showed that $\Delta\Phi \leq -\Delta\text{HERC}$ in both cases. ◀

► **Theorem 4.** For the Hydra game played on any tree T of height h_T and L_T leaves, the total cost of HERC is at most $O(h_T \cdot (1 + \log L_T))$.

Proof. Let Φ_B denote the initial value of Φ . By non-negativity of Φ and Lemma 3, it holds that the total cost of HERC is at most Φ_B . The latter amount is at most $O(h_T \cdot (1 + \log L_T))$ by Lemma 2. \blacktriangleleft

Although HERC and Theorem 4 may seem simple, when applied to appropriate trees, they yield improved bounds for the generalized k -server problem in uniform metrics, as shown in the next section.

3 Improved Algorithm for Generalized k -Server Problem

In this part, we show how any solution for the Hydra game on a specific tree (defined later) implies a solution to the generalized k -server problem in uniform metrics. This will yield an $O(k^2 \log k)$ -competitive randomized algorithm for the generalized k -server problem, improving the previous bound of $O(k^3 \cdot \log k)$ [2]. We note that this reduction is implicit in the paper of Bansal et al. [2], so our contribution is in formalizing the Hydra game and solving it more efficiently.

3.1 Preliminaries

The generalized k -server problem in uniform metrics is formally defined as follows. The offline part of the input comprises k uniform metric spaces M_1, \dots, M_k . The metric M_i has $n_i \geq 2$ points, the distance between each pair of its points is 1. There are k servers denoted s_1, \dots, s_k , the server s_i starts at some fixed point in M_i and always remains at some point of M_i .

The online part of the input is a sequence of requests, each request being a k -tuple $(r_1, \dots, r_k) \in \prod_{i=1}^k M_i$. To service a request, an algorithm needs to move its servers, so that at least one server s_i ends at the request position r_i . Only after the current request is serviced, an online algorithm is given the next one.

The cost of an algorithm ALG on input I , denoted $\text{ALG}(I)$, is the total distance traveled by all its k servers. We say that a randomized online algorithm ALG is β -competitive if there exists a constant γ , such that for any input I , it holds that $\mathbf{E}[\text{ALG}(I)] \leq \beta \cdot \text{OPT}(I) + \gamma$, where the expected value is taken over all random choices of ALG, and where $\text{OPT}(I)$ denotes the cost of an optimal *offline* solution for input I . The constant γ may be a function of k , but it cannot depend on an online part of the input.

3.2 Phase-Based Approach

We start by showing how to split the sequence of requests into phases. To this end, we need a few more definitions. A (server) *configuration* is a k -tuple $c = (c_1, \dots, c_k) \in \prod_{i=1}^k M_i$, denoting positions of respective servers. For a request $r = (r_1, \dots, r_k) \in \prod_{i=1}^k M_i$, we define the set of *compatible* configurations $\text{comp}(r) = \{(c_1, \dots, c_k) : \exists_i c_i = r_i\}$, i.e., the set of all configurations that can service the request r without moving a server. Other configurations we call *incompatible* with r .

An input is split into phases, with the first phase starting with the beginning of an input. The phase division process described below is constructed to ensure that OPT pays at least 1 in any phase, perhaps except the last one. At the beginning of a phase, all configurations are *phase-feasible*. Within a phase, upon a request r , all configurations incompatible with r become *phase-infeasible*. The phase ends once all configurations are phase-infeasible; if this is not the end of the input, the next phase starts immediately, i.e., all configurations are

restored to the phase-feasible state before the next request. Note that the description above is merely a way of splitting an input into phases and marking configurations as phase-feasible and phase-infeasible. The actual description of an online algorithm will be given later.

Fix any finished phase and any configuration c and consider an algorithm that starts the phase with its servers at configuration c . When configuration c becomes phase-infeasible, such algorithm is forced to move and pay at least 1. As each configuration eventually becomes phase-infeasible in a finished phase, any algorithm (even OPT) must pay at least 1 in any finished phase. Hence, if the cost of a phase-based algorithm for servicing requests of a single phase can be bounded by $f(k)$, the competitive ratio of this algorithm is then at most $f(k)$.

3.3 Configuration Spaces

Phase-based algorithms that we construct will not only track the set of phase-feasible configurations, but they will also group these configurations in certain sets, called *configuration spaces*.

To this end, we introduce a special *wildcard* character \star . Following [2], for any k -tuple $q = (q_1, \dots, q_k) \in \prod_{i=1}^k (M_i \cup \{\star\})$, we define a (*configuration*) *space* $S[q] = \{(c_1, \dots, c_k) \in \prod_{i=1}^k M_i : \forall_i c_i = q_i \vee q_i = \star\}$. A coordinate with $q_i = \star$ is called *free* for the configuration space $S[q]$. That is, $S[q]$ contains all configurations that agree with q on all non-free coordinates.

The number of free coordinates in q defines the *dimension* of $S[q]$ denoted $\dim(S[q])$. Observe that the k -dimensional space $S[(\star, \dots, \star)]$ contains all configurations. If tuple q has no \star at any position, then $S[q]$ is 0-dimensional and contains only (configuration) q . The following lemma, proven by Bansal et al. [2], follows immediately from the definition of configuration spaces.

► **Lemma 5** (Lemma 3.1 of [2]). *Let $S[q]$ be a d -dimensional configuration space (for some $d \geq 0$) whose all configurations are phase-feasible. Fix a request r . If there exists a configuration in $S[q]$ that is not compatible with r , then there exist d (not necessarily disjoint) subspaces $S[q_1], \dots, S[q_d]$, each of dimension $d - 1$, such that $\bigcup_i S[q_i] = S[q] \cap \text{comp}(r)$. Furthermore, for all i , the k -tuples q_i and q differ exactly at one position.*

Using the lemma above, we may describe a way for an online algorithm to keep track of all phase-feasible configurations. To this end, it maintains a set \mathcal{A} of (not necessarily disjoint) configuration spaces, such that their union is exactly the set of all phase-feasible configurations. We call spaces from \mathcal{A} *alive*.

At the beginning, $\mathcal{A} = \{S[(\star, \dots, \star)]\}$. Assume now that a request r makes some configurations from a d -dimensional space $S[q] \in \mathcal{A}$ phase-infeasible. (A request may affect many spaces from \mathcal{A} ; we apply the described operations to each of them sequentially in an arbitrary order.) In such case, $S[q]$ stops to be alive, it is removed from \mathcal{A} and till the end of the phase it will be called *dead*. Next, we apply Lemma 5 to $S[q]$, obtaining d configuration spaces $S[q_1], \dots, S[q_d]$, such that their union is $S[q] \cap \text{comp}(r)$, i.e., contains all those configurations from $S[q]$ that remain phase-feasible. We make all spaces $S[q_1], \dots, S[q_d]$ alive and we insert them into \mathcal{A} . (Note that when $d = 0$, set $S[q]$ is removed from \mathcal{A} , but no space is added to it.) This way we ensure that the union of spaces from \mathcal{A} remains equal to the set of all phase-feasible configurations. Note that when a phase ends, \mathcal{A} becomes empty. We emphasize that the evolution of set \mathcal{A} within a phase depends only on the sequence of requests and not on the particular behavior of an online algorithm.

3.4 Factorial Trees: From Hydra Game to Generalized k -Server

Given the framework above, an online algorithm may keep track of the set of alive spaces \mathcal{A} , and at all times try to be in a configuration from some alive space. If this space becomes dead, an algorithm changes its configuration to any configuration from some other alive space from \mathcal{A} .

The crux is to choose an appropriate next alive space. To this end, our algorithm for the generalized k -server problem will internally run an instance of the Hydra game (a new instance for each phase) on a special tree, and maintain a mapping from alive and dead spaces to alive and dead nodes in the tree. Moreover, spaces that are created during the algorithm runtime, as described in Section 3.3, have to be dynamically mapped to tree nodes that were so far asleep.

In our reduction, we use a k -factorial tree. It has height k (the root is on level k and leaves on level 0). Any node on level d has exactly d children, i.e., the subtree rooted at a d -level node has $d!$ leaves, hence the tree name. On the k -factorial tree, the total cost of HERC is $O(k \cdot (1 + \log k!)) = O(k^2 \cdot \log k)$. We now show that this implies an improved algorithm for the generalized k -server problem.

► **Theorem 6.** *If there exists a (randomized) online algorithm H for the Hydra game on the k -factorial tree of total (expected) cost R , then there exists a (randomized) $(R+1)$ -competitive online algorithm G for the generalized k -server problem in uniform metrics.*

Proof. Let I be an input for the generalized k -server problem in uniform metric spaces. G splits I into phases as described in Section 3.2 and, in each phase, it tracks the phase-feasible nodes using set \mathcal{A} of alive spaces as described in Section 3.3. For each phase, G runs a new instance I_H of the Hydra game on a k -factorial tree T , translates requests from I to adversarial actions in I_H , and reads the answers of H executed on I_H . At all times, G maintains a (bijective) mapping from alive (respectively, dead) d -dimensional configuration spaces to alive (respectively, dead) nodes on the d -th level of the tree T . In particular, at the beginning, the only alive space is the k -dimensional space $S[(\star, \dots, \star)]$, which corresponds to the tree root (on level k). The configuration of G will always be an element of the space corresponding to the tree node containing H. More precisely, within each phase, a request r is processed in the following way by G.

- Suppose that request r does not make any configuration phase-infeasible. In this case, G services r from its current configuration and no changes are made to \mathcal{A} . Also no adversarial actions are executed in the Hydra game.
- Suppose that request r makes some (but not all) configurations phase-infeasible. We assume that this kills only one d -dimensional configuration space $S[q]$. (If r causes multiple configuration spaces to become dead, G processes each such killing event separately, in an arbitrary order.)

By the description given in Section 3.3, $S[q]$ is then removed from \mathcal{A} and d new $(d-1)$ -dimensional spaces $S[q_1], \dots, S[q_d]$ are added to \mathcal{A} . G executes appropriate adversarial actions in the Hydra game: a node v corresponding to $S[q]$ is killed and its d children on level $d-1$ change state from asleep to alive. G modifies the mapping to track the change of \mathcal{A} : (new and now alive) spaces $S[q_1], \dots, S[q_d]$ become mapped to (formerly asleep and now alive) d children of v . Afterwards, G observes the answer of algorithm H on the factorial tree and replays it. Suppose H moves from (now dead) node v to an alive node v' , whose corresponding space is $S[q'] \in \mathcal{A}$. In this case, G changes its configuration to the closest configuration (requiring minimal number of server moves) from $S[q']$. It remains to relate its cost to the cost of H. By Lemma 5 (applied to spaces corresponding to all nodes

on the tree path from v to v' , the corresponding k -tuples q, q' differ on at most $\text{dist}(v, v')$ positions. Therefore, adjusting the configuration of G , so that it becomes an element of $S[q']$, requires at most $\text{dist}(v, v')$ server moves, which is exactly the cost of H .

Finally, note that when G processes all killing events, it ends in a configuration of an alive space, and hence it can service the request r from its new configuration.

- Suppose that request r makes all remaining configurations phase-infeasible. In such case, G moves an arbitrary server to service this request, which incurs a cost of 1. In this case, the current phase ends, a new one begins, and G initializes a new instance of the Hydra game.

Let $f \geq 1$ be the number of all phases for input I (the last one may be not finished). The cost of OPT in a single finished phase is at least 1. By the reasoning above, the (expected) cost of G in a single phase is at most $R + 1$. Therefore, $\mathbf{E}[G(I)] \leq (R + 1) \cdot f \leq (R + 1) \cdot \text{OPT}(I) + (R + 1)$, which completes the proof. ◀

Using our algorithm HERC for the Hydra game along with the reduction given by Theorem 6 immediately implies the following result.

► **Corollary 7.** *There exists a randomized $O(k^2 \cdot \log k)$ -competitive online algorithm for the generalized k -server problem in uniform metrics.*

4 Lower bound

Next, we show that that competitive ratio of any (even randomized) online algorithm for the generalized k -server problem in uniform metrics is at least $\Omega(k)$, as long as each metric space M_i contains at least two points. For each M_i , we choose two distinct points, the initial position of the i -th server, which we denote 0 and any other point, which we denote 1. The adversary is going to issue only requests satisfying $r_i \in \{0, 1\}$ for all i , hence without loss of generality any algorithm will restrict its server's position in each M_i to 0 and 1. (To see this, assume without loss of generality that the algorithm is lazy, i.e., it is only allowed to move when a request is not covered by any of its server, and is then allowed only to move a single server to cover that request.) For this reason, from now on we assume that $M_i = \{0, 1\}$ for all i , ignoring superfluous points of the metrics.

The configuration of any algorithm can be then encoded using a binary word of length k . It is convenient to view all these 2^k words (configurations) as nodes of the k -dimensional hypercube: two words are connected by a hypercube edge if they differ at exactly one position. Observe that a cost of changing configuration c to c' , denoted $\text{dist}(c, c')$ is exactly the distance between c and c' in the hypercube, equal to the number of positions on which the corresponding binary strings differ.

In our construction, we compare the cost of an online algorithm to the cost of an algorithm provided by the adversary. Since OPT 's cost can be only lower than the latter, such approach yields a lower bound on the performance of the online algorithm.

For each word w , there is exactly one word at distance k , which we call its *antipode* and denote \bar{w} . Clearly, $\bar{w}_i = 1 - w_i$ for all i . Whenever we say that an adversary *penalizes* configuration c , it issues a request at \bar{c} . An algorithm that has servers at configuration c needs to move at least one of them. On the other hand, any algorithm with servers at configuration $c' \neq c$ need not move its servers; this property will be heavily used by an adversary's algorithm.

4.1 A Warm-Up: Deterministic Algorithms

To illustrate our general framework, we start with a description of an $\Omega(2^k/k)$ lower bound that holds for any deterministic algorithm DET [2]. (A more refined analysis yields a better lower bound of $2^k - 1$ [16].) The adversarial strategy consists of a sequence of independent identical phases. Whenever DET is in some configuration, the adversary penalizes this configuration. The phase ends when $2^k - 1$ *different* configurations have been penalized. This means that DET was forced to move at least $2^k - 1$ times, at a total cost of at least $2^k - 1$. In the same phase, the adversary's algorithm makes only a single move (of cost at most k) at the very beginning of the phase: it moves to the only configuration that is not going to be penalized in the current phase. This shows that the DET-to-OPT ratio in each phase is at least $(2^k - 1)/k$.

4.2 Extension to Randomized Algorithms

Adopting the idea above to a randomized algorithm RAND is not straightforward. Again, we focus on a single phase and the adversary wants to leave (at least) one configuration non-penalized in this phase. However, now the adversary only knows RAND's probability distribution μ over configurations and not its actual configuration. (At any time, for any configuration c , $\mu(c)$ is the probability that RAND's configuration is equal to c .) We focus on a greedy adversarial strategy that always penalizes the configuration with maximum probability. However, arguing that RAND incurs a significant cost is not as easy as for DET.

First, the support of μ can also include configurations that have been already penalized by the adversary in the current phase. This is but a nuisance, easily overcome by penalizing such configurations repeatedly if RAND keeps using them, until their probability becomes negligible. Therefore, in this informal discussion, we assume that once a configuration c is penalized in a given phase, $\mu(c)$ remains equal to zero.

Second, a straightforward analysis of the greedy adversarial strategy fails to give a non-trivial lower bound. Assume that $i \in \{0, \dots, 2^k - 2\}$ configurations have already been penalized in a given phase, and the support of μ contains the remaining $2^k - i$ configurations. The maximum probability assigned to one of these configurations is at least $1/(2^k - i)$. When such configuration is penalized, RAND needs to move at least one server with probability at least $1/(2^k - i)$. With such bounds, we would then prove that the algorithm's expected cost is at least $\sum_{i=0}^{2^k-2} 1/(2^k - i) = \Omega(\log 2^k) = \Omega(k)$. Since we bounded the adversary's cost per phase by k , this gives only a constant lower bound.

What we failed to account is that the actual distance traveled by RAND in a single step is either larger than 1 or RAND would not be able to maintain a uniform distribution over non-penalized configurations. However, actually exploiting this property seems quite complex, and therefore we modify the adversarial strategy instead.

The crux of our actual construction is choosing a subset Q of the configurations, such that Q is sufficiently large (we still have $\log(|Q|) = \Omega(k)$), but the minimum distance between any two points of Q is $\Omega(k)$. Initially, the adversary forces the support of μ to be contained in Q . Afterwards, the adversarial strategy is almost as described above, but reduced to set Q only. This way, in each step the support of μ is a set $S \subseteq Q$, and the adversary forces RAND to move with probability at least $1/|S|$ over a distance at least $\Omega(k)$, which is the extra $\Theta(k)$ factor. We begin by proving the existence of such a set Q for sufficiently large k . The proof is standard (see, e.g., Chapter 17 of [13]); we give it below for completeness.

► **Lemma 8.** *For any $k \geq 16$, there exists a set $Q \subseteq \{0, 1\}^k$ of binary words of length k , satisfying the following two properties:*

size property: $|Q| \geq 2^{k/2}/k$,

distance property: $\text{dist}(v, w) \geq k/16$ for any $v, w \in Q$.

Proof. Let $\ell = \lfloor k/16 \rfloor \geq k/32$. For any word q , we define its ℓ -neighborhood $B_\ell(q) = \{w : \text{dist}(q, w) \leq \ell\}$.

We construct set Q greedily. We maintain set Q and set $\Gamma(Q) = \bigcup_{q \in Q} B_\ell(q)$. We start with $Q = \emptyset$ (and thus with $\Gamma(Q) = \emptyset$). In each step, we extend Q with an arbitrary word $w \in \{0, 1\}^k \setminus \Gamma(Q)$ and update $\Gamma(Q)$ accordingly. We proceed until set $\Gamma(Q)$ contains all possible length- k words. Clearly, the resulting set Q satisfies the distance property.

It remains to show that $|Q| \geq 2^{k/2}/k$. For a word q , the size of $B_\ell(q)$ is

$$\begin{aligned} |B_\ell(q)| &= \sum_{i=0}^{\lfloor k/16 \rfloor} \binom{k}{i} < k \cdot \binom{k}{\lfloor k/16 \rfloor} \leq k \cdot \left(\frac{k \cdot e}{\lfloor k/16 \rfloor} \right)^{\lfloor k/16 \rfloor} \\ &\leq k \cdot \left(\frac{k \cdot e}{k/32} \right)^{k/16} = k \cdot \left((32 \cdot e)^{1/8} \right)^{k/2} < k \cdot 2^{k/2}. \end{aligned}$$

That is, in a single step, $\Gamma(Q)$ increases by at most $k \cdot 2^{k/2}$ elements. Therefore, the process continues for at least $2^k / (k \cdot 2^{k/2}) = 2^{k/2}/k$ steps, and thus the size of Q is at least $2^{k/2}/k$. ◀

► **Theorem 9.** *The competitive ratio of every (randomized) online algorithm solving the generalized k -server problem in uniform metrics is at least $\Omega(k)$.*

Proof. In the following we assume that $k \geq 16$, otherwise the theorem follows trivially. We fix any randomized online algorithm RAND. The lower bound strategy consists of a sequence of independent phases. Requests of each phase can be (optimally) serviced with cost at most k and we show that RAND's expected cost for a single phase is $\Omega(k^2)$, i.e., the ratio between these costs is $\Omega(k)$. As the adversary may present an arbitrary number of phases to the algorithm, this shows that the competitive ratio of RAND is $\Omega(k)$, i.e., by making the cost of RAND arbitrarily high, the additive constant in the definition of the competitive ratio (cf. Section 3.1) becomes negligible.

As in our informal introduction, $\mu(c)$ denotes the probability that RAND has its servers in configuration c (at time specified in the context). We extend the notion μ to sets, i.e., $\mu(X) = \sum_{c \in X} \mu(c)$ where X is a set of configurations. We denote the complement of X (to $\prod_{i=1}^k M_i$) by X^C . We use $\varepsilon = 2^{-(2k+2)}$ throughout the proof.

To make the description concise, we define an auxiliary routine CONFINE(X) for the adversary (for some configuration set X). In this routine, the adversary repeatedly checks whether there exists a configuration $c \notin X$, such that $\mu(c) > \varepsilon$. In such case, it penalizes c ; if no such configuration exists, the routine terminates. We may assume that the procedure always terminates after finite number of steps, as otherwise RAND's competitive ratio would be unbounded. (RAND pays at least ε in each step of the routine while an adversary's algorithm may move its servers to any configuration from set X , and from that time service all requests of CONFINE(X) with no cost.)

The adversarial strategy for a single phase is as follows. First, it constructs Q_1 as the configuration set fulfilling the properties of Lemma 8; let m denote its cardinality. The phase consists then of m executions of CONFINE routine: CONFINE(Q_1), CONFINE(Q_2), ..., CONFINE(Q_m). For $i \in \{2, \dots, m\}$, set Q_i is defined in the following way. The adversary observes RAND's distribution μ right after routine CONFINE(Q_{i-1}) terminates; at this point this distribution is denoted μ_{i-1} . Then, the adversary picks configuration c_{i-1} to be the element of Q_{i-1} that maximizes the probability μ_{i-1} , and sets $Q_i = Q_{i-1} \setminus \{c_{i-1}\}$.

We begin by describing the way that the adversary services the requests. Observe that set Q_m contains a single configuration, henceforth denoted c^* . The configuration c^* is contained in all sets Q_1, \dots, Q_m , and thus c^* is never penalized in the current phase. Hence, by moving to c^* at the beginning of the phase, which costs at most k , and remaining there till the phase ends, the adversary's algorithm services all phase requests at no further cost.

It remains to lower-bound the cost of RAND . $\text{CONFINE}(Q_1)$ may incur no cost; its sole goal is to confine the support of μ to Q_1 . Now, we fix any $i \in \{2, \dots, m\}$ and estimate the cost incurred by $\text{CONFINE}(Q_i)$. Recall that the probability distribution right before $\text{CONFINE}(Q_i)$ starts (and right after $\text{CONFINE}(Q_{i-1})$ terminates) is denoted μ_{i-1} and the distribution right after $\text{CONFINE}(Q_i)$ terminates is denoted μ_i .

During $\text{CONFINE}(Q_i)$ a probability mass $\mu_{i-1}(c_{i-1})$, is moved from c_{i-1} to nodes of set Q_i (recall that $Q_i \uplus \{c_{i-1}\} = Q_{i-1}$). Some negligible amounts (at most $\mu_i(Q_i^C)$) of this probability may however remain outside of Q_i after $\text{CONFINE}(Q_i)$ terminates. That is, RAND moves at least the probability mass of $\mu_{i-1}(c_{i-1}) - \mu_i(Q_i^C)$ from configuration c_{i-1} to configurations from Q_i (i.e., along a distance of at least $\text{dist}(c_{i-1}, Q_i)$). Therefore, its expected cost due to $\text{CONFINE}(Q_i)$ is at least $(\mu_{i-1}(c_{i-1}) - \mu_i(Q_i^C)) \cdot \text{dist}(c_{i-1}, Q_i)$.

First, using the properties of $\text{CONFINE}(Q_{i-1})$ and the definition of c_{i-1} , we obtain

$$\mu_{i-1}(c_{i-1}) \geq \frac{\mu_{i-1}(Q_{i-1})}{|Q_{i-1}|} = \frac{1 - \mu_{i-1}(Q_{i-1}^C)}{|Q_{i-1}|} \geq \frac{1 - |Q_{i-1}^C| \cdot \varepsilon}{|Q_{i-1}|} > \frac{1 - 2^{-(k+2)}}{|Q_{i-1}|}. \quad (3)$$

Second, using the properties of $\text{CONFINE}(Q_i)$ yields

$$\mu_i(Q_i^C) \leq |Q_i^C| \cdot \varepsilon < 2^{-(k+2)} = \frac{2^{-2}}{2^k} < \frac{2^{-2}}{|Q_{i-1}|}. \quad (4)$$

Using (3) and (4), we bound the expected cost of RAND due to routine $\text{CONFINE}(Q_i)$ as

$$\begin{aligned} \mathbf{E}[\text{RAND}(\text{CONFINE}(Q_i))] &\geq (\mu_{i-1}(c_{i-1}) - \mu_i(Q_i^C)) \cdot \text{dist}(c_{i-1}, Q_i) \\ &\geq \left(\frac{1 - 2^{-(k+2)}}{|Q_i|} - \frac{2^{-2}}{|Q_i|} \right) \cdot \frac{k}{16} \geq \frac{1}{2 \cdot |Q_i|} \cdot \frac{k}{16} \\ &= k / (32 \cdot (m - i + 1)) \end{aligned} \quad (5)$$

The second inequality above follows as all configurations from $\{c_{i-1}\} \uplus Q_i$ are distinct elements of Q_1 , and hence their mutual distance is at least $k/16$ by the distance property of Q_1 (cf. Lemma 8). By summing (5) over $i \in \{2, \dots, m\}$, we obtain that the total cost of RAND in a single phase is $\mathbf{E}[\text{RAND}] \geq \sum_{i=2}^m \mathbf{E}[\text{RAND}(\text{CONFINE}(Q_i))] \geq \frac{k}{32} \cdot \sum_{i=2}^m \frac{1}{m-i+1} = \Omega(k \cdot \log m) = \Omega(k^2)$. The last equality holds as $m \geq 2^{k/2}/k$ by the size property of Q_1 . (cf. Lemma 8). \blacktriangleleft

5 Final remarks

In this paper, we presented an abstract Hydra game whose solution we applied to create an algorithm for the generalized k -server problem. Any improvement of our HERC strategy for the Hydra game would yield an improvement for the generalized k -server problem. However, we may show that on a wide class of trees (that includes factorial trees used in our reduction), HERC is optimal up to a constant factor. Thus, further improving our upper bound of $O(k^2 \log k)$ for the generalized k -server problem will require another approach.

A lower bound for the cost of any randomized strategy for the Hydra game is essentially the same as our single-phase construction from Section 4.2 for the generalized k -server problem. That is, the adversary fixes a subset Q of tree leaves, makes only nodes of Q alive

(this forces the algorithm to be inside set Q), and then iteratively kills nodes of Q where the algorithm is most likely to be. As in the proof from Section 4.2, such adversarial strategy incurs the cost of $\Omega(\text{mindist}(Q) \cdot \log |Q|)$, where $\text{mindist}(Q) = \min_{u \neq v \in Q} \text{dist}(u, v)$.

The construction of appropriate Q for a tree T of depth $k = h_T$ (be either the k -factorial tree or the complete k -ary tree) is as follows. Let Z be the set of all nodes of T at level $\lfloor k/2 \rfloor$; for such trees, $\log |Z| = \Omega(\log L_T)$. Let Q consist of $|Z|$ leaves of the tree, one per node of Z chosen arbitrarily from its subtree. Then, $\text{mindist}(Q) = \Omega(h_T)$ and $\log |Q| = \Omega(\log L_T)$, and thus the resulting lower bound $\Omega(h_T \cdot \log L_T)$ on the cost asymptotically matches the performance of HERC from Theorem 4.

References

- 1 Nikhil Bansal, Marek Eliás, and Grigorios Koumoutsos. Weighted k -Server Bounds via Combinatorial Dichotomies. In *Proc. 58th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 493–504. IEEE Computer Society, 2017.
- 2 Nikhil Bansal, Marek Eliás, Grigorios Koumoutsos, and Jesper Nederlof. Competitive Algorithms for Generalized k -Server in Uniform Metrics. In *Proc. 29th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 992–1001, 2018.
- 3 Yair Bartal, Béla Bollobás, and Manor Mendel. Ramsey-type theorems for metric spaces with applications to online problems. *J. Comput. Syst. Sci.*, 72(5):890–921, 2006.
- 4 Yair Bartal, Nathan Linial, Manor Mendel, and Assaf Naor. On metric Ramsey-type phenomena. In *Proc. 35th ACM Symp. on Theory of Computing (STOC)*, pages 463–472, 2003.
- 5 Alan Borodin, Nati Linial, and Michael E. Saks. An optimal on-line algorithm for metrical task system. *Journal of the ACM*, 39(4):745–763, 1992.
- 6 Sébastien Bubeck, Michael B. Cohen, Yin Tat Lee, James R. Lee, and Aleksander Madry. k -server via multiscale entropic regularization. In *Proc. 50th ACM Symp. on Theory of Computing (STOC)*, pages 3–16. ACM, 2018.
- 7 Ashish Chiplunkar and Sundar Vishwanathan. On Randomized Memoryless Algorithms for the Weighted K -Server Problem. In *Proc. 54th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 11–19, 2013.
- 8 Marek Chrobak. SIGACT news online algorithms col. 1. *SIGACT News*, 34(4):68–77, 2003.
- 9 Marek Chrobak, Howard J. Karloff, Thomas H. Payne, and Sundar Vishwanathan. New Results on Server Problems. *SIAM Journal on Discrete Mathematics*, 4(2):172–181, 1991.
- 10 Marek Chrobak and Lawrence L. Larmore. An Optimal On-Line Algorithm for k -Servers on Trees. *SIAM Journal on Computing*, 20(1):144–148, 1991.
- 11 Marek Chrobak and Jirí Sgall. The weighted 2-server problem. *Theoretical Computer Science*, 324(2-3):289–312, 2004.
- 12 Amos Fiat and Moty Ricklin. Competitive Algorithms for the Weighted Server Problem. *Theoretical Computer Science*, 130(1):85–99, 1994.
- 13 Stasys Jukna. *Extremal Combinatorics*. Springer, 2011.
- 14 Elias Koutsoupias. The k -server problem. *Computer Science Review*, 3(2):105–118, 2009.
- 15 Elias Koutsoupias and Christos H. Papadimitriou. On the k -Server Conjecture. *Journal of the ACM*, 42(5):971–983, 1995.
- 16 Elias Koutsoupias and David Scot Taylor. The CNN problem and other k -server variants. *Theoretical Computer Science*, 324(2-3):347–359, 2004.
- 17 James R. Lee. Fusible HSTs and the Randomized k -Server Conjecture. In *Proc. 59th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 438–449, 2018.
- 18 Mark S. Manasse, Lyle A. McGeoch, and Daniel D. Sleator. Competitive algorithms for server problems. *Journal of the ACM*, 11(2):208–230, 1990.
- 19 René Sitters. The Generalized Work Function Algorithm Is Competitive for the Generalized 2-Server Problem. *SIAM Journal on Computing*, 43(1):96–125, 2014.

- 20 René A. Sitters and Leen Stougie. The generalized two-server problem. *Journal of the ACM*, 53(3):437–458, 2006.
- 21 Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

A Probability Distribution and Algorithms

When we described our algorithm HERC for the Hydra game, we assumed that its current position in the tree is a random node with probability distribution given by η . In a single step, HERC decreases probability at some node u from $\eta(u)$ to zero and increases the probabilities of some other nodes w_1, \dots, w_ℓ by a total amount of $\eta(u)$. Such change can be split into ℓ elementary changes, each decreasing the probability at node u by p_i and increasing it at node w_i by the same amount. Each elementary change can be executed and analyzed as shown in the following lemma.

► **Lemma 10.** *Let η be a probability distribution describing the position of ALG in the tree. Fix two tree nodes, u and w . Suppose η' is a probability distribution obtained from η by decreasing $\eta(u)$ by p and increasing $\eta(w)$ by p . Then, ALG can change its random position, so that it will be described by η' , and the expected cost of such change is $p \cdot \text{dist}(u, w)$.*

Proof. We define ALG's action as follows: if ALG is at node u , then with probability $p/\eta(u)$ it moves to node w . If ALG is at some other node it does not change its position.

We observe that the new distribution of ALG is exactly η' . Indeed, the probability of being at node u decreases by $\eta(u) \cdot p/\eta(u) = p$, while the probability of being at node w increases by the same amount. The probabilities for all nodes different than u or w remain unchanged.

Furthermore, the probability that ALG moves is $\eta(u) \cdot (p/\eta(u)) = p$ and the traveled distance is $\text{dist}(u, w)$. The expected cost of the move is then $p \cdot \text{dist}(u, w)$, as desired. ◀

Measure and Conquer for Max Hamming Distance XSAT

Gordon Hoi

School of Computing, National University of Singapore,
13 Computing Drive, Block COM1, Singapore 117417, Republic of Singapore
e0013185@u.nus.edu

Frank Stephan

Department of Mathematics, National University of Singapore,
10 Lower Kent Ridge Road, Block S17, Singapore 119076, Republic of Singapore
School of Computing, National University of Singapore,
13 Computing Drive, Block COM1, Singapore 117417, Republic of Singapore
fstephan@comp.nus.edu.sg

Abstract

XSAT is defined as the following: Given a propositional formula in conjunctive normal form, can one find an assignment to variables such that there is exactly only 1 literal that is true in every clause, while the other literals are false. The decision problem XSAT is known to be **NP**-complete. Crescenzi and Rossi [12] introduced the variant where one searches for a pair of two solutions of an X3SAT instance with maximal Hamming Distance among them, that is, one wants to identify the largest number k such that there are two solutions of the instance with Hamming Distance k . Dahllöf [15, 16] provided an algorithm using branch and bound method for Max Hamming Distance XSAT in $O(1.8348^n)$; Fu, Zhou and Yin [8] worked on a more specific problem, the Max Hamming Distance X3SAT, and found for this problem an algorithm with runtime $O(1.6760^n)$. In this paper, we propose an exact exponential algorithm to solve the Max Hamming Distance XSAT problem in $O(1.4983^n)$ time. Like all of them, we will use the branch and bound technique alongside a newly defined measure to improve the analysis of the algorithm.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases XSAT, Measure and Conquer, DPLL, Exponential Time Algorithms

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.15

Funding *Frank Stephan*: supported in part in part by the Singapore Ministry of Education Academic Research Fund Tier 2 grant MOE2016-T2-1-019 / R146-000-234-112.

Acknowledgements The authors would like to thank the anonymous referees of ISAAC 2019 for useful suggestions. Furthermore, the authors would like to thank internet companies for putting services like Wolfram Alpha Equation Solver, Firefox Scratchpad and Google Scholar for free onto the internet.

1 Introduction

The Satisfiability problem has been an important part of complexity theory and continues to be to this age. Given a Boolean formula φ in conjunctive normal form (CNF), can we find an assignment to the variables such that there are at least 1 literal in each clause that evaluates to “True”. There are many variants of the satisfiability problem to date and many are shown to be at least as hard as it. One variant that we will consider in this paper is the exact satisfiability problem (XSAT). Given a boolean formula φ in CNF, can we find a satisfying assignment such that exactly 1 literal in each clause is true while all the other literals are false. If we restrict the number of literals that can appear in any clause, then the problem comes as X k SAT, where k is the maximum number of literals that appear in any clause. Both XSAT and X3SAT are known to be **NP**-complete.



© Gordon Hoi and Frank Stephan;

licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 15; pp. 15:1–15:19

Leibniz International Proceedings in Informatics



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

There are no exact polynomial time solution for these problems unless $\mathbf{P} = \mathbf{NP}$. As such, one has to resort to designing exponential time algorithms to solve these problems in the exact manner. One common way of designing exact algorithms is to build DPLL [10, 9] style algorithms. The idea behind is to branch over a selected variable in order to decompose the problem into smaller subproblems and then solve them recursively. Every branching algorithm contains two kinds of rules: simplification and branching rules. Simplification rules are used to simplify the problem or to terminate the algorithm. Branching rules, on the other hand, are used to recursively break the problem down into smaller subproblems. For more on this topic, we refer the reader to chapter 2 of the textbook of Fomin and Kratsch [4].

The overall runtime to decide XSAT has been well-explored. Dahllöf, Jonsson and Beigel [17] gave an algorithm in $O(1.1907^n)$, improving the current state of the art from $O(1.2299^n)$; later Byskov, Maden and Skjærnaa [7] improved it to $O(1.1749^n)$. The algorithms mentioned here gives a solution to a particular XSAT instance. However, there are times when we want to consider many solutions and we want to see how “different” are these from one another. For example, consider the UNIQUE SAT problem [1], where there must be only 1 solution to this problem. On the other hand, for the usual satisfiability problem, the number of solutions for that formula can vary and the solutions can be very “different” from one another.

To make precise this notion of “different” solution, the definition of the hamming distance problem captures this combinatorial aspect naturally. The Max Hamming Distance problem asks, given a formula φ in CNF, what is the maximum number of variables that can differ between any two solutions to φ ? In this paper, we’ll study the Max Hamming Distance XSAT problem where all solutions must satisfy a given XSAT instance. In fact, one may note that a solution to the Max Hamming Distance XSAT problem can even yield a solution to solve the XSAT decision problem. It closely resembles a counting problem and since it is more complex than a XSAT decision problem, a larger overall runtime than $O(1.1749^n)$ may be expected.

Dahllöf [15] first gave an algorithm for Max Hamming Distance XSAT in $O^*(2^n)$ and an improved version in $O^*(1.8348^n)$, where the notation $O^*(.)$ denotes the suppression of the polynomial terms. The first algorithm enumerates all possible subset of all sizes while checking that they meet certain condition. The second algorithm uses techniques found in DPLL algorithms. Fu, Zhou and Yin [8] worked on the X3SAT problem instead and gave an algorithm to determine the Max Hamming Distance of two solutions of an X3SAT instance in time $O^*(1.6760^n)$.

In this paper, we will propose an algorithm to solve the Max Hamming Distance XSAT in $O(1.4983^n)$. Since X3SAT is a more specific version of an XSAT instance, our algorithm practically solves both problems in a shorter overall runtime. Like the above authors, our algorithm is also a DPLL algorithm which consists of branching and simplification rules. The novelty in this paper is the designing of branching cases used in combination with a nonstandard measure. If we were to use the standard measure, the algorithm takes $O(1.5761^n)$ ($\tau(7, 1)^2 = 1.5761$), while the use of a nonstandard measure brings the same bottleneck down to $O(1.4983^n)$. This on the other hand meant that we have to pay special attention to some cases. We’ll explain them in greater detail in the next section. In addition, our algorithm does a little more than just outputting the number to compute the Max Hamming Distance XSAT. It outputs a polynomial $p = \sum_k a_k u^k$ such that the coefficient a_k is the number of pairs of solutions with Hamming distance k of the given XSAT instance. In other words, we form for each pair of solutions the polynomial u^k where k is the Hamming distance of these solutions and then p is the sum over all the so obtained polynomials in the formal variable u .

2 Preliminaries

In order to analyse the time complexity of DPLL algorithms, one can consider search trees to help us illustrate the branching rules. One can consider the root of the search tree as the whole problem, and the successive child nodes as smaller instances of the problem when applying the branching algorithm. Using the search tree generated by the DPLL algorithm, if we can bound the number of leaves in the tree, then we will know the worst case runtime of this algorithm.

Kullmann [11] is one of the authors describing a technique to analyse DPLL algorithms; furthermore, Eppstein [2, 3] dealt with this technique in algorithms on graph colouring and more generally backtracking algorithms using the method of quasiconvex analysis of algorithms. The technique is to analyze each branching rule of an DPLL algorithm as follows: Let $T(n)$ denote the time needed for n variables, or more precisely the number of leaves of the search tree. Then the runtime of that branching rule is given by $T(n) = T(n - a_1) + T(n - a_2) + \dots + T(n - a_r)$, where r denote the number of branches (or child nodes), $r \geq 2$, generated from that node and each branch i removes a_i many variables. This can then be formulated as a linear recurrence $x^n = x^{n-a_1} + x^{n-a_2} + \dots + x^{n-a_r}$ and $\tau(a_1, a_2, \dots, a_r) = \min\{x \geq 1 : x^{-a_1} + x^{-a_2} + \dots + x^{-a_r} \leq 1\}$ is then the solution to the linear recurrence and $T(n) = \tau(a_1, a_2, \dots, a_r)^n$. This $\tau(a_1, a_2, \dots, a_r)$ is known as the branching factor. Note that $a_1 > a'_1$, then $\tau(a_1, a_2, \dots, a_r) < \tau(a'_1, a_2, \dots, a_r)$ and $\tau(i + \epsilon, j - \epsilon) < \tau(i, j)$, for all i, j, ϵ with $0 < i < j$ and $0 < \epsilon < \frac{j-i}{2}$.

A more sophisticated technique of analysing the runtime of branching algorithms is measure and conquer; a typical reference to this are by Fomin, Grandoni and Kratsch [5]. See also the textbooks of Downey and Fellows [13] and of Fomin and Kratsch [4]. It focuses on designing a new measure instead of changing the algorithm and a measure is a weight assigned to a variable in our case. Typically, a measure should observe 3 properties:

- A measure should be at least 0;
- The measure of an instance of a subproblem obtained after branching should be smaller than the measure of the instance before branching;
- The measure of an instance should be bounded above by some function on the parameter of the problem.

In simple measure, every variable is given a weight of 1 and eliminated variables are given a weight 0. When measure and conquer is used, usually one designs a new nonstandard measure in the hope of bringing down the analysis of algorithm further without changing the algorithm. One may then again apply Kullmann's technique to solve the linear recurrence under the new measure to obtain the running time of the branching rule.

3 High-level description of algorithm

We give a high level description of the algorithm here before giving the whole algorithm. To know the Max Hamming Distance of a given XSAT instance, we consider each pair of solution and the number of variables that each pair differs. Therefore, to achieve this, we will take in two identical instances of φ , we call it φ_1 and φ_2 such that $\varphi_1 = \varphi_2$ and branch them individually. Through the course of branching, φ_1 and φ_2 will differ later on.

The idea here is to branch whenever cases arise that match our branching cases. Our aim is to break connected components of clauses together into smaller isolated sets of clauses; this is done by first breaking chains of length 6 or more and second by branching variables that have 8 or more neighbours. Once they are small enough, we can brute force the Max Hamming Distance from the remaining sets of small isolated cliques. Note that the Hamming

15:4 Measure and Conquer for Max Hamming Distance XSAT

Distance can only be computed once the same variable on both φ_1 and φ_2 have been branched. Suppose that the branched i -th variable $x_{1,i}$ and $x_{2,i}$ have the same value, then they will have Hamming distance 0 and represent this value as $u^0 = 1$. Otherwise, they have different values and they will have Hamming Distance 1 and we represent this as $u^1 = u$. As we build the search tree, and we traverse a path from the root to the leaf, we multiply all the polynomials along each edge of the path. Finally, we form the sum all the multiplied polynomials along each path up.

We use polynomials to represent our Hamming Distance. The degree k of the polynomial denotes the Hamming Distance k and the coefficients of degree k denote the number of pairs of solution having Hamming Distance k . The notion of polynomials allow us to add and multiply together which makes it easier to understand. The reader might also consult Example 11 below for a better understanding of the algorithm.

4 A new measure

Here, we define the measure that we will be using throughout the entire algorithm. Given any variable x_i , we give the following formulas for weight w_i of variable x_i and measure μ as

$$w_i = \begin{cases} 1, & \text{if } x_i \text{ has at least 3 neighbours;} \\ 0.905813, & \text{otherwise;} \end{cases} \quad \mu = \sum_{i=1, \dots, n} w_i \leq n.$$

Here a neighbour is defined as a variable that appears alongside with it in the same clause, see Definition 1 below. So most cases of weight 0.905813 are variables occurring in exactly one clause and this has 3 literals. So if the formula consists of clauses (a, b, c, d) , (c, d, e, f) , (f, g, h) the weight of a, b, c, d, e, f is 1 each and the weight of g, h is 0.905813 each; for example, a has the neighbours b, c, d , c has the neighbours a, b, d, e, f and g has the neighbours f, h .

The idea of choosing the reduced weight for variables with two or less neighbours is that creating such variables, by branching a node in a 4-clause, should give some savings in anticipation of the further savings done by handling 3-clauses. Furthermore, the value 0.905813 is chosen by a computer program optimising the runtime and mainly satisfies that $3^{1/(3 \cdot 0.905813)}$ and $\tau(1 + 4 \cdot 0.094187, 7)^2$ are both bounded by 1.4983, which are the estimate of the basis of the exponentiation in Case 2 and the bottleneck of Case 1, respectively. Here note that $0.094187 = 1 - 0.905813$.

5 The algorithm in detail

We give a few definition that will be needed throughout the algorithm.

► **Definition 1.** *A variable x occurs in a clause iff the clause contains at least one of the literals x and $\neg x$. Two variables are neighbours iff they are different and occur jointly in at least one clause; two clauses are neighbours iff there is at least one variable occurring in both of them.*

For example, if there are clauses (a, b, c) , (c, d, e) , (e, f, g) then the neighbours of the variable c are a, b, d, e ; the clause (c, d, e) is a neighbour of the clause (a, b, c) , but (e, f, g) is not a neighbour of (a, b, c) .

► **Definition 2.** *We say that a clause c is isolated if for every variable x_i in the clause c does not appear in any other clauses. We say that a variable is a singleton if it appears, as negated or an unnegated literal, in exactly one clause.*

► **Definition 3.** We say that two variables x and y are linked together if there are up to 3 clauses which allow together to derive that either $x = y$ or $x = \neg y$. If two variables are linked, we can remove one variable, say y , by replacing all occurrences of y by respectively x or $\neg x$. We write $x \sim y$. If there are two clauses overlapping with each other, we call the non-overlapping variables that appear in the two clauses as outside variables.

► **Definition 4.** A chain is a sequence C_1, C_2, \dots, C_k of clauses such that C_i, C_j share at least one variable iff $|i - j| \leq 1$ and k is the length of the chain.

For example, the chain $(\alpha, b), (b, d, a), (a, e, c), (c, \beta)$ in Figure 12 is a chain of at least length 4 and $(a, b, c), (c, d, e)$ is a chain of length 2. Figure 5 is not a chain, as all three members have the joint variable a .

Algorithm MHXSAT (Max Hamming Distance XSAT).

- **Input:** φ_1 and φ_2 , where $\varphi_1 = \varphi_2$ and both are instances of XSAT; A set of polynomials $p_{i,b_i,j,b_j}(u)$ for each possible pair (x_i, b_i, y_j, b_j) of variables x_i in φ_1 and y_j in φ_2 including specific variables x_0, y_0 which only occur with value $b_0 = 0$ and $c_0 = 0$.
- **Initial Recursive Call:** Call algorithm with φ_1, φ_2 both being the initial formula and the polynomials p_{i,b_i,j,b_j} be defined for all i, j, b, b' such that (i is index of x_i and $b \in \{0, 1\}$ or $i = 0 \wedge b = 0$) and (j is index of y_j and $b' \in \{0, 1\}$ or $j = 0 \wedge b' = 0$); the initial value p_{i,b_i,j,b_j} is as follows: if $i = j$ and $b \neq b'$ then $p_{i,b_i,j,b_j}(u) = u$ else $p_{i,b_i,j,b_j}(u) = 1$.
- **Output:** a formal polynomial $p(u) = \sum_k a_k u^k$ where for each k are a_k pairs of satisfying assignments where the two assignments have Hamming distance k .
- **Label Start.**
- **Simplification for φ_1 .**
 - **If** there is a clause α where, whatever values one chooses for the variables, the sum of the literals is not 1 **Then Return** with the polynomial $p(u) = 0$.
 - **If** there are clauses α, β with a literal using x_i occurring at least in α and a value b_i which is the only value from 0, 1 for x_i such that α, β can be made true **Then Begin** replace x_i by the constant b_i everywhere, update $p_{0,0,j,b_j} = p_{0,0,j,b_j} \cdot p_{i,b_i,j,b_j}$ and remove x_i from the set of possible variables and i from the possible indices of polynomials on the φ_1 -side and **Goto Start; End.**
 - **If** there is a clause α mentioning exactly two variables x_i, x_j in the literals such that one can deduce either $x_i = x_j$ or $x_i = \neg x_j$ for all possible assignments making α true and there is at least one clause which contains exactly one of x_i, x_j and also a further variable in some literal **Then Begin** replace x_i everywhere by x_j or $\neg x_j$, respectively, and update $p_{j,b_j,k,b_k} = p_{j,b_j,k,b_k} \cdot p_{i,b_i,k,b_k}$ or $p_{j,b_j,k,b_k} = p_{j,b_j,k,b_k} \cdot p_{i,1-b_i,k,b_k}$ in the respective case for all $b, b' \in \{0, 1\}$ and indices k on the φ_2 -side and remove x_i, i from the list of possible variables and indices of φ_1 /* this is called linking */ **Goto Start; End.** /* Note that the only case where one cannot link the variables is the one where one variable occurs in two literals in the clause, say x_i and $\neg x_i$ or two times x_i ; this case is already caught by the previous case, as either $x_i, \neg x_i$ both occur and the literal containing x_j must be 0 or x_i occurs twice and x_i must be 0. */
 - **If** there are clauses of form $x_i \vee \alpha$ and $\neg x_i \vee \beta$ in φ_1 and x_i does not have 10 or more neighbours of weight 0.905813 **Then Begin** update for all x_k occurring in α with $b_k = 1$ if x_k is the literal in α and $b_k = 0$ if $\neg x_i$ is the literal in α the polynomials as $p_{k,b_k,j,b_j} = p_{k,b_k,j,b_j} \cdot p_{i,0,j,b_j}$ and for all x_k occurring in β with $b_k = 1$ if x_k is the literal in β and with $b_k = 0$ if $\neg x_i$ is the literal in β the polynomials as $p_{k,b_k,j,b_j} = p_{k,b_k,j,b_j} \cdot p_{i,1,j,b_j}$ for all variables y_j on the φ_2 -side and all $b' = 0, 1$ and

15:6 Measure and Conquer for Max Hamming Distance XSAT

replace all clauses of the form $x_i \vee \gamma$ by $\beta \vee \gamma$ and all clauses of the form $\neg x_i \vee \delta$ by $\alpha \vee \delta$ and remove x_i, i from the list of possible variables and indices of φ_1 /* this is called making a cut or also called resolving x_i */ **Goto Start; End.**

- **Simplification for φ_2 .** Do the actions analogue to those for φ_1 given above and go back to **Start** whenever any of these actions has been performed.
- **Branching for φ_1 .**
 - **If** there is a clause of form $\alpha \vee \beta$ of φ_1 such that branching $\alpha = 1$ versus $\alpha = 0$ has branching factor at most 1.4983 as indicated in Proposition 5 or List 1 (= Proposition 6) or List 2 (= Proposition 7), in this order of priority, and the subclauses α, β both contain at least one literal then compute $p = \text{MHXSAT}(\varphi_1 \wedge \alpha, \varphi_2, polys) + \text{MHXSAT}(\varphi_1 \wedge \beta, \varphi_2, polys)$ and go out with **Return(p) End.** /* This is called branching $\alpha = 1$ versus $\alpha = 0$; for doing the branching, the only requirement is that there is a clause in φ_1 of the form $\alpha \vee \beta$ where both α, β have at least one literal. After doing all choices by Proposition 5, one can w.l.o.g. assume by cut, branching and renaming that all clauses in φ_1 contain only positive literals; after doing all choices of List 1, there are except for isolated components of size up to six, no clauses in φ_1 with a multiple overlap; after doing all choices of List 2, there are no proper chains of length 6 or more in φ_1 and no variables with more than 7 neighbours. */
- **Branching for φ_2 .** Do the actions analogue to those for φ_1 given above.
- **Now none of the above cases applies.** Note that no variable has more than 7 neighbours (unless it is in an isolated component of size 9 by List 2 item 3) and no proper chain is longer than 5 clauses; thus both formulas are split into isolated components which by Proposition 8 contain at most 1364 variables and by a more involved argument with a slightly modified algorithm in Proposition 14 in the appendix at most 67 variables.
- **Measure φ_1 and φ_2 and do the following with the smaller task, say with φ_1 .**
 - **Compute** an explicit list of satisfying assignments for φ_1 only using the surviving variables x_i obtaining a list of vectors with entries b_i for each $i \neq 0$;
 - **Compute** for each assignment (b_i) an updated list of polynomials pol_s obtained by updating the previous polynomials to $p_{0,0,j,b'} = p_{0,0,j,b'} \cdot \prod_{i \neq 0} p_{i,b_i,j,b'}$;
 - **For each (b_i), pol_s Do Begin**
 - * **For each component** of variables and clauses C in φ_2 compute all possible assignments (b_j) of the $y_j \in C$ and update

$$p_{0,0,0,0} = p_{0,0,0,0} \cdot \left(\sum_{\text{assignment } (b_j) \text{ of } C} \prod_{x_j \text{ occurring in } C} p_{0,0,j,b_j} \right) \cdot \text{End}$$
- **Let p be the sum of all the $p_{0,0,0,0}$ calculated for the above list of (b_i), pol_s and **Return(p).****

6 The Simplifications and the Branchings from List 1 and List 2

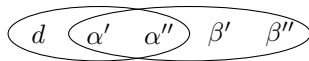
► **Proposition 5.** *If $a, \neg a$ appear both in clauses then either one can do a cut without increasing the weight or one can branch a with branching-factor below 1.31719.*

Proof. For the simplifications, as they remove variables, one should not expect problems. However, there is one case, namely making the cut. In the case of a cut eliminating variable d , one replaces clauses $d \vee \alpha$ and $\neg d \vee \beta$ by $\alpha \vee \beta$ and this increases the number of neighbours of the variables in the disjunctions α and β and might therefore increase the weights of some variables from 0.905813 to 1. If there are at most 9 such variables, the saving of removing d is at least 0.905813 while the weights going up are at most $9 \cdot 0.094187 < 0.905813$ and

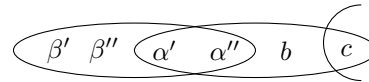
there is no problem; here note that $1 - 0.905813 = 0.094187$. If there are at least 10 such variables, one can see that for both cases $d = 0$ and $d = 1$, some additional literals are set to 0 and therefore variables are eliminated, thus one can branch d instead of making a cut and the worst case is that the eliminated variables are distributed 2 on one side and all others on the other side, hence one has $\tau(1 + 2 \cdot 0.905813, 1 + 8 \cdot 0.905813)^2 \leq 1.31719$. For that reason, one can assume when reaching the branching case, that all variables are non-negated – if a variable d is negated everywhere, one replace $\neg d$ by d everywhere and adjusts the polynomials accordingly. ◀

Now, we begin to analyse the time needed for each rule in the algorithm. We will omit the simplification rules and instead only analyze the branching rules since the simplification rules do not increase the number of leaves of the search tree. Before we go on to analyze the runtime complexity, we give an example of the different branching cases below in List 1 and List 2. List 1 contains all the different cases that we handle when there are two or more variables that appear between two clauses. List 2 contains all the different branching cases where there is only a common variable between two clauses.

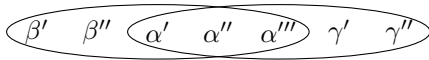
List 1.



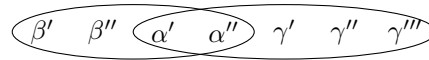
■ **Figure 1** List 1 Item 1.



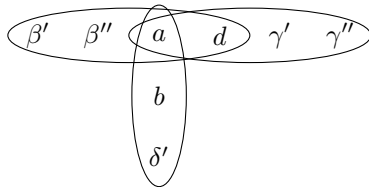
■ **Figure 2** List 1 Item 2.



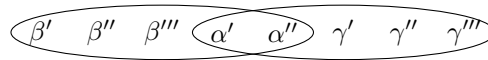
■ **Figure 3** List 1 Item 4 with 7 vars.



■ **Figure 4** List 1 Item 4 with 7 vars.



■ **Figure 5** List 1 Item 3.



■ **Figure 6** List 1 Item 4 with 8 vars.

This list gives how to handle overlaps between of two or more variables between two clauses. Furthermore, the variables are in each clause listed nonredundantly, so if a clause is (α, β) , the two lists of variables α and β are disjoint. Lower case single letters always refer to single variables. Note that the case that an isolated component with up to six variables is not covered, where isolated means that no other clause has some but not all variables from this component. Such components are treated in Case 2. In the figures above, primed Greek letters like α', α'' denote the variables in the corresponding list of variables like α in this case. Furthermore, we indicate only those clauses which are relevant; additional clauses might connect to those given in the figures or case distinction, unless explicitly said otherwise.

1. $(d, \alpha), (\alpha, \beta)$ where α, β are lists of literals not containing d and α has at least two variables. Now one can see that $d = \beta$ and by adding $\neg d$ on both sides, one gets $1 = \neg d + \beta$. Thus one has the formulas $(d, \alpha), (\neg d, \beta)$ and these allow a cut, as in the last item of the

Simplification part of the algorithm. In the case that making the cut would cause more than 10 variables to be upgraded from weight 0.905813 to 1 and therefore the measure would go up, one does not make the cut but instead branches the variable d . In the case of a branching, note that there is at least one variable in β and furthermore d has weight 1; furthermore, the literals with weight 0.905813 are except a perhaps single one in β all connected to d , as $\neg d$ was just introduced and the variables in α have weight 1. This gives the branching factor $\tau(1 + 0.905813, 3 + 10 \cdot 0.905813)^2 \leq 1.2900$. For example in Figure 1, $|\alpha| = 2$, there are 3 outside variables.

2. $(\beta, \alpha), (\alpha, b, c), (c, \delta)$ with δ containing some variable d not in (α, b, c) and α, β have both at least two variables.

If $\delta = b \vee d$ then the situation of Item 1 applies after some renaming, as the clauses $(d, c, b), (c, b, \alpha)$ exist. Thus this case is already handled.

If δ contains at least three variables including b then d has weight 1 and will in the case $\alpha = 0$ be set to 0 again and the branching factor is at most $\tau(3, 4)^2 \leq 1.4903$.

If δ does not contain b then the case $\alpha = 0$ allows to link b to c and removes the variables in α and b which all have weight 1 and $\alpha = 0$ allows to remove those in β, b, c which all have weight 1 and so the branching factor is again at most $\tau(3, 4)^2 \leq 1.4903$.

For example in Figure 2, we have two 4-clauses with overlapping part α consisting of two variables.

3. $(\beta, a, d), (a, d, \gamma), (a, b, \delta)$ or $(\beta, a, d), (a, d, \gamma), (a, b, d, \delta)$ and Items 1,2 do not apply and $|\beta| = |\gamma| = 2$. Now every subclause containing variables of β is a subclause of (β, a, d) and every subclause containing variables of γ is a subclause of (γ, a, d) and b does not occur in a, d, β, γ .

One branches $a + d = 0$ versus $a + d = 1$. If $a + d = 0$ then the weight of all the variables in β, γ is reduced from 1 to 0.905813. If $a + d = 1$ all variables in β, γ are set to 0 and either a can be linked to d or b can be set to 0, depending on what the third clause is. So the branching factor is at most $\tau(2 + 4 \cdot 0.094187, 4 + 0.905813)^2 \leq 1.4888$.

For example in Figure 5, we have the variables a, d as the overlapping variables and we have $|\delta| = 1$.

4. Not Items 1,2,3 and $|\alpha| + |\beta| + |\gamma| \geq 7$. This in particular means that α, β, γ have at least two variables. As Item 2 does not apply, if β has two exactly two variables then all clauses containing variables from β are subclauses of (α, β) , similarly for γ . One branches $\alpha = 0$ versus $\alpha = 1$.

Now consider the case where α has two variables and β, γ have together exactly five variables, so one of them, say β , has exactly two variables. Now consider the subcase that one or both variables from α are in a further clause containing a b not occurring in α, β, γ . Now $\alpha = 1$ makes the variables in β, γ all 0 and further allows to either link the variables in α or makes $b = 0$; furthermore, $\alpha = 0$ makes the variable in β have a weight 0.905813 while before they had weight 1. Thus the branching factor is at most $\tau(2 + 2 \cdot 0.094187, 5 + 0.905813)^2 \leq 1.4498$. In the other subcase that α has exactly two variables and they do not occur in clauses with variables outside α, β, γ , one takes into account that choosing $\alpha = 1$ reduces the weight of the variables of α from 1 to 0.905813. So the branching factor is at most $\tau(2 + 2 \cdot 0.094187, 5 + 2 \cdot 0.094187)^2 \leq 1.4917$.

If $|\alpha| = 2$ and $|\beta| + |\gamma| \geq 6$ then the branching factor is at most $\tau(2, 6)^2 \leq 1.4656$ without any further assumptions. If $|\alpha| \geq 3$ then the branching factor is at most $\tau(3, 4)^2 \leq 1.4903$.

In Figure 3, we have that $|\alpha| = 3$ and we have 4 outside variables where $|\beta| + |\gamma| = 4$; Figures 4 and 6 represent typical cases where $|\alpha| = 2$ and $|\beta| + |\gamma| \geq 5$.

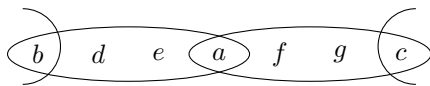
This case distinction shows that one can get rid of all multiple overlaps which are in components larger than six variables, if a component contains only $(\alpha, \beta), (\beta, \gamma)$ with α, β, γ having each two variables and perhaps further clauses only using these variables. This situation allows no simplification, but one can let it stand, as the component is already sufficiently small and deal with the other components in φ_1 and φ_2 until those are also broken down.

To see the completeness of the above case-distinction in List 1, note that Item 1 deals with the case that one of the neighbours of α is only a single variables d ; in all other cases it is assumed that the basic situation is $(\beta, \alpha), (\alpha, \gamma)$ with α, β and γ each having at least two variables. Item 2 considers the case that α has a neighbour $b \vee c$ where the variable c occurs in a further clause with at least one variable different from those in α, b, c . Item 3 considers the case where both neighbours of α have two variables and the condition from Item 2 does not apply to these neighbours, but that at least one variable a of α occurs in a clause where not all variables are from α, β, γ . Item 4 considers the case where $|\alpha| + |\beta| + |\gamma| \geq 7$ and the above cases do not apply.

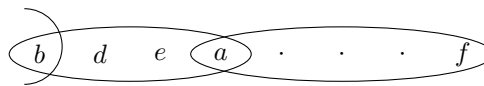
List 2. In this list, if a variable is branched, it is called a . Furthermore, b, c are variables in the chain which link the clauses considered to further members and these contain exactly one of b, c each. One does the first case in this list which applies. Again note that the case of one variable a in four clauses of size 3 without any further variables in the isolated component except for a and its eight neighbours does not need any further treatment. Thus in item 1 of the following list, one can assume that there is either one neighbour with weight 1 or at least 10 neighbours with weight 0.905813.



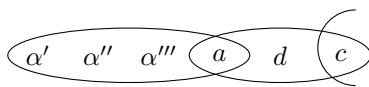
■ **Figure 7** List 2 Item 1.



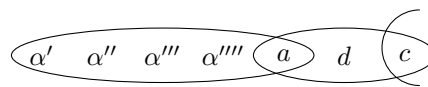
■ **Figure 8** List 2 Item 4.



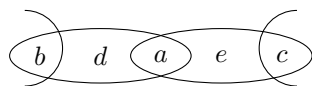
■ **Figure 9** List 2 Item 5.



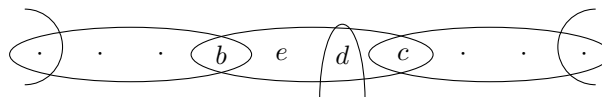
■ **Figure 10** List 2 Item 2.



■ **Figure 11** List 2 Item 2.



■ **Figure 12** List 2 Item 3.



■ **Figure 13** List 2 Item 6, subsequence of 6-chain.

1. One variable a with at least eight neighbours. One branches a . If there are eight neighbours and at least two of them have weight 1 or if there are at least nine neighbours then the branching factor is at most $\tau(1, 6 \cdot 0.905813 + 3)^2 \leq 1.4967$. If there are exactly

15:10 Measure and Conquer for Max Hamming Distance XSAT

eight neighbours and exactly one of them has weight 1 then the neighbours are in four 3-clauses and only one of these has a variable b shared with another clause, let d denote the other variable in this clause. If $a = 0$, b and d will be linked, if $a = 1$ all neighbours are 0. This gives $\tau(1 + 0.905813, 7 \cdot 0.905813 + 2)^2 \leq 1.3775$.

2. $(\alpha, a), (a, d, c)$ where α has at least three variables. One branches a . Note that a has at least five neighbours out of which only d can have measure 0.905813. This gives $\tau(1 + 0.905813, 5 + 0.905813)^2 \leq 1.4824$. For example in Figure 10, we have the overlapping variable as a , c appears as part of a larger chain and $|\alpha| = 3$ here. We have 5 outside variables in this case.
3. $(b, d, a), (a, e, c)$. One branches a . If $a = 0$ then one can link both d to b and e to c else all five variables are determined. Thus $\tau(1 + 2 \cdot 0.905813, 3 + 2 \cdot 0.905813)^2 \leq 1.4518$. This is exactly the case as given in Figure 12 where a is our overlapping variable and we have 4 outside variables in this case.
4. $(b, d, e, a), (a, f, g, c)$ with each of d, e, f, g being singleton variables and b, c being in further clauses. Now one branches a and gets $\tau(1 + 4 \cdot 0.094187, 7)^2 \leq 1.4983$, as d, e, f, g will change their measure by the branching in the case that $a = 0$. This is exactly the case as given in Figure 8 where a is the overlapping variable and we have 6 outside variables in this case.
5. $(b, d, e, a), (a, \dots, f)$ and d is a singleton variable and the clause (a, \dots, f) having at least 5 variables. Now one branches a and has $\tau(1 + 0.094187, 8)^2 \leq 1.4983$. We see this example in Figure 9 where a is our overlapping variable and we have 7 outside variables in this case.
6. (b, d, e, c) and b, c are only in inner clauses of some chain and d is in a further clause. By the cases with the 3-clauses above being done first, the clauses on the other side of b and c contain at least four literals. Thus b, c have both at least six neighbours. If one of them, say b , would have further neighbours which have weight 0.905813, these would have to be in a further 3 clause and so b would have eight neighbours, so that List 2 Item 1 above applies, thus this case does not happen. Now one distinguishes $b + c = 0$ versus $b + c = 1$. In the case that $b + c = 0$, one can link d, e and eliminate three variables. In the case that $b + c = 1$, d, e are 0 and one can make a cut exploiting that $b = \neg c$ and that neither b nor c have neighbours which have weight 0.905813. So no weight-compensation is needed and 4 variables are eliminated. This gives $\tau(3, 4)^2 \leq 1.4903$. Note that we needed the fact that the neighbours at b, c are connected to further clauses in the chain only to guarantee that these clauses have at least four variables; therefore we apply this branching rule also when a 4-clause with three neighbours satisfies that two of them are disjoint and have four variables each. An example of this can be found in Figure 13.

Assume now by way of contradiction that after all actions in List 2 are done there would be a chain of length 6. Note that no variable a in the chain can have exactly eight neighbours with all having weight 0.905813, as then every variable in the chain is either a or a neighbour of a and thus the chain has at most length 2. None of the second, third, fourth and fifth member of this chain can be 3-clauses, as these are eliminated as above. The third and fourth both cannot be two 4-clauses containing two singletons. Neither the third nor the fourth can be a 4-clause with at most one singleton variable, as such clauses are also eliminated. The third and the fourth clause cannot be a 4-clause plus a clause of five or more variables, as in such situation again the connecting variable is branched. The third plus the fourth clause cannot together have 9 or more variables, as then the connecting variable a has at least eight neighbours and can be branched. Thus there is no chain of length 6 or more.

7 Analysis of Algorithm

For the verification, we divide our algorithm into two portions: Case 1 on the different simplification and branching cases and Case 2 on the brute forcing of the independent cliques of clauses. We will be using our defined measure in this analysis as given in Section 4. In addition, note that we are branching on both formulas φ_1 and φ_2 . This means that we have $2n$ many variables in this case. All branchings in this proof are kept to the standard branching of a variable taking on values either 1 or 0 unless explicitly mentioned. We pay more attention to 3-literal and 4-literal clauses due to our nonstandard measure. In addition, we exploit the fact these clauses which we branch belong to a larger chain. We analyze the worst runtime needed for the first part below.

► **Proposition 6.** *Let $k \geq 2$. If there are k overlapping variables between two clauses, then the worst case time complexity for branching these overlapping variables in the two clauses is $O(1.4917^n)$.*

Proof. First we note that by the simplifications in the algorithm and Proposition 5, we can assume without loss of generality, that all literals are positive, that is, not negated. If only $\neg a$ occurs for some variable a , we just simply replace $\neg a$ everywhere by a and adjust the polynomials accordingly.

In the following we say that the outside variables in a pair of clauses $(\beta, \alpha), (\alpha, \gamma)$ are in $|\beta| - |\gamma|$ orientation. List 1, Item 1 tells us how to handle outside variables in $1-m$ orientation, where $m \geq 2$. Thus we only have to deal with $m-m'$ orientations where $m \geq 2$ and $m' \geq 2$.

When $k = 2$ and if there are 4 outside variables, then it must be in the 2-2 orientation (Figure 2). In this case, we exploit that these clauses are part of a larger chain. If not, then they are an isolated component and will be handled by Case 2 of our algorithm. Therefore, one of the variables in one of the orientation must be connected to a clause somewhere. Let α', α'' be our overlapping variables and let c be our variable that is connected to a different clause and b be a variable appearing in the clause $(\alpha' \vee \alpha'' \vee b \vee c)$. We branch $(\alpha' \vee \alpha'') = 1$ and $\alpha' = \alpha'' = 0$. Branching $(\alpha' \vee \alpha'') = 1$ will allow us to remove all 4 outside variables and branching $\alpha' = \alpha'' = 0$ will allow us to remove both α', α'' , and further link up the variables b and c . Therefore, we have at most $T(\mu) = T(\mu - 4) + T(\mu - 3) = O(1.2208^{2\mu}) = O(1.4903^n)$. On the other hand, suppose that we are not allowed to link b and c together, then it must be that c appears in another clause containing b as well. This other clause that c appears in must be at least 4-literal in length else List 1 Item 1 would have handled it for us. There must be a new variable d that is different from the overlapping variables and outside variables. Then branching $(\alpha' \vee \alpha'') = 1$ will allow us to remove all 4 outside variables and branching $\alpha' = \alpha'' = 0$ will allow us to remove both α' and α'' and at least another variable d . This gives us $T(\mu) = T(\mu - 4) + T(\mu - 3) = O(1.2208^{2\mu}) = O(1.4903^n)$. This completes the case for 4 outside variables.

If there are 5 outside variables as shown in Figure 4, then we must have them in the 2-3 orientation. Let β' and β'' be the two variables in the “2” orientation, while γ', γ'' and γ''' be the three variables in the “3” orientation. We consider 4 different cases here. The first case is that one of the β' or β'' is connected to a larger chain and one of α' or α'' or both variables are in a further clause containing a new variable b that is different from the outside variables and the overlapping variables. Then branching $(\alpha' \vee \alpha'') = 1$ will allow us to remove all 5 outside variables and link $\alpha' = \alpha''$ or remove b . Branching $\alpha' = \alpha'' = 0$ will allow us to remove 2 overlapping variables and link β' to β'' . This gives us $T(\mu) = T(\mu - 5.905813) + T(\mu - 3) = O(1.1757^{2\mu}) = O(1.3822^n)$. The second case is that one of the β' or β'' is connected to a larger chain and α' and α'' do not appear in further clauses. Then branching $(\alpha' \vee \alpha'') = 1$ will allow us to remove all 5 outside variables

and we can also factor in the change in measure for α' and α'' . Branching $\alpha' = \alpha'' = 0$ will allow us to remove both overlapping variables and link β' to β'' . This will give us $T(\mu) = T(\mu - 5 - 2 \cdot 0.094187) + T(\mu - 3) = O(1.1897^{2\mu}) = O(1.4154^n)$. The third case is that both β' and β'' are not connected to a larger chain and one of α' or α'' or both variables are in a further clause containing a variable b that is different from the outside variables and overlapping variables. Then branching $(\alpha' \vee \alpha'') = 1$ will allow us to remove all 5 outside variables and link up $\alpha' = \alpha''$ or remove b . Branching $\alpha' = \alpha'' = 0$ will allow us to remove both overlapping variables and also factor in the change of measure for β' and β'' . This gives us $T(\mu) = T(\mu - 5.905813) + T(\mu - 2 - 2 \cdot 0.094187) = O(1.2041^{2\mu}) = O(1.4498^n)$. Finally, the last case is that both β' and β'' are not connected to a larger chain and α' and α'' do not appear in a different clause. Then branching $(\alpha' \vee \alpha'') = 1$ will allow us to remove all 5 outside variables and allow us to factor in the change of measure for α' and α'' . On the other hand, branching $\alpha' = \alpha'' = 0$ will allow us to remove both overlapping variables and factor in the change in measure for both β' and β'' . This will give us $T(\mu) = T(\mu - 5 - 2 \cdot 0.094187) + T(\mu - 2 - 2 \cdot 0.094187) = O(1.2214^{2\mu}) = O(1.4917^n)$. This completes the case for 5 outside variables.

Now, for the number of outside variables j , with $j \geq 6$, regardless of the orientation of the outside variables, we have that applying the branching technique as above, we will arrive at $T(\mu) = T(\mu - 6) + T(\mu - 2) = O(1.2106^{2\mu}) = O(1.4657^n)$ regardless if they appear as part of a larger chain. Now we have that $\tau(j, 2)^2 \leq \tau(6, 2)^2 < 1.4657$. The case for $k = 2$ is therefore complete.

For $k = 3$, we consider the case that we have 4 outside variables as shown in Figure 3. Let our overlapping variables be $\alpha', \alpha'', \alpha'''$ and we will branch $(\alpha' \vee \alpha'' \vee \alpha''') = 1$ and $\alpha' = \alpha'' = \alpha''' = 0$ and this gives us $T(\mu) = T(\mu - 3) + T(\mu - 4) = O(1.2207^{2\mu}) = O(1.4903^n)$ regardless if they appear in a larger chain. Now for $j \geq 4$ outside variables, our branching factor must be at most $\tau(3, j)^2 \leq \tau(4, 3)^2 = 1.4903$. The case for $k = 3$ is therefore complete.

For $k > 3$ overlapping variables and for $j \geq 4$ outside variables, our branching factor must be bounded above by 1.4903. This can be seen from the fact that $\tau(k, j)^2 < \tau(4, 3)^2 = 1.4903$. Therefore, the case for all $k \geq 2$ has been covered and will take at most $O(1.4903^n)$ time. ◀

► **Proposition 7.** *The worst-case runtime complexity of the branching cases when there is exactly 1 overlapping variable between two clauses is $O(1.4983^n)$.*

Proof. Let j be the number of outside variables that the overlapping variable has. Note that $j \geq 4$. For $j = 4$, we have that it must be in a 2-2 orientation as shown in Figure 12. Now, we branch the overlapped variable and this gives us $T(\mu) = T(\mu - 3 - 2 \cdot 0.905813) + T(\mu - 1 - 2 \cdot 0.905813)$. Therefore, we have $T(\mu) = O(1.2049^{2\mu}) = O(1.4518^n)$. Note that this is the worst case when we have exactly two singletons in two of the outside variable.

If $j = 5$, then we can only have them in the 2-3 orientation like in Figure 10. Therefore, we branch the overlapping variable a and we have $T(\mu) = T(\mu - 5 - 0.905813) + T(\mu - 1 - 0.905813 + 2 \cdot (0.905813 - 1)) = O(1.2083^{2\mu}) = O(1.4600^n)$. In this case, we assume that there are 3 singletons out of the 5 outside variable. If we have 2 singletons out of the 5 variables spread out in a 1-1 fashion, then we have $T(\mu) = T(\mu - 5 - 0.905813) + T(\mu - 1 - 0.905813 - (1 - 0.905813)) = O(1.2128^{2\mu}) = O(1.4709^n)$. On the other hand, if both singletons are now on the 4-literal clause, then we have $T(\mu) = T(\mu - 6) + T(\mu - 2) = O(1.2107^{2\mu}) = O(1.4658^n)$. If we have 1 singleton out of the 5 variable, then we consider the fact that this singleton can be at the 4-literal or the 3-literal clause. If it appears on the 3-literal clause, then we have $T(\mu) = T(\mu - 5 - 0.905813) + T(\mu - 1 - 0.905813) = O(1.2176^{2\mu}) = O(1.4826^n)$.

If it appears on the 4-literal clause, then we have $T(\mu) = T(\mu - 6) + T(\mu - 2 - (1 - 0.905813)) = O(1.2062^{2\mu}) = O(1.4550^n)$. Finally, if there are no singletons, then we have $T(\mu) = T(\mu - 6) + T(\mu - 2) = O(1.2106^{2\mu}) = O(1.4656^n)$. This completes the case for 5 outside variables.

If $j = 6$, then we can have it either in the 2-4 orientation or the 3-3 orientation, as shown in Figure 11 and 8 respectively. If it is the 2-4 orientation, then we branch the overlapping variable and we have $T(\mu) = T(\mu - 6 - 0.905813) + T(\mu - 1 - 0.905813) = O(1.19655^{2\mu}) = O(1.4318^n)$. For the 5-literal clause, we do not need to consider the case if there are singleton variables or not as the presence or absence of it will not change the measure. Therefore, we will only need to consider the case where there are no singletons on the 3-literal clause. We have $T(\mu) = T(\mu - 7) + T(\mu - 2) = O(1.1908^{2\mu}) = O(1.4181^n)$. If the outside variables appear in the 3-3 orientation, then again, we branch the overlapping variable. We will therefore have $T(\mu) = T(\mu - 7) + T(\mu - 1 - 4 \cdot (1 - 0.905813)) = O(1.22403^{2\mu}) = O(1.22403^{2n}) = O(1.4983^n)$. In this case, we assume that all 4 out of the 6 outside variables are singletons. We also need to handle the case that there are less than 4 singletons in this case. Suppose that there exist at least one of the variables such that it is not a singleton, as shown in Figure 13, then we have to change our approach in branching this problem because of the change in measure. Instead of looking at the overlapping variable, we instead look at the 4-literal clause containing the non-singleton variable. Let the variables b and c be connected to a larger chain, d be that non-singleton variable and the last variable be e . Now both b and c cannot be connected to 3-literal clauses as the earlier cases would already have handled it. In addition, if b and c appear at least 3 times, else we can branch them immediately to get a branching factor of at most $T(\mu) = T(\mu - 7 - 2 \cdot 0.905813) + T(\mu - 1) = O(1.2165^{2\mu}) = O(1.4798^n)$. Therefore both b and c must appear exactly twice. Now we branch $(b \vee c) = 1$ and $b = c = 0$. If $b = c = 0$, then we can eliminate 3 variables by linking up d and e and if $(b \vee c) = 1$, we can remove all 4 variables. Therefore, $T(\mu) = T(\mu - 3) + T(\mu - 4) = O(1.2207^{2\mu}) = O(1.4903^n)$. This completes the case for $j = 6$ outside variables.

If $j = 7$, then we can have it either in the 3-4 orientation (Figure 9) or the 2-5 orientation. Now suppose that we have the 3-4 orientation. Then we will have $T(\mu) = T(\mu - 8) + T(\mu - 1 - 2 \cdot (1 - 0.905813)) = O(1.2169^{2\mu}) = O(1.4809^n)$. In this case, we assume that 2 of the 7 outside variables in the 4-literal clause are singletons.

If this case does not happen, then again we have to look at the other neighbours of the 4-literal. Now, the neighbours of this 4-literal clause cannot be 3-literal or 4-literal, as they would have been handled by the earlier cases. Therefore, the neighbours of this 4-literal clause must be a 5-literal clause.

We first choose any two variables b and c that are non-singletons. If variables b and c appear at least 3 times, then we'll branch them immediately to get a branching factor of at most $T(\mu) = T(\mu - 8 - 2 \cdot 0.905813) + T(\mu - 1) = O(1.2003^{2\mu}) = O(1.4406^n)$. If not, then these variables appear exactly twice and we branch them as $(b \vee c) = 1$ or $b = c = 0$. Then we will have a branching factor of $T(\mu) = T(\mu - 3) + T(\mu - 4) = O(1.2208^{2\mu}) = O(1.4904^n)$. If we have it in the 2-5 orientation, then we have $T(\mu) = T(\mu - 7.905813) + T(\mu - 1.905813) = O(1.1799^{2\mu}) = O(1.3922^n)$. Now, if there are no singletons on the 3-literal clause, then we will have $T(\mu) = T(\mu - 8) + T(\mu - 2) = O(1.1750^{2\mu}) = O(1.3807^n)$. This completes the case for 7 outside variables.

If $j = 8$, for example in Figure 7, branching the overlapping variable will give us a branching factor $\tau(j, i)^2 \leq \tau(9, 1)^2 < 1.4718$ regardless of the orientation of the outside variables. In addition for $j > 9$, we have that $\tau(j, 1)^2 < \tau(9, 1)^2 < 1.4718$. Note that this case applies to a 9-literal clause. We can just branch any variable appearing in a 9-literal clause to

15:14 Measure and Conquer for Max Hamming Distance XSAT

have a branching factor of $\tau(9, 1)^2 = 1.4718$. If there are 8 neighbours with two of them having at least weight 1, then we have a branching factor of at most $\tau(1, 6 \cdot 0.905813 + 3)^2 = 1.4967$. If there are exactly 8 neighbours and one of them having weight 1, then we have a case of an overlapping variable appearing in four 3-literal clauses. Branching the common variable will give us a branching factor of at most $\tau(1 + 0.905813, 7 \cdot 0.90513 + 2)^2 = 1.3775$. This completes the case for all overlapping variables of exactly 1 variable.

The upper bound of these branching rules is the branching rules that has the worst case time bound. Since the branching rule of the common variable between two 4-literal clause has the worst case timebound, therefore the worst case time bound is $O(1.4983^n)$. ◀

After all the above branching rules have been applied and we come to a point where no branching rules can be further applied, we come to the second part of the algorithm where both φ_1 and φ_2 consists of small isolated components. See also Figure 14 in the appendix. The following gives a rough estimate of the size of the isolated components; a better one is found in the appendix.

► **Proposition 8.** *After all actions in Case 1 is done, φ_1 and φ_2 consist of disjoint components each having at most 1364 variables.*

Proof. Note that by doing the actions in List 1, every two clauses in the component intersect by at most one variable, unless the component exist of exactly two clauses of the form $(a, b, c, d), (c, d, e, f)$. By doing the actions in List 2, every chain has at most length 5 and every chain of four members has in the interior a clause of size 4 and perhaps also a clause of size 5. So one chooses a clause C_1 of size 4. Now when starting from C_1 , one gives an upper bound on all nodes which are in the last clause of chains of form C_1 or C_1, C_2 or C_1, C_2, C_3 or C_1, C_2, C_3, C_4 or C_1, C_2, C_3, C_4, C_5 . Here one uses that each variable in an inner C_k has at least four and at most five members and thus each $a \in C_k$ has at most four new neighbours which are not covered by chains of the form C_1, C_2, \dots, C_k . So one gets the overall number of variables estimated by $4 \cdot (1 + 4 + 16 + 64 + 256) = 5 \cdot 341 = 1364$. The appendix gives an improved bound of 80 with a much more involved argument which also needs a slight generalisation of the cases in List 2. ◀

Let μ be the measure for φ_1 and ν for φ_2 at this point. Now we will apply branch and bound to the formula φ_k , where $k \in \{1, 2\}$ with $\min\{\mu, \nu\}$ and then brute force the Hamming Distance from the other formula. For the brute forcing portion, we know that the size of each component is bounded above by some constant c . Therefore, the only thing we need to ensure is that the branch and bound of the formula measure is still well within the time bound of $O(1.4983^n)$.

► **Proposition 9.** *The branch and bound of the formula with lower measure has worst case run time of $O(1.4983^n)$.*

Proof. Note that when we are in this case, we no longer have $2n$ variables but only n variables from one of the formula. In addition, we can safely assume that we will not have a variable with 8 other neighbours as it will be handled by the branching case above. We consider all possible cases here.

First we consider standalone clauses with length < 9 . Suppose that we have a clause of length 2. Then we branch the entire clause by the values $(1, 0)$ and $(0, 1)$. This will incur $T(\mu) = T(\mu - 2 \cdot 0.905813) + T(\mu - 2 \cdot 0.905813) = O(1.4662^n)$. Suppose that we have a length of length 3, then again we branch the clause with values $(1, 0, 0)$, $(0, 1, 0)$ and $(0, 0, 1)$. This

will incur $T(\mu) = 3T(\mu - 3 \cdot 0.905813) = O(1.4983^n)$. Now let $n \in \mathbb{N}$. We know that $n^{\frac{1}{n}}$ is decreasing for $n \geq 3$ and hence for any length of any standalone clauses with $3 < j < 9$, we have that the branching factor with j branches and removal of j variables for each branch, $\tau(j, j, \dots, j) < \tau(3 \cdot 0.905813, 3 \cdot 0.905813, 3 \cdot 0.905813) < 1.4983$. This completes the case for all standalone clauses.

We next consider clauses with at exactly 1 overlapping variable with < 8 neighbours. Now the worst case that we can have is an overlapping variable with 4 outside variables in a 2-2 orientation. In this case, we will have $T(\mu) = T(\mu - 1 - 4 \cdot 0.905813) + T(\mu - 1) = O(1.3433^n)$. Again, let j be the number of neighbours in this case with $4 < j < 8$. Then $\tau(1 + j \cdot 0.905813, 1) < 1.3433$. This completes the case for all clauses with exactly 1 overlapping variable. For two or more overlapping variables, we can treat it as a similar case by just branching only 1 of the overlapping variable. This completes the case for all clauses with overlapping variables. \blacktriangleleft

► **Theorem 10.** *The algorithm takes $O(1.4983^n)$ time.*

Proof. To know the worst case runtime of our algorithm, we have to consider the branching rule which generates the most number of leaves. For branching rules in Case 1, we have that the runtime is bounded above by $O(1.4983^n)$ as given by Proposition 7 and 6. For Case 2, as shown in Proposition 9, the runtime is again bounded above by $O(1.4983^n)$. So the overall complexity is $O(1.4983^n)$. \blacktriangleleft

► **Example 11.** *Consider φ_1, φ_2 to contain the following clauses : $x_1 \vee x_2 \vee x_3, x_1 \vee x_4 \vee x_5, x_1 \vee x_6 \vee x_7, x_2 \vee x_8 \vee x_9 \vee x_{10}$. For variable x_k , the initial values of the polynomials are $p_{k,a,k,b} = u$ in the case that $a \neq b$ and 1 in the case that $a = b$; furthermore, all polynomials involving mixed variables are 1 and also the polynomials with $k = 0$ on either side are 1.*

Now the algorithm branches x_1 , first in φ_1 . Now let x_1 take the value a_1 in φ_1 . If $a_1 = 1$ then the variables x_h with $h = 2, 3, 4, 5, 6, 7$ take the value $a_h = 0$. Furthermore, we update the polynomials as follows: $p_{0,0,k,b} = \prod_{h=1,\dots,7} p_{h,a_h,k,b}$ for all k, b and after that we let $p_{h,a,k,b} = 0$ for $h = 1, 2, \dots, 7$ and all a, b, k . The remaining formula in φ_1 is $x_8 \vee x_9 \vee x_{10}$. If $a_1 = 0$ then one can conclude that $x_2 = \neg x_3, x_4 = \neg x_5$ and $x_6 = \neg x_7$. However, only the first of these 3 possible equalities will be realised, as x_2 appears also in a further clause. So we do the update $x_3 = \neg x_2$ in φ_1 and $p_{3,a,k,b} = p_{3,a,k,b} \cdot p_{2,1-a,k,b}$ for all a, b, k and after that $p_{2,a,k,b} = 0$ for all a, b, k . The remaining formulas for φ_1 are in this case $x_4 \vee x_5, x_6 \vee x_7, x_2 \vee x_8 \vee x_9 \vee x_{10}$.

After that, one does the analogous updates in φ_2 .

The descent will result in 4 subcases where one has on each side either 3 variables and one clause or 8 variables distributed over 3 disjoint and unconnected clauses.

Now consider the example case where φ_1 has one clause (so one has branched $x_1 = 1$ previously for φ_1) and φ_2 has 3 clauses. Now one considers the 3 cases of (x_8, x_9, x_{10}) taking the values $(0, 0, 1), (0, 1, 0)$ and $(1, 0, 0)$ in φ_1 and one subbranches into these cases which will set the remaining variables accordingly. Then only the polynomials $p_{0,0,k,b}$ are non-zero and we will process the 3 components of variables $(x_2, x_8, x_9, x_{10}), (x_4, x_5)$ and (x_6, x_7) for φ_2 accordingly.

So for each $(b, b', b'') \in \{(0, 0, 1), (0, 1, 0), (1, 0, 0)\}$, we do separate computations where each of them starts with the same version of the polynomials and first updates for all applicable k and all $a \in \{0, 1\}$ the polynomials $p_{0,0,k,a} = p_{0,0,k,a} \cdot p_{8,b,k,a} \cdot p_{9,b',k,a} \cdot p_{10,b'',k,a}$ and once, this is done, one updates the polynomial $p_{0,0,0,0}$ 3 times as follows: First for 4 possible vectors in $U = \{(0, 0, 0, 1), (0, 0, 1, 0), (0, 1, 0, 0), (1, 0, 0, 0)\}$ of (x_2, x_8, x_9, x_{10}) , we update $p_{0,0,0,0} = p_{0,0,0,0} \cdot (\sum_{(c,c',c'',c''') \in U} p_{0,0,2,c} \cdot p_{0,0,8,c'} \cdot p_{0,0,9,c''} \cdot p_{0,0,10,c'''})$ and then we deal with the two solutions for $x_4 \vee x_5 = 1$ by updating $p_{0,0,0,0} = p_{0,0,0,0} \cdot (p_{0,0,4,1} \cdot p_{0,0,5,0} + p_{0,0,4,0} \cdot p_{0,0,5,1})$

and afterwards we do the same for $x_6 \vee x_7 = 1$ by updating $p_{0,0,0,0} = p_{0,0,0,0} \cdot (p_{0,0,6,1} \cdot p_{0,0,7,0} + p_{0,0,6,0} \cdot p_{0,0,7,1})$ and we receive for each of the 3 starting vectors (b, b', b'') a sum polynomial and these 3 sum polynomials are added up to the return value of this branch. The other 3 cases arising from different branchings of x_1 in either formula are handled analogously.

8 Conclusion

In this paper, we introduced the concept of finding the most number of variables that can differ between a pair of solution and called it the Max Hamming Distance problem and we focused on the Max Hamming Distance XSAT problem. We introduced a DPLL algorithm with a nonstandard measure to bring the complexity down to $O(1.4983^n)$, and therefore beating both Dahllöf's state of the art algorithm to solve Max Hamming Distance XSAT and Fu, Zhou and Yin's algorithm to compute the Max Hamming Distance X3SAT.

Here is a possible direction where interested readers can take our work further. Our current nonstandard measure gave us a huge improvement in performance from $O(1.5761^n)$ if we were to use the standard measure. Can a more creative and cleverly designed measure bring the complexity of the algorithm down further?

An anonymous referee also pointed out that the large constant of Proposition 8 is large and that this constant goes in exponentiated form into the runtime; thus the algorithm is only of theoretical nature and not implementable in practice. Proposition 14 in the appendix gives a better value, but there is still room for improvement.

One might ask whether there are heuristics using known methods which might beat our algorithm. One such approach would be to enumerate all solutions of φ_1 at the beginning and then to solve for each of this solution φ_2 . In the general case, we note that the number of solutions of φ_1 is a badly conditioned function. Say if there are $n = 3m + 1$ variables consisting on m clauses with 4 variables where always the first variable is the same and the other 3 are uniquely to the clause, then the number of solutions is $1 + 3^m$ which is least $\Omega(1.4422^n)$. For each of these solutions, though not in this specific case, one has to solve on the other side a variable-weighted maximum XSAT formula which actually takes longer than solving XSAT; Porschen [14] solved this in $O(2^{0.244n})$ which is $O(1.184^n)$. So combining known algorithm might give only $O(1.7075^n)$, as $1.184 \cdot 1.4422 \geq 1.7075$. Thus even if the first bound can be improved, as the current bound requires an easy structure, we do not expect this method to give our bounds.

Hoi, Jain and Stephan [6] provide a better algorithm for computing the maximum hamming distance of X3SAT; however, this better bound exploits several properties of X3SAT which do not hold in general XSAT and this method does not generalise here. It is a often observed phenomenon that algorithms for X3SAT have a better time performance than their counterparts for the more general XSAT problem.

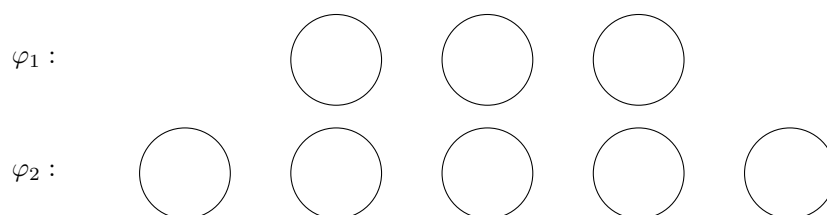
References

- 1 Andreas Blass and Yuri Gurevich. On the unique satisfiability problem. *Information and Control*, 55(1–3):80–88, 1982.
- 2 David Eppstein. Small maximal independent sets and faster exact graph coloring. *Proceedings of the Seventh Workshop on Algorithms and Data Structures, Springer Lecture Notes in Computer Science*, 2125:462–470, 2001.
- 3 David Eppstein. Quasiconvex analysis of multivariate recurrence equations for backtracking algorithms. *ACM Transactions on Algorithms*, 2(4):492–509, 2006.
- 4 Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science, EATCS, Springer, Berlin, Heidelberg, 2010.

- 5 Fedor V. Fomin, Fabrizio Grandoni and Dieter Kratsch. A measure and conquer approach for the analysis of exact algorithms. *Journal of the ACM*, 56(5):25, 2009.
- 6 Gordon Hoi, Sanjay Jain and Frank Stephan. A fast exponential time algorithm for Max Hamming X3SAT. *Foundations of Software Technology and Theoretical Computer Science, FSTTCS*, 2019.
- 7 Jesper Makhholm Byskov, Bolette Amitzbøll Madsen and Bolette Skjernaa. New algorithms for exact satisfiability. *Theoretical Computer Science*, 332(1-3):515–541, 2005.
- 8 Linlu Fu, Junping Zhou and Minghao Yin. Worst case upper bound for the maximum Hamming distance X3SAT problem. *Journal of Frontiers of Computer Science and Technology*, 6(7):664–671, 2012.
- 9 Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.
- 10 Martin Davis, George Logemann and Donald W. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, 1962.
- 11 Oliver Kullmann. New methods for 3-SAT decision and worst-case analysis. *Theoretical Computer Science*, 223(1–2):1–72, 1999.
- 12 Pierluigi Crescenzi and Gianluca Rossi. On the Hamming distance of constraint satisfaction problems. *Theoretical Computer Science*, 288(1):85–100, 2002.
- 13 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Springer, Berlin, Heidelberg, 2013.
- 14 Stefan Porschen. On variable-weighted exact satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, 51(1):27–54, 2007.
- 15 Vilhelm Dahllöf. Algorithms for Max Hamming Exact Satisfiability. *International Symposium on Algorithms and Computation, ISAAC 2005, Springer Lecture Notes in Computer Science*, 3827:829–383, 2005.
- 16 Vilhelm Dahllöf. *Exact Algorithms for Exact Satisfiability Problems*. PhD thesis, Department of Computer and Information Science, Linköping University, 2006.
- 17 Vilhelm Dahllöf, Peter Jonsson and Richard Beigel. Algorithms for four variants of the exact satisfiability problem. *Theoretical Computer Science*, 320(2–3):373–394, 2004.

A Appendix

Figure 14 shows an example of case 2. Each circle in both φ_1 and φ_2 represents an isolated group of clauses.



■ **Figure 14** Case 2.

An anonymous referee pointed out to the authors, that the size of the components after Case 1 is critical to the performance of the algorithm when implemented, as it is multiplicative constant in the runtime which is exponentiated, so the overall runtime multiplies with approximately $2^{m/2} \cdot \text{Poly}(n)$ which is an estimate to count all the weighted number of solutions of an component of size m with the weights being polynomials in u of degree n . One can go for this through all possible solutions by descent. Due to the exponentiation of m , it is important that one gets a bound on m to be as small as possible. Though a large

m does not influence the theoretical analysis of the algorithm in terms of its asymptotic complexity, it still makes the algorithm useless for practical implementations. The following relaxation from chains to pseudochains is critical for the improved bound on m .

► **Definition 12.** *A pseudochain is a sequence of clauses C_1, C_2, \dots, C_k such that two neighbouring clauses C_h, C_{h+1} overlap by exactly one variable and for two subsequent pairs of neighbouring clauses C_h, C_{h+1} and C_{h+1}, C_{h+2} , the variables in the overlaps are different.*

In this definition, the second condition implies that there are no pseudochains with a subsequence C_h, C_{h+1}, C_h , so that one cannot go back and forth between two clauses. Now for the following, the conditions in List 2 will be slightly generalised, in the sense that they can also apply to pseudochains and not only proper chains. So in Item 2, it is allowed that the neighbouring clause of c has a joint variable with the first clause, but this needs to be different from c . In Item 3, b, c are in further clauses and there are no constraint on what these further clauses are, they could even be the same clause; however, $b \neq c$ is required. In Items 4 and 5, it is not required that b, c are in further clauses and if they are, there are no constraints on what they are. In Item 6, it is only required that the two additional clauses where b and c are in have at least four variables, no further requirement is there. The two clauses have to be different, but they can have a joint variable. Note that b and c are then exactly in two clauses, as they have already six neighbours and as they cannot have eight neighbours out of which six have weight 1 by Item 1 to be done first when it applies. These relaxations do not influence the branching factors.

► **Proposition 13.** *Every component of the formulas after Case 1 with the more modified conditions in List 2 as above does not contain pseudochains of length six or more; furthermore, it contains no circular pseudochains like C_1, C_2, C_3, C_1 .*

Proof. Note that Item 1 enforces that no variable a in a component has eight or more neighbours, except in the case that the whole isolated component of the consists the variable a and eight neighbours which are all in clauses (a, \dots) consisting of a and two further variables. Furthermore the Simplification Rules enforce that there are no clauses of 1 or 2 variables, except the case of an isolated component consisting of a single 2-variable-clause. Thus no member of a pseudochain has more than six variables, as its neighbour must have at least three variables. Items 2 and 3 in a pseudochain enforce that no inner member of a pseudochain has three variables except in the case of a pseudochain C_1, C_2, C_3 where all three clauses have exactly three variables. If now a pseudochain is of the form $C_1, C_2, C_3, C_4, C_5, C_6$, the clauses C_2, C_3, C_4, C_5 have all at least four variables and it cannot be that both C_3, C_4 have both four variables, as then either Item 4 applies or one of them has a variable connecting to a further clause and Item 6 applies, causing a further reduction of the variables. Furthermore, it cannot be that one of C_3, C_4 has four variables and the other one five or more variables, as then either Item 5 or Item 6 applies. Furthermore, it cannot be that both C_3, C_4 have at least five variables, as then the connecting variable can be branched by Item 1. Thus pseudochains of six or more clauses do not survive until all branchings or cuts which can apply by List 2 are done.

Note that for a circular pseudochain $C_1, C_2, \dots, C_k, C_1$, it is required that C_2 and C_k connect to C_1 by different variables, as the writing should not depend on where one breaks the circle. Thus one can also view it as a long pseudochain $C_1, C_2, \dots, C_k, C_1, C_2, \dots, C_k$ and, as $k \geq 3$, this pseudochain has at least six members and does not exist after all branchings of the modified List 2 are done. ◀

► **Proposition 14.** *Every component of the formulas after Case 1 with the more modified conditions in List 2 as above has at most 80 variables.*

Proof. As seen in the preceding proposition, there are no circular pseudochains and no pseudochains of length six or more. So let C_1, C_2, \dots, C_k be a pseudochain of maximal length, therefore k is at most five.

Now consider the case $k = 5$. Let a be the common variable of C_2, C_3 and b be the common variable of C_3, C_4 . If there is a clause of distance 3 from C_3 , it is connected by clauses C_3, C_6, C_7, C_8 . Either C_6 does not connect to C_3 through a or does not connect to C_3 through b , say the first. Now $C_1, C_2, C_3, C_6, C_7, C_8$ is a pseudochain of length six which does not exist. Thus all clauses are either neighbours of C_3 or neighbours of neighbours of C_3 . So assume that C_6 is a neighbour of C_3 with common variable c which has a further neighbour C_7 which is not a neighbour of C_3 . Then C_6 has at least four and at most five variables. Thus c can only be in the clauses C_3, C_6 as these have together already at least seven variables and being in a further clause would cause c to have eight or more neighbours. So there are either three or four variables in C_6 other than c and these have each at most $8 - |C_6|$ neighbours which are not in C_6 . So if C_6 has four variables these are 3 neighbours of c outside C_3 plus $3 \cdot 4$ neighbours of neighbours; if C_6 has five variables these are 4 neighbours of c outside C_3 plus $4 \cdot 3$ neighbours of neighbours; in total c contributes to at most 16 neighbours and neighbours of neighbours. Note that 16 will only be reached, if C_6 has five variables and therefore C_3 four. So one has at most $\max\{4 \cdot (1 + 16), 5 \cdot (1 + 15)\} = 80$ variables in the isolated component.

If $k = 4$ then every clause would have at most distance two from C_2 which can be seen as follows: If there would be a clause of distance three then it would be a pseudochain C_2, C_5, C_6, C_7 which can be either extended to C_1, C_2, C_5, C_6, C_7 or C_3, C_2, C_5, C_6, C_7 and does not exist. Similarly, any clause has distance at most three from C_3 . If now a clause C_7 is of distance 2 from both C_2, C_3 then this is witnessed by C_2, C_5, C_7 and C_3, C_6, C_7 and C_5, C_6 do not connect to C_2, C_3 , respectively, through the common variable a of C_2 and C_3 . Thus C_7, C_5, C_2, C_6, C_7 would be pseudochain of length five with does not exist by assumption. Thus, every clause other than C_2 and C_3 is a neighbour of exactly one of these two clauses, but not of both. At most one of C_2, C_3 is of size five and both have at least size four. Thus a has up to 15 neighbours and neighbours of neighbours from one side and up to 16 from the other side, so the overall number of variables is at most $16 + 15 + 1 = 32$.

If $k = 3$ then every clause is a neighbour of C_2 . Each variable in C_2 has at most $8 - |C_2|$ neighbours which are not in C_2 and so the overall number of nodes is variables in the connected component is bounded by $|C_2| + |C_2| \cdot (8 - |C_2|) = |C_2| \cdot (9 - |C_2|)$ which is maximised at $|C_2| \in \{4, 5\}$ and is 20.

If $k = 2$ then every clause contains the connecting variable of C_1 and C_2 so that the overall number of variables is at most 9. If $k = 1$ then the clause has at most size 8 and there are no neighbouring clauses. ◀

It might be worth to mention that when dealing in a brute-force way with a component of size 80, one can break it down by taking, in the case of $k = 5$, first the central clause C_3 and branch into the subcases according to which of the variables in it is 1, each of these branches splits the component into subcomponents of size at most 16 which are easy to handle, as all variables of C_3 are set to constants. This would then make the case of dealing with isolated components to become more treatable from an implementation perspective. In the case of $k = 4$, one can branch the variable connecting C_2 and C_3 and get a similar breakdown of the component into two smaller, isolated components of up to 16 variables. The cases $k = 3, k = 2, k = 1$ have already very small components.

Cyclability in Graph Classes

Christophe Crespelle

Department of Informatics, University of Bergen, Norway
Christophe.Crespelle@uib.no

Carl Feghali

Department of Informatics, University of Bergen, Norway
Carl.Feghali@uib.no

Petr A. Golovach

Department of Informatics, University of Bergen, Norway
Petr.Golovach@uib.no

Abstract

A subset $T \subseteq V(G)$ of vertices of a graph G is said to be *cyclable* if G has a cycle C containing every vertex of T , and for a positive integer k , a graph G is *k-cyclable* if every subset of vertices of G of size at most k is cyclable. The **TERMINAL CYCLABILITY** problem asks, given a graph G and a set T of vertices, whether T is cyclable, and the **k-CYCLABILITY** problem asks, given a graph G and a positive integer k , whether G is *k-cyclable*. These problems are generalizations of the classical **HAMILTONIAN CYCLE** problem. We initiate the study of these problems for graph classes that admit polynomial algorithms for **HAMILTONIAN CYCLE**. We show that **TERMINAL CYCLABILITY** can be solved in linear time for interval graphs, bipartite permutation graphs and cographs. Moreover, we construct certifying algorithms that either produce a solution, that is, a cycle, or output a graph separator that certifies a no-answer. We use these results to show that **k-CYCLABILITY** can be solved in polynomial time when restricted to the aforementioned graph classes.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Graph algorithms analysis

Keywords and phrases Cyclability, interval graphs, bipartite permutation graphs, cographs

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.16

Funding The research leading to these results has received funding from the Research Council of Norway via the projects “CLASSIS” and “MULTIVAL”, and from the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 749022.

1 Introduction

A subset $T \subseteq V(G)$ of vertices of a graph G is said to be *cyclable* if G has a cycle C containing every vertex of T . In this case, C is said to *cover* T . We assume that a single element set is cyclable. For a positive integer k , a graph G is *k-cyclable* if every set T of size at most k is cyclable. The *cyclability* of G , denoted $\text{cyc}(G)$, is the maximum k such that G is *k-cyclable*. We consider the following generalizations of the classical **HAMILTONIAN CYCLE** problem.

TERMINAL CYCLABILITY

Input: A graph G and a nonempty set $T \subseteq V(G)$ of *terminals*.
Task: Decide whether T is cyclable.

k-CYCLABILITY

Input: A graph G and a positive integer k .
Task: Decide whether G is *k-cyclable*.



© Christophe Crespelle, Carl Feghali, and Petr A. Golovach;
licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 16; pp. 16:1–16:13

Leibniz International Proceedings in Informatics



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

The investigation of `TERMINAL CYCLABILITY` and k -`CYCLABILITY` started in the 1960s with the pioneer work of Dirac [18] who proved that, for each $k \geq 2$, every k -connected graph is k -cyclable. Since then, a number of related results have been obtained and the majority of them follow the line of research of Dirac [18]: to give sufficient conditions for a graph G to be k -cyclable or for a given subset $T \subseteq V(G)$ to be cyclable; we refer the reader to the survey paper of Gould [22] for results of this type.

From the computational complexity viewpoint, both `TERMINAL CYCLABILITY` and k -`CYCLABILITY` are at least as hard as the `HAMILTONIAN CYCLE` problem, which is well-known to be NP-complete [19]. Positive results can be found in the Parameterized Complexity framework (we refer to the recent book of Cygan et al. [12] for an introduction to the field). For instance, by the celebrated results of Robertson and Seymour [29] about the `DISJOINT PATHS` problem, `TERMINAL CYCLABILITY` is fixed-parameter tractable (FPT) when parameterized by $|T|$. So far, the best known FPT (randomized) algorithm is due to Björklund, Husfeldt and Taslaman [3]. Golovach et. al. [20] also proved that deciding if G is k -cyclable is `co-W[1]`-hard for split graphs and that k -`CYCLABILITY` is FPT on planar graphs when parameterized by k .

There is also a long history of research on `HAMILTONIAN CYCLE` and related problems for the classes of cographs, bipartite permutation graphs, interval graphs and some of their superclasses (see [5, 7, 8, 11, 13, 14, 15, 16, 23, 24, 26, 28] and the bibliography therein).

A lot of this research is connected with the conjecture of Chvátal [9]; see the survey of Bauer, Broersma and Schmeichel [2] for the statement, history and details around the conjecture. Let $c(G)$ denote the number of connected components of a graph G , Chvátal [9] observed that if there exists a vertex separator S of a graph G such that $c(G - S) > |S|$, then G has no Hamiltonian cycle. Hence, the condition that $c(G - S) \leq |S|$ holds for every separator of a graph G is a necessary Hamiltonicity condition. For interval graphs, bipartite permutation graphs and cographs that are connected and have at least three vertices, this condition turns out to be also sufficient [11, 13, 15]. Motivated by this necessary condition, Jung [25] defined the *scattering* number of a noncomplete graph G as

$$\text{sc}(G) = \max\{c(G - S) - |S| \mid S \text{ is a separator of } G\}, \quad (1)$$

and the set S^* for which the maximum in (1) is achieved is called a *scattering* set. For a complete graph G , $\text{sc}(G) = -\infty$. For the class of cocomparability graphs G with at least three vertices (that is a superclass of the classes of interval graphs and permutation graphs), the following two dualities were established in [15]. Firstly, it is shown that G has a Hamiltonian cycle if and only if $\text{sc}(G) \leq 0$ and, secondly, that the set of vertices of G can be covered by at most k vertex-disjoint paths if and only if $\text{sc}(G) \leq k$.

From these equivalences, one can construct *certifying* polynomial time algorithms for `HAMILTONIAN PATH` and `HAMILTONIAN CYCLE` problems. Note that a certifying algorithm outputs, together with a solution, a *certificate* that demonstrates the correctness of the solution that can be verified independently. Typically, the size of a certificate should be small with respect to the input size and the verification algorithm should be simple. The main advantage of certifying algorithms over standard ones is that their implementations are a great deal more reliable and can be used without knowing the code; see the survey papers [1, 27] for an introduction to certifying algorithms. The certifying algorithms for `HAMILTONIAN PATH` and `HAMILTONIAN CYCLE` either output a Hamiltonian path or a Hamiltonian cycle, or produce a separator that certifies a no-answer [10, 15, 11, 7].

We continue the study of `TERMINAL CYCLABILITY` and k -`CYCLABILITY` from a complexity viewpoint by first showing that analogous dualities hold for these problems on interval graphs, bipartite permutation graphs and cographs (see Section 2 for the formal definitions of these

graph classes). We will then show how to construct, from these dualities, polynomial time algorithms for `TERMINAL CYCLABILITY` and `k-CYCLABILITY` on these graph classes, which are also certifying algorithms in the case of `TERMINAL CYCLABILITY`. In fact, for `TERMINAL CYCLABILITY` we will consider a slightly more general problem. To be more precise, let G be a graph and let $T \subseteq V(G)$ such that T is not a clique. Let $c_T(G)$ denote the number of connected components of G containing some vertex of T and say that a subset $S \subseteq V(G)$ is a T -separator of G if $c_T(G - S) \geq 2$. The T -scattering number of G is given by

$$sc_T(G) = \max\{c_T(G - S) - |S| \mid S \text{ is a } T\text{-separator}\}, \tag{2}$$

and the set S^* for which the maximum in (2) is achieved is called a T -scattering set. A cycle or a family of vertex-disjoint paths containing the vertices of T is said to be a T -cycle-segment cover. The size of a T -cycle-segment cover is defined to be zero if it is a cycle and to be the number of paths in the family otherwise. The T -cycle-segment cover number, denoted $seg_G(T)$, is the minimum size of a T -cycle-segment cover.

As one of our main contributions, we will show that if G is an interval graph, a bipartite permutation graph or a cograph and T is not a clique, then $seg_G(T) \leq r$ if and only if $sc_T(G) \leq r$. This, in turn, will allow us to solve in polynomial (linear) time the following decision problem that generalizes `HAMILTONIAN CYCLE` and `PATH COVER`.

CYCLE SEGMENT COVER

Input: A graph G , a nonempty set $T \subseteq V(G)$ of *terminals* and a nonnegative integer r .
Task: Decide whether $seg_G(T) \leq r$.

Moreover, our algorithms for each graph class either produce a solution, that is, a T -cycle-segment cover, or return a T -separator S^* such that $c_T(G - S^*) - |S^*| > r$ that certifies a no-answer (unless T is a 2-clique and is not cyclable in G , which is the only case when there exists no T -separator S^* that certifies a no-answer – in this case, it suffices to check whether T induces a bridge in G). More formally, we will establish the following theorem.

► **Theorem 1.** *There is an algorithm that, given an instance (G, T, r) of `CYCLE SEGMENT COVER`, where G is an interval graph, a bipartite permutation graph or a cograph and T is not a 2-clique, either finds a T -cycle segment cover of size at most r or a T -separator with $c_T(G - S^*) - |S^*| > r$ that certifies a no-answer in $\mathcal{O}(|V(G)| + |E(G)|)$ time.*

In fact, for cographs we have a slightly better result: the algorithm runs in time $\mathcal{O}(|V(G)|)$ if the cotree of G is given (see Section 5 for the definition). We then use these results to solve `k-CYCLABILITY` for interval graphs, bipartite permutation graphs and cographs.

► **Theorem 2.** *For a graph G that is an interval graph, a bipartite permutation graph or a cograph, `k-CYCLABILITY` can be solved in time $\mathcal{O}(|V(G)|^3)$.*

In proving Theorem 2, the following definition will be essential. For a positive integer k , we define the k -scattering number of a graph G as

$$sc^k(G) = \max\{c(G - S) - |S| \mid S \text{ is a separator of } G \text{ s.t. } |S| \leq k - 1\}, \tag{3}$$

and $sc^k(G) = -\infty$ if it has no separator of size at most $k - 1$ (we assume that the empty set is a separator of a disconnected graph).

To prove Theorem 2, we first show that if G is an interval graph, a bipartite permutation graph or a cograph and k is a positive integer, then G is k -cyclable if and only if $\text{sc}^k(G) \leq 0$. Our approach for solving k -CYCLABILITY for interval graphs and cographs then consists of constructing polynomial time algorithms that compute the k -scattering number for these graph classes for all $k \in \{1, \dots, |V(G)|\}$ in cubic time. For bipartite permutation graphs, we use a different approach that gives a better running time.

The extended abstract is organized as follows. In Section 3, we sketch our algorithm for CYCLE SEGMENT COVER for interval graphs and describe how to solve k -CYCLABILITY for interval graphs. In Sections 4 and 5, we very briefly discuss our afore-mentioned results for, respectively, bipartite permutation graphs and cographs. We conclude the paper in Section 6 with some open problems. Due to space constraints, the details of most proofs are omitted.

2 Preliminaries

We consider only finite undirected simple graphs and follow the standard graph theoretic notation and terminology (see, e.g., [17]). We use n to denote the number of vertices and m the number of edges of the considered graphs unless it creates confusion. We say that a graph G is an *interval* graph if there is a family \mathcal{I} of closed intervals of the line (called *interval model* or *representation*) such that G is isomorphic to the intersection graph of \mathcal{I} . A graph G is a *permutation* graph if there is an ordering v_1, \dots, v_n of its vertices and a permutation $\pi: \{1, \dots, n\} \rightarrow \{1, \dots, n\}$ such that for $1 \leq i < j \leq n$, v_i and v_j are adjacent in G if and only if $\pi(i) > \pi(j)$. A graph is a *bipartite permutation* graph if it is both a bipartite graph and a permutation graph. A graph is a *cograph* if it has no induced subgraph isomorphic to the path on four vertices. We refer to [6, 21] for detailed introductions to these graph classes.

It is convenient to dispense with easy instances of our problems. An instance (G, T, r) of CYCLE SEGMENT COVER, where T is a clique, is a yes-instance, unless $|T| = 2$, $r = 0$ and T induces a bridge in G . Similarly, an instance (G, k) of k -CYCLABILITY, where G is a complete graph, is a yes-instance unless $|V(G)| = 2$ and $k = 2$.

► **Remark 3.** In the sequel, we assume that G is not complete and T is not a clique.

3 Interval graphs

In this section, we prove Theorems 1 and 2 for interval graphs. Our algorithms use a specific interval representation of the input graph. A *clique path* of a graph G is a sequence of cliques C_1, \dots, C_s of G such that

- (i) $C_1 \cup \dots \cup C_s = V(G)$,
- (ii) for all $uv \in E(G)$, there is $i \in \{1, \dots, s\}$ such that $u, v \in C_i$,
- (iii) for all $v \in V(G)$, if $v \in C_i \cap C_j$ for some $1 \leq i < j \leq s$, then $v \in C_h$ for all $h \in \{i, \dots, j\}$.

It is usually assumed in the definition of a clique path (see, e.g., [6, 21]) that C_1, \dots, C_s are maximal cliques of G . Here, we relax the standard definition and do not require the cliques to be inclusion-wise maximal so that some cliques may be identical or empty. It is well-known [6, 21] that a graph is an interval graph if and only if it has a clique path. The classical recognition algorithm for interval graphs of Booth and Lueker [4] constructs a clique path in time $\mathcal{O}(n + m)$. As we intend to design an $\mathcal{O}(n + m)$ -time algorithm, we can assume from now on that the input graph is given with its clique path.

For a vertex $v \in V(G)$, we let $\ell_v = \min\{i \in \{1, \dots, s\} \mid v \in C_i\}$ and $r_v = \max\{i \in \{1, \dots, s\} \mid v \in C_i\}$. We say that ℓ_v and r_v are the *left bound* and *right bound* of v respectively. Notice that the intervals $[\ell_v, r_v]$ of the real line for $v \in V(G)$ form an interval representation of G . For $1 \leq i \leq j \leq s$, we denote $C_{i,j} = \cup_{h=i}^j C_h$.

We use the following well-known observation about separators of interval graphs that results from the definition of a clique path (see, e.g., [6, 21]).

► **Observation 4.** *Let G be a connected interval graph with a clique path C_1, \dots, C_s . If $X = C_{1,i} \setminus C_{i+1} \neq \emptyset$ and $Y = C_{i+1,s} \setminus C_i \neq \emptyset$ for some $i \in \{1, \dots, s-1\}$, then $C_i \cap C_{i+1}$ is a separator of G such that X and Y are in distinct components of $G - (C_i \cap C_{i+1})$.*

In Subsection 3.1 we solve CYCLE SEGMENT COVER for interval graphs. In Subsection 3.2, we show how to compute the k -scattering number for interval graphs and use this result to solve k -CYCLABILITY.

3.1 Algorithm for Terminal Cyclability and Cycle Segment Cover

In this subsection, we describe our algorithms for TERMINAL CYCLABILITY and CYCLE SEGMENT COVER. More formally, we prove the following theorem.

► **Theorem 5.** *There is an algorithm that, given an instance (G, T) of TERMINAL CYCLABILITY where G is an interval graph and T is not a 2-clique, finds either a cycle of G covering T or a T -separator S^* with $c_T(G - S^*) - |S^*| > 0$ that certifies a no-answer in time $\mathcal{O}(n + m)$.*

The next part of the subsection contains a sketch of the proof of Theorem 5. We construct an algorithm that tries to find a cycle of a graph G that covers T . If it fails, we use the information obtained by the algorithm to construct a T -separator. The algorithm is inspired by the algorithm for finding a Hamiltonian cycle in interval graphs of Keil [26]. For us, it is more convenient to use a tailored variant of the algorithm from [7] for a more general problem as this allows us to use some results of [7] as black boxes. For this, we need some auxiliary results.

Let G be an interval graph given together with its clique path C_1, \dots, C_s , and let $T \subseteq V(G)$ such that T is not a clique (see Remark 3). If G has at least two distinct connected components containing vertices of T , then G has no cycle covering T and the algorithm returns $S^* = \emptyset$. We can thus assume that the vertices of T are in the same connected component, so we can discard the other components if they exist. Clearly, all this can be done in linear time. So we can safely assume, from now on, that G is connected.

Our algorithm (Algorithm 1 below) scans the clique path of G from the leftmost clique to the rightmost and selects vertices from these cliques depending on their bounds. In order to break ties between a subset of vertices having the same right bound (Lines 4 and 9) or the same left bound (Line 14), we use a pre-decided arbitrary total order π on the vertices of G and always select the leftmost vertex in the subset with respect to π . Let $p = \min\{r_v \mid v \in T\}$ and $q = \max\{\ell_v \mid v \in T\}$. Let w_b be the minimum vertex of T with respect to π such that $r_{w_b} = p$. Analogously, let w_e be the maximum vertex of T with respect to π such that $\ell_{w_e} = q$. Since T is not a clique, it follows that $p < q$, $w_b \neq w_e$ and $w_b w_e \notin E(G)$.

Algorithm 1 tries to construct two (w_b, w_e) -paths P_1 and P_2 that are internally vertex-disjoint such that $T \subseteq V(P_1) \cup V(P_2)$. If the algorithm succeeds, then the concatenation of P_1 and P_2 is a cycle covering T . Initially, $P_1 = P_2 = w_b$. Afterwards, the algorithm attaches new vertices to one of the end-vertex of the two paths, which we call the *extremity* of the path. For each P_i , the initial extremity of P_i is w_b and whenever we append a new vertex to the path, this vertex becomes the new extremity. It is easy to prove the following property.

► **Lemma 6.** *If Algorithm 1 returns P_1 and P_2 , then P_1 and P_2 are internally vertex-disjoint (w_b, w_e) -paths that contain all the vertices of T .*

■ **Algorithm 1** An algorithm for interval graphs that finds two internally vertex-disjoint (w_b, w_e) -paths P_1 and P_2 such that $T \subseteq V(P_1) \cup V(P_2)$.

```

1 begin
2   let  $P_1 = P_2 = w_b$ ;
3   for  $t = p$  to  $q - 1$  do
4     choose  $P_i \in \{P_1, P_2\}$  such that the extremity of  $P_i$  has the leftmost right
       bound;
5     attach the vertices  $x \in T \setminus (V(P_1) \cup V(P_2))$  s.t.  $r_x = t$  to  $P_i$ ;
6     for  $i = 1, 2$  do
7       if the extremity of  $P_i$  has right bound at most  $t$  then
8         if the subset of vertices  $y \in (C_t \cap C_{t+1}) \setminus (V(P_1) \cup V(P_2))$  is not empty
9           then extend  $P_i$  by attaching such a  $y$  having the leftmost right
              bound;
10          else report that  $T$  is not cyclable and quit;
11        end
12      end
13    end
14    attach the vertices  $x \in T \setminus \{w_e\}$  s.t.  $l_x = q$  to  $P_1$ , then attach  $w_e$  to  $P_1$  and  $P_2$ ;
15    return  $P_1$  and  $P_2$ ;
16 end

```

Our next aim is to show that if Algorithm 1 reports that T is not cyclable, then there is a T -separator S^* such that $c_T(G - S^*) > |S^*|$. The main observation that we shall use to construct the set S^* is that for $T = V(G)$, Algorithm 1 is precisely an algorithm for finding a Hamiltonian cycle in an interval graph. Our algorithm can be interpreted, to a large extent, as a variant of Keil's algorithm [26] or of Algorithm 1 of Broersma et al. [7] (the main difference between our algorithm and theirs is that our algorithm does not try to include, in the constructed paths, all vertices that it encounters). In particular, in [7] an explicit construction is given of a separator S of G such that $c(G - S) > |S|$ for the case when G has no Hamiltonian cycle. We adapt their approach by first altering our graph so as to allow the use of some of their results. The rest of our arguments are related to the ones in [7] but have their own features and are more than just a variation.

Assume that Algorithm 1 stops at Line 10 for $t = t^*$. Note that from the range of variation of t in the main loop (Line 3), we have $t^* < q$. Denote by P_1^* and P_2^* the paths constructed by the algorithm before it quits. We require some additional notations from [7].

For real numbers $a \leq b$, $[a, b] = \{x \in \mathbb{R} \mid a \leq x \leq b\}$, $[a, b) = \{x \in \mathbb{R} \mid a \leq x < b\}$, and $(a, b) = \{x \in \mathbb{R} \mid a < x < b\}$. If vertex u has been processed by the algorithm and attached to a path at some step t of the for loop at Lines 3–13, we say that u has been *activated at time* $a_u = t$. We define $a_{w_b} = p$. If u is activated and a vertex v has been attached to u at some step $t' \geq t$ of the for loop, we say that u has been *deactivated at time* $d_u = t'$. Thus, $\ell_u \leq a_u \leq d_u \leq r_u$ and u is said to be *free*, *active* or *depleted* on, respectively, the intervals $[\ell_u, a_u)$, $[a_u, d_u)$ and $[d_u, r_u]$. Note that some of these intervals may be empty. Whenever we say that u is free (respectively, active or depleted) on an interval I of the real line, this means that $I \subseteq [\ell_u, a_u)$ (respectively, $I \subseteq [a_u, d_u)$ or $I \subseteq [d_u, r_u]$). We also say that $v \in V(P_i^*)$ for $i \in \{1, 2\}$ is a *descendant* of $u \in V(P_i^*)$ if v was attached to P_i^* after u and that v is the *last descendant* on an interval I if v is the last vertex attached to P_i^* at steps $t \in I$ of the for loop at Lines 3–13. A vertex v is said to be *renounced* if it is missed by the algorithm, that is, $\ell_v \leq t^*$ and $v \notin V(P_1^*) \cup V(P_2^*)$. The set of renounced vertices is denoted by R .

Let $G^* = G - R$, and let $T^* = V(G) \setminus R$. For $i \in \{1, \dots, s\}$, denote $C_i^* = C_i \setminus R$. Clearly, C_1^*, \dots, C_s^* is a clique path of G^* . Recall that for $1 \leq i \leq j \leq s$, $C_{i,j}^* = \bigcup_{h=i}^j C_h^*$.

The description of Algorithm 1, with tie-breaking order π , implies the following property.

► **Lemma 7.** *Algorithm 1 for the instance (G^*, T^*) of TERMINAL CYCLABILITY, with tie-breaking order π , quits at Line 10 and constructs the paths P_1^* and P_2^* .*

As mentioned above, the fact that our Algorithm 1 for (G^*, T^*) works along the same lines as Algorithm 1 of [7] will allow us to use the following Lemma 2.2 of [7].

► **Lemma 8 ([7]).** *Let $t \in \{p, \dots, q-1\}$ such that Algorithm 1 with input (G^*, T^*) either finishes iteration t of the for loop at Lines 3–13 or terminates at Line 10 within iteration t . If there is at least one depleted vertex on the interval $(t, t+1)$, then there exists an integer t' such that $p \leq t' < t$ with the following properties:*

- (i) $(C_{t'+1,t}^* \setminus (C_{t'}^* \cup C_{t+1}^*)) \neq \emptyset$,
- (ii) there exists a unique vertex $u \in C_{t'}^* \cap C_{t+1}^*$ such that u is active on $(t', t'+1)$ and u is depleted on $(t, t+1)$,
- (iii) all vertices that are active on $(t, t+1)$ are also active on $(t', t'+1)$, with the only possible exception of the last descendant v of u on $(t, t+1)$ which may be free on $(t', t'+1)$,
- (iv) all vertices that are depleted on $(t, t+1)$ are also depleted on $(t', t'+1)$, except u which is active on $(t', t'+1)$,
- (v) all vertices that are active on $(t', t'+1)$ are also active on $(t, t+1)$, except u which is depleted on $(t, t+1)$, and
- (vi) all vertices that are free on $(t', t'+1)$ are also free on $(t, t+1)$, with the only possible exception of v if it is active on $(t, t+1)$.

For our purposes, we need one additional property (vii), stated by Lemma 9 below, which can be proved to be satisfied by the minimum t' satisfying properties (i)–(vi) of Lemma 8.

► **Lemma 9.** *Let $t \in \{p, \dots, q-1\}$ such that Algorithm 1 with input (G^*, T^*) either finishes iteration t of the for loop at Lines 3–13 or terminates at Line 10 within iteration t . If there is at least one depleted vertex on the interval $(t, t+1)$, then there exists an integer $t' < t$ that satisfies the conditions (i)–(vi) and the following property:*

- (vii) there is $x \in V(G^*)$ such that $a_x = t'$ and x is active during $(t', t'+1)$.

We now use Lemma 9 to construct the following decreasing sequence t_1, t_2, \dots of positive integers. We set $t_1 = t^*$. Then we construct t_{i+1} from the already constructed t_i as follows. If, for $t = t_i$, there is at least one depleted vertex on $(t, t+1)$, then find $t' < t$ such that the conditions (i)–(vii) of Lemmas 8 and 9 are satisfied and set $t_{i+1} = t'$. We stop the construction if there is no depleted vertex on $(t, t+1)$ for $t = t_i$. Clearly, the constructed sequence is finite and we denote it by t_1, \dots, t_k , with k being its number of elements.

For $i \in \{1, \dots, k\}$, we define $S_i = C_{t_i}^* \cap C_{t_{i+1}}^*$ and $S^* = \bigcup_{i=1}^k S_i$. We require the following crucial property of S^* that was shown in the proof of Theorem 2.1 of [7].

► **Lemma 10 ([7]).** *The set S^* is a separator of G^* and $c(G^* - S^*) \geq k + 1 > |S^*|$.*

From Lemma 10, we establish an essential result for the proof of Theorem 5.

► **Lemma 11.** *The set S^* is a T -separator in G and $c_T(G - S^*) > |S^*|$.*

Mindful of Lemma 10, Lemma 11 intuitively states that the set R of renounced vertices of G does not play an important role in finding a T -separator of G whose removal “maximises” the number of resulting components containing some member of T .

Proof. We use the second inequality of Lemma 10 ($k + 1 > |S^*|$) and we prove in addition that $c_T(G - S^*) \geq k + 1$. To this purpose, we define subsets X_i with $0 \leq i \leq k$ (see below) for which we show that each X_i has a non-empty intersection with T (Claim 12) and the sets X_i are separated by S^* in G (Claim 13). Let $X_k = C_{1,t_k}^* \setminus C_{t_k+1}^*$, $X_j = C_{t_{j+1}+1,t_j}^* \setminus (C_{t_{j+1}}^* \cup C_{t_j+1}^*)$ for $j \in \{1, \dots, k-1\}$ and $X_0 = C_{t_1+1,s}^* \setminus C_{t_1}^*$. We have two claims.

▷ **Claim 12.** For all i such that $0 \leq i \leq k$, $X_i \cap T \neq \emptyset$.

Let us first argue that $w_b \in X_k$ and $w_e \in X_0$. As the main loop of Algorithm 1 (Lines 3–13) starts iterating with $t = p$, there is no depleted vertex on $(t, t + 1)$ for $t < p$. Hence $t_k \geq p$ and given that $w_b \in C_p \setminus C_{p+1}$ it follows that $w_b \in C_p^* \setminus C_{p+1}^*$. In other words, $w_b \in C_{1,p}^* \setminus C_{p+1}^*$ and so $w_b \in X_k$. Similarly, $w_e \in C_q \setminus C_{q-1} = C_q^* \setminus C_{q-1}^*$ since $t^* < q$, which implies that $w_e \in C_{q,s}^* \setminus C_{q-1}^*$ and so $w_e \in X_0$.

Now fix some $i \in \{1, \dots, k-1\}$. By construction of the sequence t_1, \dots, t_k the conditions (i)–(vii) of Lemmas 9 are satisfied with $t = t_i$ and $t' = t_{i+1}$. By (ii), there is a vertex $u \in C_{t_{i+1}}^* \cap C_{t_i+1}^*$ that is active on $(t_{i+1}, t_{i+1} + 1)$ and depleted on $(t_i, t_i + 1)$. This means that $t_{i+1} + 1 \leq d_u \leq t_i$ and $r_u \geq t_i + 1$. From these bounds, some vertex x must have been attached to the path with extremity u at time $t = d_u$ in Line 5 of Algorithm 1 and so, again by the algorithm, must be a member of T with $r_x = d_u < t_i + 1$.

If we can show that $x \in X_i = C_{t_{i+1}+1,t_i}^* \setminus (C_{t_{i+1}}^* \cup C_{t_i+1}^*)$, then the claim follows. Since $r_x < t_i + 1$, x is not the last descendant of u on $(t_{i+1}, t_i + 1)$ and is not free on $(t_i, t_i + 1)$. Hence, by (vi), x is also not free on $(t_{i+1}, t_{i+1} + 1)$. Therefore, $t_{i+1} < \ell_x \leq r_x < t_i + 1$ and hence $x \notin C_{t_{i+1}}^* \cup C_{t_i+1}^*$. This means that $x \in X_i$ and the claim is proved.

▷ **Claim 13.** For all distinct $i, j \in \{0, \dots, k\}$ and every $x \in X_i$ and $y \in X_j$, x and y are in distinct components of $G - S^*$.

It suffices to show that for every $z \in R$ there is $i \in \{0, \dots, k\}$ such that $t_{i+1} + 1 \leq \ell_z \leq r_z \leq t_i$, where we assume that $t_0 = s$ and $t_{k+1} = 0$. Indeed, this implies that for all $i \in \{1, \dots, k\}$, $C_{t_i} \cap C_{t_{i+1}} \subseteq S^*$ and the claim then follows from Observation 4.

Suppose, towards a contradiction, that there is some $z \in R$ and some $i \in \{1, \dots, k\}$ with the property that $\ell_z \leq t_i < r_z$, and assume without loss of generality that i is minimum with respect to these conditions. We first show that $i > 1$. Indeed, if $i = 1$ then $\ell_z \leq t_1 = t^*$ and $r_z > t^*$. But as z is a member of R , it follows that $z \in (C_{t^*} \cap C_{t^*+1}) \setminus (V(P_1) \cup V(P_2))$, which is empty since the condition at Line 8 failed at time $t = t^*$ and Algorithm 1 quit at Line 10, a contradiction.

Therefore $i > 1$. Recall in this case that t_i was constructed from $t = t_{i-1}$ by choosing $t_i < t$ such that for $t' = t_i$ the conditions (i)–(vii) of Lemmas 8 and 9 hold. To finish off the proof of the claim, we will show that $\ell_z \leq t_i \leq t_{i-1} < r_z$, giving the final contradiction since, by the minimality of i , there is no $z \in R$ with $\ell_z \leq t_{i-1} < r_z$.

We already know that $\ell_z \leq t_i \leq t_{i-1}$. By (vii), there is $x \in V(G^*)$ such that $a_x = t_i$ and x is active on $(t_i, t_i + 1)$. By (ii) and (v), x is either active or depleted on $(t_{i-1}, t_{i-1} + 1)$. In either case, $r_x \geq t_{i-1} + 1 > t_i$. Given that $a_x = t_i$ (that is, x was attached to some path at the t_i -th iteration of the for loop of Algorithm 1 at Lines 3–13) and $r_x > t_i$, it follows that x was attached to some path at Line 9 of Algorithm 1. Hence, the right bound of x is less than or equal to that of z , which implies $r_z \geq t_{i-1} + 1$ and the claim is proved.

From Claims 12 and 13, it follows that $c_T(G - S^*) \geq k + 1$ and consequently $c_T(G - S^*) \geq |S^*|$ from the second inequality of Lemma 10. ◀

We are now ready to complete the sketch of the proof of Theorem 5.

Proof sketch of Theorem 5. As mentioned earlier, we can assume that G is connected. We can also assume that we can compute in $O(n + m)$ time a clique path C_1, \dots, C_s of G , where each clique is inclusion maximal (by the algorithm of Booth and Lueker [4]), so $s \leq n$. We also compute the left bound and right bound ℓ_v and r_v of each vertex $v \in V(G)$, which allows us to find the vertices w_b and w_e in time $O(n)$. Also in time $O(n)$, we construct the list L consisting of the right bounds of the elements of $T \setminus \{w_b, w_e\}$ in increasing order.

Next, we run Algorithm 1. At each iteration t of the for loop, Algorithm 1 only needs to decide whether the path under consideration should be extended (at Line 5 and/or Line 9), after which we also need to determine which vertex of G is to be attached to the extremity of this path. Now, given that a path is extended only if the right bound of its extremity (condition at Line 7) or of some vertex of T (condition at Line 5) is precisely t , the first computation takes constant time with the list L at hand and hence $O(n)$ time in total. Moreover, whenever a path is to be extended, we scan the vertices of $C_t \subseteq N_G(v)$. As we never extend more than once a path with the same extremity, this takes in total time $O(\sum_{v \in V} d_G(v)) = O(m)$. Thus, Algorithm 1 runs in $O(n + m)$ time. Now, if Algorithm 1 finishes at Line 15 and outputs two paths P_1 and P_2 , then we are done by Lemma 6. Otherwise, Algorithm 1 finishes at Line 10, so we work backwards through the algorithm in order to construct the sequence t_1, \dots, t_k and the set $S^* = \bigcup_{i=1}^k S_i$ that certifies a negative answer. With a careful implementation, this can be done in $O(n + m)$ time as well. ◀

► **Remark 14.** To avoid any misunderstanding, the assumption that the cliques of the clique path of G in the proof of Theorem 5 are maximal is crucial for the running time analysis. But it is also necessary to prove Lemmas 6–11 without this maximality assumption, since the cliques C_1^*, \dots, C_s^* of the graph G^* (obtained from G by the removal of the set of renounced vertices) are not necessarily maximal. In other words, it is essential to start off with an input graph whose clique path consists of maximal cliques but to also prove statements that concern interval graphs whose clique path may contain non-maximal cliques.

To solve CYCLE SEGMENT COVER, we require a folklore observation (see, e.g., [13]).

► **Observation 15.** *Let G be a graph, $T \subseteq V(G)$ and k be a positive integer. If G' is obtained from G by adding k universal vertices to G , then $sc_T(G) \leq k$ if and only if $sc_T(G') \leq 0$.*

Combining this observation with a careful analysis of the running time of the previous algorithm applied to G' instead of G , we obtain the following result.

► **Theorem 16.** *There is an algorithm that, given an instance (G, T, r) of CYCLE SEGMENT COVER, where G is an interval graph and T is not a 2-clique, finds either a cycle or a family of at most r paths that cover T or a T -separator S^* with $c_T(G - S^*) - |S^*| > r$ that certifies a no-answer in time $O(n + m)$.*

3.2 k -Cyclability for interval graphs

In this subsection we prove Theorem 2 for interval graphs. From Theorem 5 and from the definition of $sc^k(G)$, an interval graph G with at least three vertices is k -cyclable if and only if $sc^k(G) \leq 0$. So to solve k -CYCLABILITY on interval graphs, it is sufficient to construct a polynomial algorithm that computes the k -scattering number of G for any $k \leq n - 1$. The only remaining task will consist in finding the largest integer k such that $sc^k(G) \leq 0$. We use the following lemma.

► **Lemma 17.** *Let G be an interval graph, let C_1, \dots, C_s be a clique path of G , where C_1, \dots, C_s are pairwise-distinct maximal cliques of G , and let S be a separator of G . Then,*

$$\text{there exist } 1 \leq t_1 < \dots < t_r < s \text{ such that } S' = \bigcup_{i=1}^r (C_{t_i} \cap C_{t_i+1}) \quad (4)$$

and $S' \subseteq S$ and S' is a separator of G such that $c(G - S') \geq c(G - S)$.

Informally, the above lemma states that, in computing the k -scattering number, one can restrict their attention to a subset of separators, namely these separators that satisfy (4), which we call *canonical separators*. Thus, Lemma 17 implies that, for an interval graph G , $\text{sc}^k(G)$ can be equivalently defined as

$$\text{sc}^k(G) = \max\{c(G - S) - |S| \mid S \text{ is a canonical separator of } G \text{ s.t. } |S| \leq k - 1\}. \quad (5)$$

To solve k -CYCLABILITY, our algorithm computes canonical separators via a dynamic programming scheme on the given clique path C_1, \dots, C_s of a non-complete interval graph.

► **Theorem 18.** *For a non-complete interval graph G , one can solve k -CYCLABILITY and compute the scattering numbers $\text{sc}^k(G)$ for all $k \in \{1, \dots, n - 1\}$ in time $\mathcal{O}(n^3)$.*

4 Bipartite permutation graphs

In this section we briefly sketch our results for CYCLE SEGMENT COVER and k -CYCLABILITY on bipartite permutation graphs. Let $G = (V_1, V_2, E)$ a bipartite graph. Let $\sigma_1 = \langle u_1, \dots, u_p \rangle$ and $\sigma_2 = \langle v_1, \dots, v_q \rangle$ be orderings of, respectively, V_1 and V_2 . It is said that (σ_1, σ_2) is a *strong ordering* of G if for every $1 \leq i < i' \leq p$ and $1 \leq j' < j \leq q$, if $u_i v_j, u_{i'} v_{j'} \in E(G)$, then $u_i v_{j'}, u_{i'} v_j \in E(G)$. Spinrad, Brandstädt and Stewart [30] showed that (1) a bipartite graph is a permutation graph if and only if it has a strong ordering and that (2) in any such ordering, for every $v \in V(G)$, the vertices of $N_G(v)$ are consecutive either in σ_1 or in σ_2 . Using this and other results from [30], we prove the following theorem.

► **Theorem 19.** *There is an algorithm that, given an instance (G, T, r) of CYCLE SEGMENT COVER, where G is a bipartite permutation graph and T is not a 2-clique, finds either a cycle or a family of at most r paths that cover T or a T -separator S^* with $c_T(G - S^*) - |S^*| > r$ that certifies a no-answer in time $\mathcal{O}(n + m)$.*

To prove Theorem 19, we first establish the stronger fact that in a connected bipartite permutation graph G , if $T \subseteq V(G)$ is not cyclable (and not a 2-clique) then there is a T -separator S^* with the property $c_T(G - S^*) - |S^*| > 0$ such that S^* is formed by consecutive vertices with respect to the strong ordering of either V_1 or V_2 . We use this fact to construct a certifying algorithm for TERMINAL CYCLABILITY and then, finally, generalize this algorithm to CYCLE SEGMENT COVER. Note that, unlike for interval graphs, we cannot use an analogue of Observation 15 for bipartite permutation graphs because the class of bipartite permutation graphs is not closed under adding a universal vertex.

For k -CYCLABILITY, we first show that a connected bipartite permutation graph $G = (V_1, V_2, E)$ with at least three vertices is k -cyclable if and only if all the subsets T satisfying the following property are cyclable: $T \subseteq V_i$ for some $i \in \{1, 2\}$, the vertices of T are consecutive in σ_i and $|T| = \min\{|V_i|, k\}$. Using this equivalence, we test the k -cyclability of G by running the T -cyclability algorithm for each subset T of terminals satisfying the aforementioned property. As the number of such subsets T is $\mathcal{O}(n)$, we obtain the following theorem, which implies Theorem 2 for bipartite permutation graphs (notice the slightly better running time, namely $\mathcal{O}(nm)$ instead of $\mathcal{O}(n^3)$).

► **Theorem 20.** *k -CYCLABILITY can be solved in time $\mathcal{O}(nm)$ on bipartite permutation graphs.*

Notice that, unlike our approach for solving k -CYCLABILITY on interval graphs, we solve k -CYCLABILITY on bipartite permutation graphs G without determining $\text{sc}^k(G)$.

5 Cographs

In this section, we briefly sketch our results for CYCLE SEGMENT COVER and k -CYCLABILITY on cographs.

Let G_1 and G_2 be two vertex-disjoint graphs. The *union* operation creates the *disjoint union* $G_1 + G_2$ of G_1 and G_2 , that is, the graph with vertex set $V(G_1) \cup V(G_2)$ and edge set $E(G_1) \cup E(G_2)$. The *join* operation adds an edge between every vertex of G_1 and every vertex of G_2 . Cographs can be characterized as those graphs that can be generated from K_1 by a sequence of join and union operations. This gives each cograph G a nice tree representation, called the *cotree* of G , whose leaves are the vertices of G and whose internal nodes represent the join and union operations used in the construction of G .

Our algorithm for CYCLE SEGMENT COVER of a cograph G is built using dynamic programming bottom-up along the cotree of G .

► **Theorem 21.** *There is an algorithm that, given an instance (G, T, r) of CYCLE SEGMENT COVER where G is a cograph given by its cotree and T is not a 2-clique, finds either a T -cycle segment cover of size at most r or a T -separator with $c_T(G - S^*) - |S^*| > r$ that certifies a no-answer in $\mathcal{O}(n)$ time.*

We solve k -CYCLABILITY for cographs just as we did for interval graphs by determining all the scattering numbers $\text{sc}^k(G)$ for $k \in \{1, \dots, n\}$, again using a bottom-up dynamic programming scheme along the cotree of G .

► **Theorem 22.** *For a non-complete cograph G , the scattering numbers $\text{sc}^k(G)$ for all $k \in \{1, \dots, n\}$ can be computed and k -CYCLABILITY can be solved in time $\mathcal{O}(n^3)$.*

6 Conclusion

In summary, we design certifying linear-time algorithms to solve CYCLE SEGMENT COVER, which is a generalization of HAMILTONIAN CYCLE, for interval graphs, bipartite permutation graphs and cographs. We also use these results to show that k -CYCLABILITY as well can be solved in polynomial time when restricted to these graph classes.

A natural open question is to consider the aforementioned problems for other graph classes. In particular, what can be said about the class of cocomparability graphs (see [6, 21] for the formal definition and properties of this class)? For instance, it is proved by Deogun, Kratsch and Steiner [15] that a cocomparability graph G with at least three vertices has a Hamiltonian cycle if and only if $\text{sc}(G) \leq 0$. They also proved that the set of vertices of G can be covered by at most k vertex-disjoint paths if and only if $\text{sc}(G) \leq k$. This indicates that the class of cocomparability graphs is a natural candidate for CYCLE SEGMENT COVER and k -CYCLABILITY. Still, we do not see how to extend the results of [15] to our settings.


Another interesting question is about the complexity of TERMINAL CYCLABILITY. It is easy to see that the problem is in Π_2^P . Golovach et al. conjectured in [20] that TERMINAL CYCLABILITY is Π_2^P -complete. The conjecture is still open.

References

- 1 Eyad Alkassar, Sascha Böhme, Kurt Mehlhorn, and Christine Rizkallah. A Framework for the Verification of Certifying Computations. *J. Autom. Reasoning*, 52(3):241–273, 2014. doi:10.1007/s10817-013-9289-2.
- 2 Douglas Bauer, Hajo Broersma, and Edward F. Schmeichel. Toughness in Graphs - A Survey. *Graphs and Combinatorics*, 22(1):1–35, 2006. doi:10.1007/s00373-006-0649-0.
- 3 Andreas Björklund, Thore Husfeldt, and Nina Taslaman. Shortest cycle through specified elements. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1747–1753. SIAM, 2012. URL: <http://portal.acm.org/citation.cfm?id=2095255&CFID=63838676&CFTOKEN=79617016>, doi:10.1137/1.9781611973099.139.
- 4 Kellogg S. Booth and George S. Lueker. Testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity Using PQ-Tree Algorithms. *J. Comput. Syst. Sci.*, 13(3):335–379, 1976. doi:10.1016/S0022-0000(76)80045-1.
- 5 Andreas Brandstädt and Dieter Kratsch. On the restriction of some NP-complete graph problems to permutation graphs. In *Fundamentals of Computation Theory, FCT '85, Cottbus, GDR, September 9-13, 1985*, volume 199 of *Lecture Notes in Computer Science*, pages 53–62. Springer, 1985. doi:10.1007/BFb0028791.
- 6 Andreas Brandstadt, Van Bang Le, and Jeremy P. Spinrad. *Graph classes: a survey*. SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1999. doi:10.1137/1.9780898719796.
- 7 Hajo Broersma, Jirí Fiala, Petr A. Golovach, Tomás Kaiser, Daniël Paulusma, and Andrzej Proskurowski. Linear-Time Algorithms for Scattering Number and Hamilton-Connectivity of Interval Graphs. *Journal of Graph Theory*, 79(4):282–299, 2015. doi:10.1002/jgt.21832.
- 8 Maw-Shang Chang, Sheng-Lung Peng, and Jenn-Liang Liaw. Deferred-query: An efficient approach for some problems on interval graphs. *Networks*, 34(1):1–10, 1999. doi:10.1002/(SICI)1097-0037(199908)34:1<1::AID-NET1>3.0.CO;2-C.
- 9 Vasek Chvátal. Tough graphs and hamiltonian circuits. *Discrete Mathematics*, 5(3):215–228, 1973. doi:10.1016/0012-365X(73)90138-6.
- 10 Derek G. Corneil, Barnaby Dalton, and Michel Habib. LDFS-Based Certifying Algorithm for the Minimum Path Cover Problem on Cocomparability Graphs. *SIAM J. Comput.*, 42(3):792–807, 2013. doi:10.1137/11083856X.
- 11 Derek G. Corneil, H. Lerchs, and L. Stewart Burlingham. Complement reducible graphs. *Discrete Applied Mathematics*, 3(3):163–174, 1981. doi:10.1016/0166-218X(81)90013-5.
- 12 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.
- 13 Peter Damaschke. Paths in interval graphs and circular arc graphs. *Discrete Mathematics*, 112(1-3):49–64, 1993. doi:10.1016/0012-365X(93)90223-G.
- 14 Peter Damaschke, Jitender S. Deogun, Dieter Kratsch, and George Steiner. Finding Hamiltonian paths in cocomparability graphs using the bump number algorithm. *Order*, 8(4):383–391, 1991. doi:10.1007/BF00571188.
- 15 Jitender S. Deogun, Dieter Kratsch, and George Steiner. 1-Tough cocomparability graphs are hamiltonian. *Discrete Mathematics*, 170(1-3):99–106, 1997. doi:10.1016/0012-365X(95)00359-5.
- 16 Jitender S. Deogun and George Steiner. Polynomial Algorithms for Hamiltonian Cycle in Cocomparability Graphs. *SIAM J. Comput.*, 23(3):520–552, 1994. doi:10.1137/S0097539791200375.
- 17 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.
- 18 Gabriel Andrew Dirac. In abstrakten Graphen vorhandene vollständige 4-Graphen und ihre Unterteilungen. *Math. Nachr.*, 22:61–85, 1960. doi:10.1002/mana.19600220107.

- 19 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- 20 Petr A. Golovach, Marcin Kaminski, Spyridon Maniatis, and Dimitrios M. Thilikos. The Parameterized Complexity of Graph Cyclability. *SIAM J. Discrete Math.*, 31(1):511–541, 2017. doi:10.1137/141000014.
- 21 M. C. Golumbic. *Algorithmic graph theory and perfect graphs*, volume 57. Elsevier, 2004.
- 22 Ronald J. Gould. A look at cycles containing specified elements of a graph. *Discrete Mathematics*, 309(21):6299–6311, 2009. doi:10.1016/j.disc.2008.04.017.
- 23 Ruo-Wei Hung and Maw-Shang Chang. Linear-time algorithms for the Hamiltonian problems on distance-hereditary graphs'. *Theor. Comput. Sci.*, 341(1-3):411–440, 2005. doi:10.1016/j.tcs.2005.04.009.
- 24 Ruo-Wei Hung and Maw-Shang Chang. Linear-time certifying algorithms for the path cover and Hamiltonian cycle problems on interval graphs. *Appl. Math. Lett.*, 24(5):648–652, 2011. doi:10.1016/j.aml.2010.11.030.
- 25 H. A. Jung. On a class of posets and the corresponding comparability graphs. *J. Comb. Theory, Ser. B*, 24(2):125–133, 1978. doi:10.1016/0095-8956(78)90013-8.
- 26 J. Mark Keil. Finding Hamiltonian Circuits in Interval Graphs. *Inf. Process. Lett.*, 20(4):201–206, 1985. doi:10.1016/0020-0190(85)90050-X.
- 27 Ross M. McConnell, Kurt Mehlhorn, Stefan Näher, and Pascal Schweitzer. Certifying algorithms. *Computer Science Review*, 5(2):119–161, 2011. doi:10.1016/j.cosrev.2010.09.009.
- 28 Haiko Müller. Hamiltonian circuits in chordal bipartite graphs. *Discrete Mathematics*, 156(1-3):291–298, 1996. doi:10.1016/0012-365X(95)00057-4.
- 29 Neil Robertson and Paul D. Seymour. Graph Minors .XIII. The Disjoint Paths Problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995. doi:10.1006/jctb.1995.1006.
- 30 Jeremy P. Spinrad, Andreas Brandstädt, and Lorna Stewart. Bipartite permutation graphs. *Discrete Applied Mathematics*, 18(3):279–292, 1987. doi:10.1016/S0166-218X(87)80003-3.

Complexity of Linear Operators

Alexander S. Kulikov 

Steklov Mathematical Institute at St. Petersburg, Russian Academy of Sciences,
St. Petersburg State University, Russia
<https://logic.pdmi.ras.ru/~kulikov/>
kulikov@logic.pdmi.ras.ru

Ivan Mikhailin

University of California, San Diego, CA, USA
imikhail@eng.ucsd.edu

Andrey Mokhov

School of Engineering, Newcastle University, UK
andrey.mokhov@ncl.ac.uk

Vladimir Podolskii 

Steklov Mathematical Institute, Russian Academy of Sciences, Moscow, Russia
<http://www.mi-ras.ru/~podolskii/>
podolskii@mi-ras.ru

Abstract

Let $A \in \{0, 1\}^{n \times n}$ be a matrix with z zeroes and u ones and x be an n -dimensional vector of formal variables over a semigroup (S, \circ) . How many semigroup operations are required to compute the linear operator Ax ?

As we observe in this paper, this problem contains as a special case the well-known range queries problem and has a rich variety of applications in such areas as graph algorithms, functional programming, circuit complexity, and others. It is easy to compute Ax using $O(u)$ semigroup operations. The main question studied in this paper is: can Ax be computed using $O(z)$ semigroup operations? We prove that in general this is not possible: there exists a matrix $A \in \{0, 1\}^{n \times n}$ with exactly two zeroes in every row (hence $z = 2n$) whose complexity is $\Theta(n\alpha(n))$ where $\alpha(n)$ is the inverse Ackermann function. However, for the case when the semigroup is commutative, we give a constructive proof of an $O(z)$ upper bound. This implies that in commutative settings, complements of sparse matrices can be processed as efficiently as sparse matrices (though the corresponding algorithms are more involved). Note that this covers the cases of Boolean and tropical semirings that have numerous applications, e.g., in graph theory.

As a simple application of the presented linear-size construction, we show how to multiply two $n \times n$ matrices over an arbitrary semiring in $O(n^2)$ time if one of these matrices is a 0/1-matrix with $O(n)$ zeroes (i.e., a complement of a sparse matrix).

2012 ACM Subject Classification Theory of computation \rightarrow Streaming, sublinear and near linear time algorithms

Keywords and phrases algorithms, linear operators, commutativity, range queries, circuit complexity, lower bounds, upper bounds

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.17

Related Version The full version of the paper (containing all omitted proofs) is [9] available at <https://eccc.weizmann.ac.il/report/2019/002/>.

Funding *Alexander S. Kulikov*: The results presented in Section 3 are supported by Russian Science Foundation (18-71-10042).

Vladimir Podolskii: The results presented in Section 4 are supported by Russian Science Foundation (16-11-10252).

Acknowledgements We thank Paweł Gawrychowski for pointing us out to the paper [3]. We thank Alexey Talambutsa for fruitful discussions on the theory of semigroups.



© Alexander S. Kulikov, Ivan Mikhailin, Andrey Mokhov, and Vladimir Podolskii;
licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 17; pp. 17:1–17:12

Leibniz International Proceedings in Informatics



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

1 Introduction

1.1 Problem Statement and New Results

Let $A \in \{0, 1\}^{n \times n}$ be a matrix with z zeroes and u ones, and $x = (x_1, \dots, x_n)$ be an n -dimensional vector of formal variables over a semigroup (S, \circ) . In this paper, we study the complexity of the *linear operator* Ax , i.e., how many semigroup operations are required to compute a vector whose i -th element is

$$\sum_{1 \leq j \leq n \wedge A_{ij}=1} x_j$$

where the summation is over the semigroup operation \circ .¹ More specifically, we are interested in lower and upper bounds involving z and u . Matrices with $u = O(n)$ are usually called *sparse*, whereas matrices with $z = O(n)$ are called *complements of sparse matrices*. Computing all n outputs of Ax directly (i.e. using the above definition) takes $O(u)$ semigroup operations. The main question studied in this paper is: can Ax be computed using $O(z)$ semigroup operations? Note that it is easy to achieve $O(z)$ complexity if \circ has an inverse. Indeed, in this case Ax can be computed via subtraction: $Ax = (U - \bar{A})x = Ux - \bar{A}x$, where U is the all-ones matrix whose linear operator can be computed trivially using $O(n)$ semigroup operations, and \bar{A} is the complement of A and therefore has only $z = O(n)$ ones.

1.1.1 Commutative Case

Our first main result shows that in the commutative case, complements of sparse matrices can be processed as efficiently as sparse matrices. Specifically, we prove that if the semigroup is commutative, Ax can be computed in $O(z)$ semigroup operations; or, more formally, there exists a circuit of size $O(z)$ that uses $x = (x_1, \dots, x_n)$ as an input and computes Ax by only applying the semigroup operation \circ (we provide the formal definition of the computational model in Section 2.3). Moreover, the constructed circuits are *uniform* in the sense that they can be generated by an efficient algorithm. Hence, our circuits correspond to an elementary algorithm that uses no tricks like examining the values x_j , i.e., the semigroup operation \circ is applied in a (carefully chosen) order that is independent of the specific input x .

► **Theorem 1.** *Let (S, \circ) be a commutative semigroup, and $A \in \{0, 1\}^{n \times n}$ be a matrix with $z = \Omega(n)$ zeroes. There exists a circuit of size $O(z)$ that uses a vector $x = (x_1, \dots, x_n)$ of formal variables as an input, uses only the semigroup operation \circ at internal gates, and outputs Ax . Moreover, there exists a randomized algorithm that takes the positions of z zeroes of A as an input and outputs such a circuit in time $O(z)$ with probability at least $1 - \frac{O(\log^5 n)}{n}$. There also exists a deterministic algorithm with running time $O(z + n \log^4 n)$.*

We state the result for square matrices to simplify the presentation. Theorem 1 generalizes easily to show that Ax for a matrix $A \in \{0, 1\}^{m \times n}$ with $z = \Omega(n)$ zeroes can be computed using $O(m + z)$ semigroup operations. Also, we assume that $z = \Omega(n)$ to be able to state an upper bound $O(z)$ instead of $O(z + n)$. Note that when $z < n$, the matrix A is forced to contain all-one rows that can be computed trivially.

¹ Note that the result of summation is undefined in case of an all-zero row, because semigroups have no neutral element in general. One can trivially sidestep this technical issue by adding an all-one column $n + 1$ to the matrix A , as well as the neutral element x_{n+1} into the vector. Alternatively, we could switch from semigroups to *monoids*, but we choose not to do that, since we have no use for the neutral element and associated laws in the rest of the paper.

The following corollary generalizes Theorem 1 from vectors to matrices.

► **Corollary 2.** *Let (S, \circ) be a commutative semigroup. There exists a deterministic algorithm that takes a matrix $A \in \{0, 1\}^{n \times n}$ with $z = O(n)$ zeroes and a matrix $B \in S^{n \times n}$ and computes the product AB in time $O(n^2)$.*

1.1.2 Non-commutative Case

As our second main result, we show that *commutativity is essential*: for a faithful non-commutative semigroup S (the notion of faithful non-commutative semigroup is made formal later in the text), the minimum number of semigroup operations required to compute Ax for a matrix $A \in \{0, 1\}^{n \times n}$ with $z = O(n)$ zeroes is $\Theta(n\alpha(n))$, where $\alpha(n)$ is the inverse Ackermann function.

► **Theorem 3.** *Let (S, \circ) be a faithful non-commutative semigroup, $x = (x_1, \dots, x_n)$ be a vector of formal variables, and $A \in \{0, 1\}^{n \times n}$ be a matrix with $O(n)$ zeroes. Then Ax is computable using $O(n\alpha(n))$ semigroup operations, where $\alpha(n)$ is the inverse Ackermann function. Moreover, there exists a matrix $A \in \{0, 1\}^{n \times n}$ with exactly two zeroes in every row such that the minimum number of semigroup operations required to compute Ax is $\Omega(n\alpha(n))$.*

1.2 Motivation

The complexity of linear operators is interesting for many reasons.

Range queries. In the *range queries* problem, one is given a vector $x = (x_1, \dots, x_n)$ over a semigroup (S, \circ) and multiple queries of the form (l, r) , and is required to output the result $x_l \circ x_{l+1} \circ \dots \circ x_r$ for each query. It is a classical problem in data structures and algorithms with applications in many fields, such as bioinformatics and string algorithms, computational geometry, image analysis, real-time systems, and others. We review some of the less straightforward applications as well as a rich variety of algorithmic techniques for the problem in the full version of the paper [9].

The linear operator problem is a natural generalization of the range queries problem: each row of the matrix A defines a subset of the elements of x that need to be summed up and this subset is not required to be a contiguous range. The algorithms (Theorem 1 and Corollary 2) and hardness results (Theorem 3) for the linear operator problem presented in this paper are indeed inspired by some of the known results for the range queries problem.

Graph algorithms. Various graph path/reachability problems can be reduced naturally to matrix multiplication. Two classic examples are: (i) the all-pairs shortest path problem (APSP) is reducible to min-plus matrix multiplication, and (ii) the number of triangles in an undirected graph can be found by computing the third power of its adjacency matrix. It is natural to ask what happens if a graph has $O(n)$ edges or $O(n)$ anti-edges (as usual, by n we denote the number of nodes). In many cases, an efficient algorithm for sparse graphs ($O(n)$ edges) is straightforward whereas an algorithm with the same efficiency for complements of sparse graphs ($O(n)$ anti-edges) is not. For example, it is easy to solve APSP and triangle counting on sparse graphs in time $O(n^2)$, but achieving the same time complexity for complements of sparse graphs is more complicated. Theorem 1 and Corollary 2 give a black-box way to solve these two problems on complements of sparse graphs in time $O(n^2)$.

Matrix multiplication over semirings. Fast matrix multiplication methods rely essentially on the ring structure of the underlying set of elements. The first such algorithm was given by Strassen, the current record upper bound is $O(n^{2.373})$ [13, 4]. The removal of the inverse operation often drastically increases the complexity of algorithmic problems

over algebraic structures, and even the complexity of standard computational tasks are not well understood over tropical and Boolean semirings (see, e.g. [12, 6]). For various important semirings, we still do not know an $n^{3-\varepsilon}$ (for a constant $\varepsilon > 0$) upper bound for matrix multiplication, e.g., the strongest known upper bound for min-plus matrix multiplication is $n^3 / \exp(\sqrt{\log n})$ [12].

The interest in computations over such algebraic structures has recently grown substantially throughout the Computer Science community with the cases of Boolean and tropical semirings being of the main interest (see, for example, [8, 12, 2]). From this perspective, the computation complexity over sparse and complements of sparse matrices is one of the most basic questions. Theorem 1 and Corollary 2 therefore characterise natural special cases when efficient computations are possible.

Functional programming. Algebraic data structures for graphs developed in the functional programming community [10] can be used for representing and processing densely-connected graphs in linear (in the number of vertices) time and memory. As we discuss in the full version of the paper [9], Theorem 1 yields an algorithm for deriving a linear-size algebraic graph representation for complements of sparse graphs.

Circuit complexity. Computing linear operators over a Boolean semiring $(\{0, 1\}, \vee)$ is a well-studied problem in circuit complexity. The corresponding computational model is known as *rectifier networks*. An overview of known lower and upper bounds for such circuits is given by Jukna [7, Section 13.6]. Theorem 1 states that very dense linear operators have linear rectifier network complexity.

1.3 Organization

The remaining part of the paper is organized as follows. In Section 2 we introduce necessary definitions. In Section 3 we present the results on commutative case. In Section 4 we present the results on the non-commutative case. Due to the space constraints many proofs are omitted. They can be found in the full version of the paper [9].

2 Background

2.1 Semigroups and Semirings

A *semigroup* (S, \circ) is an algebraic structure, where the operation \circ is *closed*, i.e., $\circ : S \times S \rightarrow S$, and *associative*, i.e., $x \circ (y \circ z) = (x \circ y) \circ z$ for all x, y , and z in S . *Commutative* (or *abelian*) semigroups introduce one extra requirement: $x \circ y = y \circ x$ for all x and y in S .

A commutative semigroup (S, \circ) can often be extended to a *semiring* (S, \circ, \bullet) by introducing another associative (but not necessarily commutative) operation \bullet that *distributes* over \circ , that is

$$x \bullet (y \circ z) = (x \bullet y) \circ (x \bullet z)$$

$$(x \circ y) \bullet z = (x \bullet z) \circ (y \bullet z)$$

hold for all x, y , and z in S . Since \circ and \bullet behave similarly to numeric addition and multiplication, it is common to give \bullet a higher precedence to avoid unnecessary parentheses, and even omit \bullet from formulas altogether, replacing it by juxtaposition. This gives a terser and more convenient notation, e.g., the left distributivity law becomes: $x(y \circ z) = xy \circ xz$. We will use this notation, insofar as this does not lead to ambiguity. See the full version of the paper [9] for an overview of commonly used semigroups and semirings.

2.2 Range Queries Problem and Linear Operator Problem

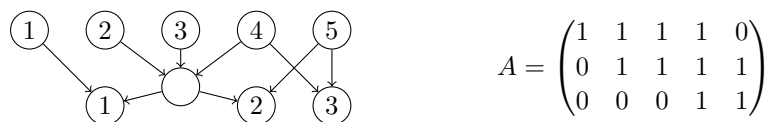
In the *range queries problem*, one is given a sequence x_1, x_2, \dots, x_n of elements of a fixed semigroup (S, \circ) . Then, a *range query* is specified by a pair of indices (l, r) , such that $1 \leq l \leq r \leq n$. The answer to such a query is the result of applying the semigroup operation to the corresponding range, i.e., $x_l \circ x_{l+1} \circ \dots \circ x_r$. The range queries problem is then to simply answer all given range queries. There are two regimes: online and offline. In the *online regime*, one is given a sequence of *values* $x_1 = v_1, x_2 = v_2, \dots, x_n = v_n$ and is asked to preprocess it so that to answer efficiently any subsequent query. By “efficiently” one usually means in time independent of the length of the range (i.e., $r - l + 1$, the time of a naive algorithm), say, in time $O(\log n)$ or $O(1)$. In this paper, we focus on the *offline* version, where one is given a sequence together with all the queries, and are interested in the minimum number of semigroup operations needed to answer all the queries. Moreover, we study a more general problem: we assume that x_1, \dots, x_n are formal variables rather than actual semigroup values. That is, we study the *circuit size* of the corresponding computational problem.

The *linear operator* problem generalizes the range queries problem: now, instead of contiguous ranges one wants to compute sums over arbitrary subsets. These subsets are given as rows of a 0/1-matrix A .

2.3 Circuits

We assume that the input consists of n formal variables $\{x_1, \dots, x_n\}$. We are interested in the minimum number of semigroup operations needed to compute all given words $\{w_1, \dots, w_m\}$ (e.g., for the range queries problem, each word has a form $x_l \circ x_{l+1} \circ \dots \circ x_r$). We use the following natural *circuit* model. A circuit computing all these queries is a directed acyclic graph. There are exactly n nodes of zero in-degree. They are labelled with $\{1, \dots, n\}$ and are called *input gates*. All other nodes have positive in-degree and are called *gates*. Finally, some m gates have out-degree 0 and are labelled with $\{1, \dots, m\}$; they are called *output gates*. The *size* of a circuit is its number of edges (also called *wires*). Each gate of a circuit computes a word defined in a natural way: input gates compute just $\{x_1, \dots, x_n\}$; any other gate of in-degree r computes a word $f_1 \circ f_2 \circ \dots \circ f_r$ where $\{f_1, \dots, f_r\}$ are words computed at its predecessors (therefore, we assume that there is an underlying order on the incoming wires for each gate). We say that the circuit computes the words $\{w_1, \dots, w_m\}$ if the words computed at the output gates are equivalent to $\{w_1, \dots, w_m\}$ over the considered semigroup.

For example, the following circuit computes range queries $(l_1, r_1) = (1, 4)$, $(l_2, r_2) = (2, 5)$, and $(l_3, r_3) = (4, 5)$ over inputs $\{x_1, \dots, x_5\}$ or, equivalently, the linear operator Ax where the matrix A is given below.



For a 0/1-matrix A , by $C(A)$ we denote the minimum size of a circuit computing the linear operator Ax .

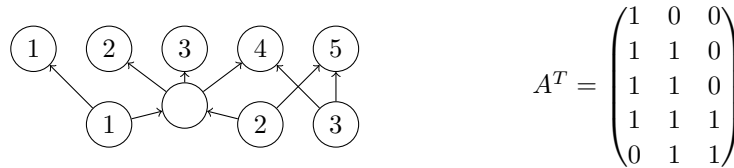
A *binary circuit* is a circuit having no gates of fan-in more than two. It is not difficult to see that any circuit can be converted into a binary circuit of size at most twice the size of the original circuit. For this, one just replaces every gate of fan-in k , for $k > 2$, by a binary tree with $2k - 2$ wires (such a tree contains k leaves hence $k - 1$ inner nodes and $2k - 2$ edges). In the binary circuit the number of gates does not exceed its size (i.e., the number of wires). And the number of gates in a binary circuit is exactly the minimum number of semigroup operations needed to compute the corresponding function.

17:6 Complexity of Linear Operators

We call a circuit C computing A *regular* if for every pair (i, j) such that $A_{ij} = 1$, there exists exactly one path from the input j to the output i . A convenient property of regular circuits is the following observation.

► **Observation 1.** *Let C be a regular circuit computing a 0/1-matrix A over a commutative semigroup. Then, by reversing all the wires in C one gets a circuit computing A^T .*

Instead of giving a formal proof, we provide an example of a reversed circuit from the example given above. It is because of this observation that we require circuit outputs to be gates of out-degree zero (so that when reversing all the wires the inputs and the outputs exchange places).



3 Commutative Case

This section is devoted to the proofs of Theorem 1 and Corollary 2. We start by proving two simpler statements to show how commutativity is important.

► **Lemma 4.** *Let S be a semigroup (not necessarily commutative) and let $A \in \{0, 1\}^{n \times n}$ contain at most one zero in every row. Then $C(A) = O(n)$.*

Proof. To compute the linear operator Ax , we first precompute all prefixes and suffixes of $x = (x_1, \dots, x_n)$. Concretely, let $p_i = x_1 \circ x_2 \circ \dots \circ x_i$. All p_i 's can be computed using $(n-1)$ binary gates as follows:

$$p_1 = x_1, p_2 = p_1 \circ x_2, p_3 = p_2 \circ x_3, \dots, p_i = p_{i-1} \circ x_i, \dots, p_n = p_{n-1} \circ x_n.$$

Similarly, we compute all suffixes $s_j = x_j \circ x_{j+1} \circ \dots \circ x_n$ using $(n-1)$ binary gates. From these prefixes and suffixes all outputs can be computed as follows: if a row of A contains no zeroes, the corresponding output is p_n ; otherwise if a row contains a zero at position i , the output is $p_{i-1} \circ s_{i+1}$ (for $i = 1$ and $i = n$, we omit the redundant term). ◀

In the rest of the section, we assume that the underlying semigroup is commutative. Allowing at most two zeroes per row already leads to a non-trivial problem. We give only a sketch of the solution below, since we will further prove a more general result. It is interesting to compare the following lemma with Theorem 3 that states that in the non-commutative setting matrices with two zeroes per row are already hard.

► **Lemma 5.** *Let $A \in \{0, 1\}^{n \times n}$ contain at most two zeroes in every row. Then $C(A) = O(n)$.*

Proof sketch. Consider the following undirected graph: the set of nodes is $\{1, 2, \dots, n\}$; two nodes i and j are joined by an edge if there is a row having zeroes in columns i and j . In the worst case (all rows are different and contain exactly two zeroes), the graph has exactly n edges and hence it contains a cut (L, R) of size at least $n/2$. This cut splits the columns of the matrix into two parts (L and R). Now let us also split the rows into two parts: the top part T contains all columns that have exactly one zero in each L and R ; the bottom part B

contains all the remaining rows. What is nice about the top part of the matrix ($T \times (L \cup R)$) is that it can be computed by $O(n)$ gates (using Lemma 4). For the bottom part, let us cut all-1 columns out of it and make a recursive call (note that this requires the commutativity). The corresponding recurrence relation is $T(n) \leq cn + T(n/2)$ for a fixed constant c , implying $T(n) = O(n)$, and hence $C(A) = O(n)$. ◀

We now state a few auxiliary lemmas that will be used as building blocks in the proof of Theorem 1.

▶ **Lemma 6.** *There exists a binary regular circuit of size $O(n \log n)$ such that any range can be computed in a single additional binary gate using two gates of the circuit. It can be generated in time $O(n \log n)$.*

▶ **Lemma 7.** *There exists a binary regular circuit of size $O(n)$ such that any range of length at least $\log n$ can be computed in two binary additional gates from the gates of the circuit. It can be generated by an algorithm in time $O(n)$.*

▶ **Lemma 8.** *Let $m \leq n$ and $A \in \{0, 1\}^{m \times n}$ be a matrix with $z = \Omega(n)$ zeroes and at most $\log n$ zeroes in every row. There exists a circuit of size $O(z)$ computing Ax . Moreover, there exists a randomized $O(z)$ time algorithm that takes as input the positions of z zeros and outputs a circuit computing Ax with probability at least $1 - \frac{O(\log^5 n)}{n}$. There also exists a deterministic algorithm with running time $O(n \log^4 n)$.*

Proof of Theorem 1. Denote the set of rows and the set of columns of A by R and C , respectively. Let $R_0 \subseteq R$ be all the rows having at least $\log n$ zeroes and $R_1 = R \setminus R_0$. Every row of A can be decomposed into (maximal) contiguous ranges of ones. We will call them simply ranges of A . We will compute all of them. From these ranges, it takes $O(z)$ additional binary gates to compute all the outputs.

We compute the matrices $R_0 \times C$ and $R_1 \times C$ separately. The main idea is that $R_0 \times C$ is easy to compute because it has a small number of rows (at most $z/\log n$), while $R_1 \times C$ is easy to compute because it has a small number of zeroes in every row (at most $\log n$).

The matrix $R_1 \times C$ can be computed using Lemma 8. To compute $R_0 \times C$, it suffices to compute $C \times R_0$ by a regular circuit, thanks to the Observation 1. Let $|R_0| = t$. Clearly, $t \leq z/\log n$. Using Lemma 6, one can compute all ranges of $C \times R_0$ by a circuit of size

$$O(t \log t + z) = O\left(\frac{z}{\log n} \cdot \log z + z\right) = O(z + n) = O(z),$$

since $z = O(n^2)$.

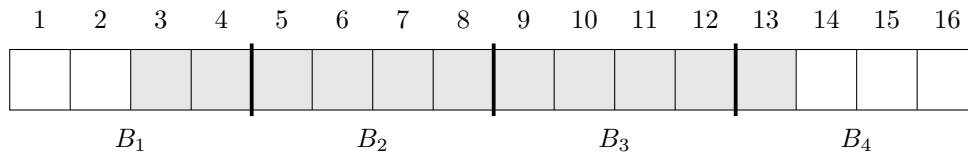
The algorithm for generating the circuit is just a combination of the algorithms from Lemmas 6 and 8. ◀

Proof of Lemma 6. We adopt the divide-and-conquer construction by Alon and Schieber [1]. Split the input range $(1, n)$ into two half-ranges of length $n/2$: $(1, n/2)$ and $(n/2 + 1, n)$. Compute all suffixes of the left half and all prefixes of the right half. Using these precomputed suffixes and prefixes one can answer any query (l, r) such that $l \leq n/2 \leq r$ in a single additional gate. It remains to be able to answer queries that lie entirely in one of the halves. We do this by constructing recursively circuits for both halves. The resulting recurrence relation $T(n) \leq 2T(n/2) + O(n)$ implies that the resulting circuit has size at most $O(n \log n)$. ◀

17:8 Complexity of Linear Operators

Proof of Lemma 7. We use the block decomposition technique for constructing the required circuit. Partition the input range $(1, n)$ into $n/\log n$ ranges of length $\log n$ and call them blocks. Compute the range corresponding to each block (in total size $O(n)$). Build a circuit from Lemma 6 on top of these blocks. The size of this circuit is $O(n)$ since the number of blocks is $n/\log n$. Compute all prefixes and all suffixes of every block. Since the blocks partition the input range $(1, n)$, this also can be done with an $O(n)$ size circuit.

Consider any range of length at least $\log n$. Note that it cannot lie entirely inside the block. Hence, any such range can be decomposed into three subranges: a suffix of a block, a range of blocks, and a prefix of a block (where any of the three components may be empty). For example, for $n = 16$, a range $(3, 13)$ is decomposed into a suffix $(3, 4)$ of the first block, a range $(2, 3)$ of blocks (B_1, B_2, B_3, B_4) , and a prefix $(13, 13)$ of the last block:



It remains to note that all these three components are already precomputed. ◀

Proof of Lemma 8. All the z zeroes of A break its rows into ranges. Let us call a range *short* if its length is at most $\log n$. We will show that it is possible to permute the columns of A so that the total length of all short ranges is at most $O(\frac{n}{\log n})$. Then, all such short ranges can be computed by a circuit of size $O(\frac{\log n}{n} \cdot n) = O(n) = O(z)$. All the remaining ranges can be computed by a circuit of size $O(n)$ using Lemma 7.

It is easy to construct the required permutation randomly. For this, one just estimates the expected total length of all short ranges in a random permutation. It is then possible to derandomize this approach using a greedy algorithm. We provide all formal details the full version of the paper [9]. ◀

Proof of Corollary 2. One deterministically generates a circuit for A of size $O(n)$ in time $O(n \log^4 n) = O(n^2)$ by Theorem 1. This circuit can be used to multiply A by any column of B in time $O(n)$. For this, one constructs a topological ordering of the gates of the circuits and computes the values of all gates in this order. Hence, AB can be computed in time $O(n^2)$. ◀

4 Non-commutative Case

In the previous section, we have shown that for commutative semigroups dense linear operators can be computed by linear size circuits. A closer look at the circuit constructions reveals that we use commutativity crucially: it is important that we may reorder the columns of the matrix (we do this in the proof of Lemma 8). In this section, we show that this trick is unavoidable: for non-commutative semigroups, it is not possible to construct linear size circuits for dense linear operators. Namely, we prove Theorem 3.

► **Theorem 3.** *Let (S, \circ) be a faithful non-commutative semigroup, $x = (x_1, \dots, x_n)$ be a vector of formal variables, and $A \in \{0, 1\}^{n \times n}$ be a matrix with $O(n)$ zeroes. Then Ax is computable using $O(n\alpha(n))$ semigroup operations, where $\alpha(n)$ is the inverse Ackermann function. Moreover, there exists a matrix $A \in \{0, 1\}^{n \times n}$ with exactly two zeroes in every row such that the minimum number of semigroup operations required to compute Ax is $\Omega(n\alpha(n))$.*

4.1 Faithful semigroups

We consider computations over general semigroups that are not necessarily commutative. In particular, we will establish lower bounds for a large class of semigroups and our lower bound does not hold for commutative semigroups. This requires a formal definition that captures semigroups with rich enough structure and in particular requires that a semigroup is substantially non-commutative.

Previously lower bounds in the circuit model for a large class of semigroups were known for the Range Queries problem [14, 3]. These result are proven for a large class of commutative semigroups that are called *faithful* (we provide a formal definition below). Since we are dealing with non-commutative case we need to generalize the notion of faithfulness to non-commutative semigroups.

To provide formal definition of faithfulness it is convenient to introduce the following notation. Suppose (S, \circ) is a semigroup. Let $X_{S,n}$ be a semigroup with generators $\{x_1, \dots, x_n\}$ and with the equivalence relation consisting of identities in variables $\{x_1, \dots, x_n\}$ over (S, \circ) . That is, for two words W and W' in the alphabet $\{x_1, \dots, x_n\}$ we have $W \sim W'$ in $X_{S,n}$ iff no matter which elements of the semigroup S we substitute for $\{x_1, \dots, x_n\}$ we obtain a correct equation over S . In particular, note that if S is commutative (respectively, idempotent), then $X_{S,n}$ is also commutative (respectively, idempotent). The semigroup $X_{S,n}$ is studied in algebra under the name of relatively free semigroup of rank n of a variety generated by semigroup S [11]. We will often omit the subscript n and write simply X_S since the number of generators will be clear from the context.

Below we will use the following notation. Let W be a word in the alphabet $\{x_1, \dots, x_n\}$. Denote by $\text{Var}(W)$ the set of letters that are present in W .

We are now ready to introduce the definition of a commutative faithful semigroup.

► **Definition 9** ([14, 3]). *A commutative semigroup (S, \circ) is faithful commutative if for any equivalence $W \sim W'$ in X_S we have $\text{Var}(W) = \text{Var}(W')$.*

Note that this definition does not pose any restrictions on the cardinality of each letter in W and W' . This allows to capture in this definition important cases of idempotent semigroups. For example, semigroups $(\{0, 1\}, \vee)$ and (\mathbb{Z}, \min) are commutative faithful.

We need to study the non-commutative case, and moreover, our results establish the difference between commutative and non-commutative cases. Thus, we need to extend the notion of faithfulness to non-commutative semigroups to capture their non-commutativity in the whole power. At the same time we would like to keep the case of idempotency. We introduce the notion of faithfulness for the non-commutative case inspired by the properties of free idempotent semigroups [5]. To introduce this notion we need several definitions.

The *initial mark* of W is the letter that is present in W such that its first appearance is farthest to the right. Let U be the prefix of W consisting of letters preceding the initial mark. That is, U is the maximal prefix of W with a smaller number of generators. We call U the *initial* of W . Analogously we define the *terminal mark* of W and the *terminal* of W .

► **Definition 10.** *We say that a semigroup X with generators $\{x_1, \dots, x_n\}$ is strongly non-commutative if for any words W and W' in the alphabet $\{x_1, \dots, x_n\}$ the equivalence $W \sim W'$ holds in X only if the initial marks of W and W' are the same, terminal marks are the same, the equivalence $U \sim U'$ holds in X , where U and U' are the initials of W and W' , respectively, and the equivalence $V \sim V'$ holds in X , where V and V' are the terminals of W and W' , respectively.*

17:10 Complexity of Linear Operators

In other words, this definition states that the first and the last occurrences of generators in the equivalence separates the parts of the equivalence that cannot be affected by the rest of the generators and must therefore be equivalent themselves. We also note that this definition exactly captures the idempotent case: for a free idempotent semigroup the condition in this definition is “if and only if”[5].

► **Definition 11.** *A semigroup (S, \circ) is faithful non-commutative if X_S is strongly non-commutative.*

We note that this notion of faithfulness is relatively general and is true for semigroups (S, \circ) with considerable degree of non-commutativity in their structure. It clearly captures free semigroups with at least two generators. It is also easy to see that the requirements in Definition 11 are satisfied for the free idempotent semigroup with n generators (if S is idempotent, then $X_{S,n}$ is also clearly idempotent and no other relations are holding in $X_{S,n}$ since we can substitute generators of S for x_1, \dots, x_n).

Next we observe some properties of strongly non-commutative semigroups that we need in our constructions.

► **Lemma 12.** *Suppose X is strongly non-commutative. Suppose the equivalence $W \sim W'$ holds in X and $|\text{Var}(W)| = |\text{Var}(W')| = k$. Suppose U and U' are minimal (maximal) prefixes of W and W' such that $|\text{Var}(U)| = |\text{Var}(U')| = l \leq k$. Then the equivalence $U \sim U'$ holds in X . The same is true for suffixes.*

Proof. The proof is by induction on the decreasing l . Consider the maximal prefixes first. For $l = k$ and maximal prefixes we just have $U = W$ and $U' = W'$. Suppose the statement is true for some l , and denote the corresponding prefixes by U and U' , respectively. Then note that the maximal prefixes with $l - 1$ variables are initials of U and U' . And the statement follows by Definition 10.

The proof of the statement for minimal prefixes is completely analogous. Note that on the step of induction the prefixes differ from the previous case by one letter that are initial marks of the corresponding prefixes. So these additional letters are also equal by the Definition 10.

The case of suffixes is completely analogous. ◀

The next lemma is a simple corollary of Lemma 12.

► **Lemma 13.** *Suppose X is strongly non-commutative. Suppose $W \sim W'$ holds in X . Let us write down the letters of W in the order in which they appear first time in W when we read it from left to right. Let's do the same for W' . Then we obtain exactly the same sequences of letters. The same is true if we read the words from right to left.*

4.2 Proof Strategy

We now proceed to the proof of Theorem 3. The upper bound follows easily by a naive algorithm: split all rows of A into ranges, compute all ranges by a circuit of size $O(n\alpha(n))$ using Yao's construction [14], then combine ranges into rows of A using $O(n)$ gates.

Thus, we focus on lower bounds. We will view the computation of the circuit as a computation in a strongly non-commutative semigroup $X = X_S$.

We will use the following proof strategy. First we observe that it is enough to prove the lower bound for the case of idempotent strongly non-commutative semigroups X . Indeed, if X is not idempotent, we can factorize it by idempotency relations and obtain a strongly non-commutative idempotent semigroup X_{id} . A lower bound for the case of X_{id} implies lower bound for the case of X . We provide a detailed explanation in the full version of the paper [9].

Hence, from this point we can assume that X is idempotent and strongly non-commutative. Next for idempotent case we show that our problem is equivalent to the commutative version of the range query problem.

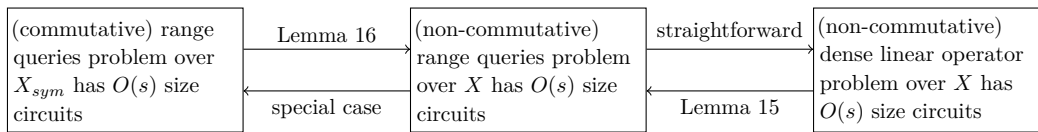
For a semigroup X with generators $\{x_1, \dots, x_n\}$ denote by X_{sym} its factorization under commutativity relations $x_i x_j \sim x_j x_i$ for all i, j . Note that if X is idempotent and strongly non-commutative, then X_{sym} is just the semigroup in which $W \sim W'$ iff $\text{Var}(W) = \text{Var}(W')$ (this is free idempotent commutative semigroup).

► **Theorem 14.** *For an idempotent strongly non-commutative X and for any $s = \Omega(n)$ we have that (commutative) range queries problem over X_{sym} has size $O(s)$ circuits iff (non-commutative) dense linear operator problem over X has size $O(s)$ circuits.*

Using this theorem, it is straightforward to finish the proof of Theorem 3. Indeed, by Theorem 14 if non-commutative dense linear operator problem has size s circuit, then the commutative range queries problem also does. However, for the latter problem it is proved by Chazelle and Rosenberg [3] that $s = \Omega(n\alpha(n))$. Moreover, in our construction for the proof of Theorem 14 it is enough to consider dense linear operators with exactly two zeroes in every row. From this the second part of Theorem 3 follows.

Note that for the proof of Theorem 3 only one direction of Theorem 14 is needed. However, we think that the equivalence in Theorem 14 might be of independent interest, so we provide the proof for both directions.

Thus, it remains to prove Theorem 14. We do this by showing the following equivalences for any $s = \Omega(n)$.



Note that two of the reductions on this diagram are trivial. The other two are formulated in the following lemmas.

► **Lemma 15.** *If the (non-commutative) dense linear operator problem over X has size s circuit then the (non-commutative) range queries problem over X has size $O(s)$ circuit.*

► **Lemma 16.** *If the (commutative) version of the range queries problem over X_{sym} has size s circuits then the (non-commutative) version over X also does.*

5 Open Problems

There are two natural problems left open.

1. Design a deterministic $O(z)$ time algorithm for generating a circuit in the commutative case. For this, it suffices to design an $O(n)$ deterministic algorithm for the following problem: given a list of positions of n zeroes of an $n \times n$ 0/1-matrix with at most $\log n$ zeroes in every row, permute its columns so that the total length of all segments of length at most $O(\log n)$ is $O(\frac{n}{\log n})$.
2. Determine the asymptotic complexity of the linear operator in terms of the number of zeroes in the non-commutative case.

References

- 1 Noga Alon and Baruch Schieber. Optimal preprocessing for answering on-line product queries. Technical report, Tel Aviv University, 1987.
- 2 Peter Butkovič. *Max-linear Systems: Theory and Algorithms*. Springer, 2010.
- 3 Bernard Chazelle and Burton Rosenberg. The complexity of computing partial sums off-line. *Int. J. Comput. Geometry Appl.*, 1(1):33–45, 1991. doi:10.1142/S0218195991000049.
- 4 François Le Gall. Powers of tensors and fast matrix multiplication. In Katsusuke Nabeshima, Kosaku Nagasaka, Franz Winkler, and Ágnes Szántó, editors, *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*, pages 296–303. ACM, 2014. doi:10.1145/2608628.2608664.
- 5 J. A. Green and D. Rees. On semi-groups in which $x^r = x$. *Mathematical Proceedings of the Cambridge Philosophical Society*, 48(1):35–40, 1952. doi:10.1017/S0305004100027341.
- 6 Dima Grigoriev and Vladimir V. Podolskii. Complexity of Tropical and Min-plus Linear Pre-varieties. *Computational Complexity*, 24(1):31–64, 2015. doi:10.1007/s00037-013-0077-5.
- 7 Stasys Jukna. *Boolean Function Complexity - Advances and Frontiers*, volume 27 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-24508-4.
- 8 Stasys Jukna. Tropical Complexity, Sidon Sets, and Dynamic Programming. *SIAM J. Discrete Math.*, 30(4):2064–2085, 2016. doi:10.1137/16M1064738.
- 9 Alexander S. Kulikov, Ivan Mikhailin, Andrey Mokhov, and Vladimir V. Podolskii. Complexity of Linear Operators. *Electronic Colloquium on Computational Complexity (ECCC)*, 26:2, 2019. URL: <https://eccc.weizmann.ac.il/report/2019/002>.
- 10 Andrey Mokhov. Algebraic graphs with class (functional pearl). In *Proceedings of the 10th ACM SIGPLAN International Symposium on Haskell*, pages 2–13. ACM, 2017.
- 11 H. Neumann. *Varieties of Groups*. Ergebnisse der Mathematik und ihrer Grenzgebiete. 2. Folge. Springer Berlin Heidelberg, 2012. URL: <https://books.google.ru/books?id=VaMjCQAAQBAJ>.
- 12 Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 664–673, 2014. doi:10.1145/2591796.2591811.
- 13 Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith–Winograd. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 887–898. ACM, 2012. doi:10.1145/2213977.2214056.
- 14 Andrew Chi-Chih Yao. Space-Time Tradeoff for Answering Range Queries (Extended Abstract). In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, editors, *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 128–136. ACM, 1982. doi:10.1145/800070.802185.

New Results for the k -Secretary Problem

Susanne Albers

Department of Informatics, Technical University of Munich, Germany
albers@in.tum.de

Leon Ladewig

Department of Informatics, Technical University of Munich, Germany
ladewig@in.tum.de

Abstract

Suppose that n numbers arrive online in random order and the goal is to select k of them such that the expected sum of the selected items is maximized. The decision for any item is irrevocable and must be made on arrival without knowing future items. This problem is known as the k -secretary problem, which includes the classical secretary problem with the special case $k = 1$. It is well-known that the latter problem can be solved by a simple algorithm of competitive ratio $1/e$ which is asymptotically optimal. When k is small, only for $k = 2$ does there exist an algorithm beating the threshold of $1/e$ [Chan et al. SODA 2015]. The algorithm relies on an involved selection policy. Moreover, there exist results when k is large [Kleinberg SODA 2005].

In this paper we present results for the k -secretary problem, considering the interesting and relevant case that k is small. We focus on simple selection algorithms, accompanied by combinatorial analyses. As a main contribution we propose a natural deterministic algorithm designed to have competitive ratios strictly greater than $1/e$ for small $k \geq 2$. This algorithm is hardly more complex than the elegant strategy for the classical secretary problem, optimal for $k = 1$, and works for all $k \geq 1$. We explicitly compute its competitive ratios for $2 \leq k \leq 100$, ranging from 0.41 for $k = 2$ to 0.75 for $k = 100$. Moreover, we show that an algorithm proposed by Babaioff et al. [APPROX 2007] has a competitive ratio of 0.4168 for $k = 2$, implying that the previous analysis was not tight. Our analysis reveals a surprising combinatorial property of this algorithm, which might be helpful for a tight analysis of this algorithm for general k .

2012 ACM Subject Classification Theory of computation \rightarrow Online algorithms

Keywords and phrases Online algorithms, secretary problem, random order model

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.18

Funding Work supported by the European Research Council, Grant Agreement No. 691672.

1 Introduction

The *secretary problem* is a well-known problem in the field of optimal stopping theory and is defined as follows: Given a sequence of n numbers which arrive online and in random order, select the maximum number. Thereby, upon arrival of an item, the decision to accept or reject it must be made immediately and irrevocably, especially without knowing future items. The statement of the problem dates back to the 1960s and its solution is due to Lindley [23] and Dynkin [10]. For discussions on the origin of the problem, we refer to the survey [13].

In the past years, generalizations of the secretary problem involving selection of multiple items have become very popular. We consider one of the most canonical generalizations known as the k -secretary problem: The algorithm is allowed to choose k elements and the goal is to maximize the expected sum of accepted elements. Other objective functions, such as maximizing the probability of accepting the k best [2, 14] or general submodular functions [20], have been studied as well. Maximizing the sum of accepted items is closely related to the *knapsack secretary problem* [3, 19]. If all items have unit weight and thus the knapsack capacity is a cardinality bound, the k -secretary problem arises. The *matroid*



© Susanne Albers and Leon Ladewig;

licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 18; pp. 18:1–18:19

Leibniz International Proceedings in Informatics



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

secretary problem, introduced by Babaioff et al. [6], is a generalization where an algorithm must maintain a set of accepted items that form an independent set of a given matroid. We refer the reader to [11, 12, 22] for recent work. If the matroid is k -uniform, again, the k -secretary problem occurs. Another closely related problem was introduced by Buchbinder, Jain, and Singh [8]. In the (J, K) -secretary problem, an algorithm has J choices and the objective is to maximize the number of selected items among the K best. Assuming the ordinal model [17] and a monotonicity property of the algorithm, any c -competitive algorithm for the (k, k) -secretary problem is c -competitive for the k -secretary problem, and vice versa [8]. In the ordinal model [17], an algorithm decides based on the total order of items only, rather than on their numeric values. In fact, most known and elegant algorithms for the k -secretary problem assume the ordinal model [3, 10, 21, 23].

The large interest in generalizations of the classical secretary problem is motivated mainly by numerous applications in online market design [4, 6, 21]. Apart from these applications, the secretary problem is the prototype of an online problem analyzed in the random order model: An adversarial input order often rules out (good) competitive ratios when considering online optimization problems without further constraints. By contrast, the assumption that the input is ordered randomly improves the competitive ratios in many optimization problems. This includes packing problems [18, 19], scheduling problems [15], and graph problems [7, 24]. Therefore, developing new techniques for secretary problems may, more generally, yield relevant insights for the analysis of online problems in randomized input models as well.

1.1 Previous Work

The k -secretary problem was introduced by Kleinberg [21] in 2005. He presents a randomized algorithm attaining a competitive ratio of $1 - 5/\sqrt{k}$, which approaches 1 for $k \rightarrow \infty$. Moreover, Kleinberg gives in [21] a hardness result stating that any algorithm has a competitive ratio of $1 - \Omega(\sqrt{1/k})$. Therefore, from an asymptotic point of view, the k -secretary problem is solved by Kleinberg's result. However, the main drawback can be seen in the fact that the competitive ratio is not defined if $k \leq 24$ and breaks the barrier of $1/e$ only if $k \geq 63$ (see Figure 2, p. 11).

In 2007 the problem was revisited by Babaioff et al. [3]. The authors propose two algorithms called VIRTUAL and OPTIMISTIC and prove that both algorithms have a competitive ratio of at least $1/e$ for any k . While the analysis of VIRTUAL is simple and tight, it takes much more effort to analyze OPTIMISTIC [3, 4]. The authors believe that their analysis for OPTIMISTIC is not tight for $k \geq 2$.

Buchbinder, Jain, and Singh [8] developed a framework to analyze secretary problems and their optimal algorithms using linear programming techniques. By numerical simulations for the (k, k) -secretary problem with $n = 100$, Buchbinder et al. obtained competitive ratios of 0.474, 0.565, and 0.612, for $k = 2, 3$, and 4, respectively. However, obtaining an algorithm from their framework requires a formal analysis of the corresponding LP in the limit of $n \rightarrow \infty$, which is not provided in the article [8, p. 192].

Chan, Chen, and Jiang [9] revisited the (J, K) -secretary problem and obtained several fundamental results. Notably, they showed that optimal algorithms for the k -secretary problem require access to the numeric values of the items, which complements the previous line of research in the ordinal model. Chan et al. demonstrate this by providing a 0.4920-competitive algorithm for the 2-secretary problem which is based on a 0.4886-competitive algorithm for the $(2, 2)$ -secretary problem. Still, an analysis for the general (J, K) -case is not known, even for $J = K$. Moreover, the resulting algorithms seem overly involved. This dims the prospect of elegant k -secretary algorithms for $k \geq 3$ obtained from this approach.

■ **Table 1** Competitive ratios α of SINGLE-REF for $k \in [1..20]$.

k	1	2	3	4	5	6	7	8	9	10
α	$1/e$	0.4119	0.4449	0.4785	0.4999	0.5148	0.5308	0.5453	0.5567	0.5660
k	11	12	13	14	15	16	17	18	19	20
α	0.5740	0.5834	0.5914	0.5983	0.6043	0.6096	0.6155	0.6211	0.6261	0.6306

1.2 Our Contribution

We study the k -secretary problem, the most natural and immediate generalization of the classical secretary problem. While the extreme cases $k = 1$ and $k \rightarrow \infty$ are well studied, hardly any results for small values of $k \geq 2$ exist. We believe that simple selection algorithms, performing well for small k , are interesting both from a theoretical point of view and for practical settings. Moreover, the hope is that existing algorithms for related problems based on k -secretary algorithms can be improved this way [8, p. 191]. We study algorithms designed for the ordinal model, which guarantees robustness and plainer decision rules.

For this purpose, we propose a simple deterministic algorithm SINGLE-REF. This algorithm uses a single value as threshold for accepting items. Although similar approaches based on this natural idea have been used to solve related problems [1], to the best of our knowledge, this algorithm has not been explored for the k -secretary problem so far. As a strength of our algorithm we see its simplicity: It is of plain combinatorial nature and can be fine-tuned using only two parameters. In contrast, the optimal algorithms which follow theoretically from the (J, K) -secretary approach [9] would involve k^2 parameters and the same number of different decision rules.

An important insight for the analysis of SINGLE-REF is that items can be partitioned into two classes, which we will call *dominating* and *non-dominating*. Both have certain properties on which we base our fully parameterized analysis. In Table 1, we list the competitive ratios of SINGLE-REF for $k \leq 20$. While the competitive ratio for $k = 1$ is optimal, we obtain a value significantly greater than $1/e$ already for $k = 2$. Furthermore, the competitive ratios are monotonically increasing in the interval $k \in [1..20]$, already breaking the threshold of 0.5 at $k = 6$. Numerical computations suggest that this monotonicity holds for general k . See Figure 2 (p. 11) for the competitive ratios up to $k = 100$ and a comparison with Kleinberg's algorithm [21]. Providing a closed formula for the competitive ratio for any value of k is one direction of future work (see Section 5).

Moreover, we investigate the OPTIMISTIC algorithm by Babaioff et al. [3] for the case $k = 2$. Although Chan et al. [9] provide the optimal algorithm for $k = 2$, we think studying this elegant algorithm is interesting for two reasons: First, a tight analysis of OPTIMISTIC is stated as open problem in [3]. Article [3] does not provide the proof of the $(1/e)$ -bound and a recent journal publication [5] (evolved from [3] and [6]) does not cover the OPTIMISTIC algorithm at all. We make progress in this problem by proving that for $k = 2$ its competitive ratio is exactly 0.4168 which significantly breaks the $(1/e)$ -barrier. Second, our proof reveals an interesting property of this algorithm, which we show in Lemma 4.1: The probability that OPTIMISTIC accepts the second best item is exactly the probability that the optimal algorithm for $k = 1$ from [10, 23] accepts the best item. A similar property might hold for $k \geq 3$, which could be a key insight into the general case.

From a technical point of view, we derive the exact probabilities using basic combinatorial constructs exclusively. This is in contrast to previous approaches [8, 9] which can only be analyzed using heavyweight linear programming techniques. In addition, we always

consider the asymptotic setting of $n \rightarrow \infty$ items, which gives more meaningful bounds on the competitive ratio. Throughout the analyses of both algorithms, we associate probabilities with sets of permutations (see Section 2.2). Hence, probability relations can be shown equivalently by set relations. This is a simple but powerful technique which may be useful in the analysis of other optimization problems with random arrival order as well.

2 Preliminaries

Let $v_1 > v_2 > \dots > v_n$ be the *elements* (also called *items*) of the input. In the ordinal model, we can assume w.l.o.g. all items to be distinct. Therefore we say that i is the *rank* of element v_i . An *input sequence* is any permutation of the list v_1, \dots, v_n . We denote the position of an element v given a specific input sequence π with $\text{pos}_\pi(v) \in \{1, \dots, n\}$ and write $\text{pos}(v)$ whenever the input sequence is clear from the context.

Given any input sequence, an algorithm can accept up to k items, where the decision whether to accept or reject an item must be made immediately upon its arrival. Let ALG denote the sum of items accepted by the algorithm. The algorithm is α -*competitive* if $\mathbf{E}[\text{ALG}] \geq \alpha \cdot \text{OPT}$ holds for all item sets. Here the expectation is taken over the uniform distribution of all $n!$ input sequences and $\text{OPT} = \sum_{i=1}^k v_i$.

Notation. For $a, b \in \mathbb{N}$ with $a \leq b$, we use the notation $[a..b]$ to denote the set of integers $\{a, a+1, \dots, b\}$ and write $[a]$ for $[1..a]$. The (half-)open integer intervals $(a..b]$, $[a..b)$, and $(a..b)$ are defined accordingly. Further, we use the notation $n^{\underline{k}}$ for the falling factorial $\frac{n!}{(n-k)!}$.

2.1 Algorithms

In the following, we state the OPTIMISTIC algorithm proposed by Babaioff et al. (Algorithm 1) and our proposed algorithm SINGLE-REF (Algorithm 2) and compare both strategies.

Algorithm 1 OPTIMISTIC [3].

Parameters: $t \in (k..n - k]$ (sampling threshold)

- 1 **Sampling phase:** Reject the first $t - 1$ items.
 - 2 Let $s_1 > \dots > s_k$ be the k best items from the sampling phase.
 - 3 **Selection phase:** As j -th accepted item, choose the first item better than s_{k-j+1} .
-

Algorithm 2 SINGLE-REF.

Parameters: $t \in (k..n - k]$ (sampling threshold), $r \in [k]$ (reference rank)

- 1 **Sampling phase:** Reject the first $t - 1$ items.
 - 2 Let s_r be the r -th best item from the sampling phase.
 - 3 **Selection phase:** Choose the first k items better than s_r .
-

While both algorithms consist of a sampling phase in which the first $t - 1$ items are rejected, the main difference is the policy for accepting items: OPTIMISTIC uses the k best items from the sampling as reference elements. Right after the sampling phase, the first item better than s_k (the k -th best from the sampling) will be accepted. The following accepted items are chosen similarly, but with $s_{k-1}, s_{k-2}, \dots, s_1$ as reference items. Note that this algorithm always sticks to this order of reference points, even if the first item already outperforms s_1 . Hence, it is optimistic in the sense that it always expects that high-value items occur in the future.

SINGLE-REF has a simpler structure since it only uses a single item s_r from the sampling as reference point. Here, each item is compared to s_r (the r -th best from sampling), thus the first k elements better than s_r will be selected. Despite its simpler structure, the analysis of SINGLE-REF is involved due to the additional parameter r , as it is not clear how to choose this parameter optimally.

Note that in the case $k = 1$, OPTIMISTIC and SINGLE-REF (when setting $r = 1$) become the strategy known for the classical secretary problem [10, 23]: After rejecting the first $t - 1$ items, choose the first one better than the best from sampling. A simple argument shows that this strategy selects the best item with probability $\frac{t-1}{n} \sum_{i=t}^n \frac{1}{i-1}$. If n tends to infinity and $t - 1 \approx n/e$, this term approaches $1/e$ which is optimal.

The following lemma is used to bound the competitive ratios of both algorithms. It heavily relies on the monotonicity property of the algorithms, i.e., for any $v_i > v_j$, both algorithms select v_i with greater or equal probability than v_j .

► **Lemma 2.1.** *Let \mathcal{A} be OPTIMISTIC or SINGLE-REF and for each $i \in [n]$ let p_i be the probability that \mathcal{A} selects item v_i . The competitive ratio of \mathcal{A} is $(1/k) \sum_{i=1}^k p_i$.*

Proof. First, we will argue that $p_i \geq p_{i+1}$ for all $i \in [n - 1]$, i.e., \mathcal{A} selects items of smaller rank with greater or equal probability. This follows if we can show that the number of permutations where v_{i+1} is accepted is not greater than the respective number of permutations for v_i (this concept is described more detailed in Section 2.2).

Consider any input sequence π in which v_{i+1} is accepted. Let $s_j < v_{i+1}$ be the sampling item to which v_{i+1} is compared (in case of SINGLE-REF we have $j = r$). Since v_{i+1} is accepted, we have $s_j \neq v_i$. By swapping v_i with v_{i+1} , we obtain a new permutation π' with the same reference element s_j . This is obvious if v_i is not in the sampling of π . Otherwise, note that in the ordered sequences of sampling items from π and π' , both v_{i+1} and v_i have the same position. This implies that s_j is the j -th best sampling item in π' . Further, item v_i is at the former position of v_{i+1} in π' , thus \mathcal{A} accepts v_i at this position since $v_i > v_{i+1} > s_j$.

Thus, both sequences p_1, \dots, p_k and v_1, \dots, v_k are sorted decreasingly. Let $\text{OPT}_k = \sum_{i=1}^k v_i$ and $\mathbf{E}[\mathcal{A}]$ be the expected sum of the items accepted by \mathcal{A} . Chebyshev's sum inequality [16] states that if $a_1 \geq a_2 \geq \dots \geq a_n$ and $b_1 \geq b_2 \geq \dots \geq b_n$, then $\sum_{i=1}^n a_i b_i \geq (1/n) (\sum_{i=1}^n a_i) (\sum_{i=1}^n b_i)$. Applying this inequality yields

$$\mathbf{E}[\mathcal{A}] = \sum_{i=1}^n p_i v_i \geq \sum_{i=1}^k p_i v_i \geq \frac{1}{k} \left(\sum_{i=1}^k v_i \right) \left(\sum_{i=1}^k p_i \right) = \left(\frac{1}{k} \sum_{i=1}^k p_i \right) \text{OPT}_k.$$

Note that the above inequalities are tight: Assuming that the first k items are almost identical, i.e. $v_i = 1 - i\varepsilon$ for $i \in [1..k]$ and $\varepsilon \rightarrow 0$, and $v_i = 0$ for all remaining items of rank $i \in (k..n]$, the competitive ratio is exactly $(1/k) \sum_{i=1}^k p_i$. ◀

The same argument is used in [8] to show the equivalence of the k -secretary and the (k, k) -secretary problem for ordinal monotone algorithms.

2.2 Random Order Model

To analyze an algorithm given a random permutation, we often fix an order u_1, u_2, \dots, u_n of positions. Then, we draw the element for position u_1 uniformly from all n elements, next the element for position u_2 from the remaining $n - 1$ elements, and so on. It is easy to see that by this process we obtain a permutation drawn uniformly at random.

Moreover, the uniform distribution allows us to prove probability relations using functions: Suppose that p_i is the probability that item v_i is accepted in a random permutation, then $p_i = |P_i|/n!$ where P_i is the set of all input sequences where v_i is accepted. Thus, we can

■ **Table 2** Several identities involving binomial coefficients [16].

Rule	Equation	Parameters
(R1) Sum of products	$\sum_{k=0}^l \binom{l-k}{m} \binom{q+k}{n} = \binom{l+q+1}{m+n+1}$	$l, m, n, q \in \mathbb{Z}$ with $l, m \geq 0$ and $n \geq q \geq 0$
(R2) Symmetry	$\binom{n}{k} = \binom{n}{n-k}$	$n, k \in \mathbb{Z}$ with $n \geq 0$
(R3) Trinomial revision	$\binom{r}{m} \binom{m}{k} = \binom{r}{k} \binom{r-k}{m-k}$	$m, k \in \mathbb{Z}$ and $r \in \mathbb{R}$

prove $p_i \leq p_j$ by finding an injective function $f: P_i \rightarrow P_j$ and get $p_i = p_j$ if f is bijective. For example, this technique turns out to be highly useful in the proof of Lemma 4.1, where probabilities of different algorithms are related.

2.3 Combinatorics

We often need to analyze probabilities described by the following random experiment.

► **Fact 2.2.** *Suppose there are N balls in an urn from which M are blue and $N - M$ red. The probability of drawing K blue balls without replacement in a sequence of length K is $h(N, M, K) := \binom{M}{K} / \binom{N}{K}$.*

This fact follows from a special case of the hypergeometric distribution.

Furthermore, we make use of several identities involving binomial coefficients throughout the following sections. These equations, denoted by (R1), (R2), and (R3), are listed in Table 2.

3 Analysis of SINGLE-REF

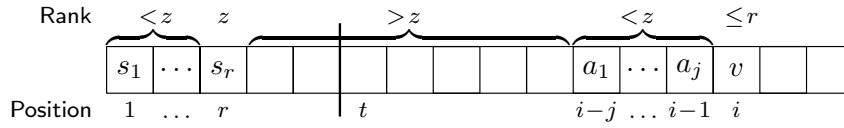
In this section we analyze our proposed algorithm SINGLE-REF, which we denote by \mathcal{A} throughout this section. Recall that this algorithm uses s_r , the r -th best sampling item, as the threshold for accepting items. As implied by the proof of Lemma 2.1, only the k largest items v_1, \dots, v_k contribute to the objective function. One essential idea of our approach is to separate the set of top- k items into two classes according to the following definition.

► **Definition 3.1.** *We say that item v_i is dominating if $i \leq r$, and non-dominating if $r + 1 \leq i \leq k$.*

The crucial property of dominating items becomes clear in the following scenario: Assume that any dominating item v occurs after the sampling phase. Since s_r is the r -th best item from the sampling phase, it follows that $v > s_r$. That is, each dominating item outside the sampling beats the reference item. Therefore there are only two situations when dominating items are rejected: Either they appear before position t , or after k accepted items.

3.1 Acceptance of Dominating Items

First we focus on dominating items. As we will show in Lemma 3.2, the algorithm cannot distinguish between them and thus each dominating item has equal acceptance probability.



■ **Figure 1** Event $\tilde{E}_j(z, i)$ considered in the proof of Lemma 3.2.

► **Lemma 3.2.** *Let v be a dominating item and $j \in [0..k)$. Let E_j be the event that \mathcal{A} selects v as $(j + 1)$ -th item. It holds that $\Pr[E_j] = \frac{\kappa\tau}{n} \sum_{i=t+j}^n \binom{i-t}{j} \frac{1}{(i-1)^{r+j}}$, where $\tau = (t - 1)^\tau$ and $\kappa = (r - 1 + j)^{\underline{j}}$.*

Proof. Let $E_j(z, i)$ be the event that \mathcal{A} accepts v as $(j + 1)$ -th item at position $i = \text{pos}(v)$ and s_r has rank z (thus $s_r = v_z$). Note that there must be elements s_1, \dots, s_{r-1} of rank smaller than z in the sampling (such that s_r is in fact the r -th best sampling element). Similarly, there must be j elements a_1, \dots, a_j after the sampling but before v of rank smaller than z (which are accepted by \mathcal{A}).

The proof is in several steps. We first consider a stronger event $\tilde{E}_j(z, i)$. Later, we show how the probability of $E_j(z, i)$ can be obtained from $\tilde{E}_j(z, i)$. In the end, the law of total probability yields $\Pr[E_j]$.

Analysis of $\tilde{E}_j(z, i)$. Event $\tilde{E}_j(z, i)$ is defined as $E_j(z, i)$ with additional position constraints (see Figure 1): Elements s_1, \dots, s_r are in this order at the first r positions and elements a_1, \dots, a_j are in this order at the j positions immediately before v . Therefore, $\tilde{E}_j(z, i)$ occurs if and only if the following conditions hold:

- (i) $\text{pos}(v) = i$, $\text{pos}(s_\ell) = \ell$ for $\ell \in [r]$, and $\text{pos}(a_m) = i - j + m - 1$ for $m \in [j]$.
- (ii) Elements s_1, \dots, s_{r-1} have rank smaller than z
- (iii) Elements a_1, \dots, a_j have rank smaller than z
- (iv) All remaining items at positions $r + 1, \dots, i - j - 1$ have rank greater than z .

Using the concept described in Section 2.2, we think of sequentially drawing the elements for the positions $1, \dots, r, i - j, \dots, i$ and then $r + 1, \dots, i - j - 1$. The probability for (i) is $\prod_{\ell=0}^{j+r} \frac{1}{n-\ell} = 1/n^{\underline{j+r+1}} =: \beta$, since each item has the same probability to occur at each remaining position. In (ii), the $r - 1$ elements can be chosen out of $z - 2$ remaining items of rank smaller than z (since v is dominating and was already drawn). Therefore we get a factor of $\binom{z-2}{r-1}$. After this step, there remain $z - 2 - (r - 1) = z - r - 1$ elements of rank smaller than z , so we get factor $\binom{z-r-1}{j}$ for step (iii).

Finally, the probability of (iv) can be formulated using Fact 2.2. Note that at this point, there remain $n - (1 + r + j)$ items and no item of rank greater than z has been drawn so far. In terms of the random experiment described in Fact 2.2, we draw $K = i - j - r - 1$ balls (items) from an urn of size $N = n - (1 + r + j)$ where $M = n - z$ balls are blue (rank greater than z). Hence, the probability for (iv) is $H := h(n - r - j - 1, n - z, i - j - r - 1)$. Therefore we obtain

$$\Pr[\tilde{E}_j(z, i)] = \beta \cdot \binom{z-2}{r-1} \binom{z-r-1}{j} \cdot H. \tag{1}$$

This term can be simplified further by applying (R3) and (R2). Let $R = z - 2$, $K = r - 1$, and $M = j + r - 1$. It holds that

$$\binom{z-2}{r-1} \binom{z-r-1}{j} \stackrel{(R3)}{=} \binom{R}{M} \binom{M}{K} \stackrel{(R2)}{=} \binom{R}{M} \binom{M}{M-K} = \binom{z-2}{j+r-1} \binom{j+r-1}{j}.$$

Let $\kappa = (j + r - 1)^{\underline{j}}$, then $\binom{j+r-1}{j} = \kappa/j!$ and we get $\Pr[\tilde{E}_j(z, i)] = \frac{\beta\kappa}{j!} \cdot \binom{z-2}{j+r-1} \cdot H$.

Relating $\tilde{E}_j(z, i)$ to $E_j(z, i)$. In contrast to $\tilde{E}_j(z, i)$, in the event $E_j(z, i)$, the elements s_1, \dots, s_r can have any positions in $[t-1]$ and a_1, \dots, a_j any positions in $[t, i]$. In the random order model, the probability of an event depends linearly on the number of permutations for which the event happens. Hence, we can multiply the probability with corresponding factors $(t-1)^r =: \tau$ and $(i-t)^j = \binom{i-t}{j} j!$ and get $\Pr[E_j(z, i)] = \binom{i-t}{j} \tau j! \cdot \Pr[\tilde{E}_j(z, i)]$.

Relating $E_j(z, i)$ to E_j . As the final step, we sum over all possible values for i and z to obtain $\Pr[E_j]$. The position i of item v ranges between $t+j$ and n , while the reference rank z is between $r+j+1$ (there are $r-1$ sampling elements and $j+1$ accepted elements of rank less than z) and n . Thus we get:

$$\begin{aligned} \Pr[E_j] &= \sum_{i=t+j}^n \sum_{z=r+j+1}^n \Pr[E_j(z, i)] = \tau j! \sum_{i=t+j}^n \binom{i-t}{j} \sum_{z=r+j+1}^n \Pr[\tilde{E}_j(z, i)] \\ &= \beta \kappa \tau \sum_{i=t+j}^n \binom{i-t}{j} \sum_{z=r+j+1}^n \binom{z-2}{j+r-1} \cdot H \\ &= \beta \kappa \tau \sum_{i=t+j}^n \binom{i-t}{j} \frac{1}{\binom{n-r-j-1}{i-j-r-1}} \sum_{z=r+j+1}^n \binom{z-2}{j+r-1} \binom{n-z}{i-j-r-1}, \end{aligned} \quad (2)$$

where the last step follows from Fact 2.2. The sum over z in Equation (2) can be resolved using (R1). Let $L = n - r - j - 1$, $N = Q = r + j - 1$, and $M = i - j - r - 1$. Then we have

$$\begin{aligned} \sum_{z=r+j+1}^n \binom{z-2}{j+r-1} \binom{n-z}{i-j-r-1} &= \sum_{z=0}^{n-r-j-1} \binom{r+j-1+z}{j+r-1} \binom{n-r-j-1-z}{i-j-r-1} \\ &= \sum_{z=0}^L \binom{Q+z}{N} \binom{L-z}{M} = \binom{L+Q+1}{M+N+1} = \binom{n-1}{i-1}. \end{aligned} \quad (3)$$

Note that in order to apply (R1) we need to verify $L, M \geq 0$ and $N \geq Q \geq 0$. We can assume $k \leq n/2$, since for $k > n/2$, there exist a trivial $(1/2)$ -competitive algorithm. Therefore, we have $L = n - r - j - 1 \geq n - k - (k - 1) - 1 = n - 2k \geq 0$. Further, $i \geq t + j$, thus $i - j \geq t \geq k + 1 \geq r + 1$ which implies $M \geq 0$. The condition $N \geq Q \geq 0$ holds trivially. By inserting Equation (3) into Equation (2), we obtain the quotient of binomial coefficients $\binom{n-1}{i-1} / \binom{n-r-j-1}{i-j-r-1}$. From (R3) we get

$$\binom{n-1}{i-1} / \binom{n-1-(r+j)}{i-1-(r+j)} = \binom{n-1}{r+j} / \binom{i-1}{r+j} = \frac{(n-1)^{r+j}}{(i-1)^{r+j}}.$$

Recall $\beta = 1/n^{j+r+1}$, thus $(n-1)^{r+j} \cdot \beta = 1/n$. Together with Equation (2) we get

$$\Pr[E_j] = \beta \kappa \tau \cdot (n-1)^{r+j} \sum_{i=t+j}^n \binom{i-t}{j} \frac{1}{(i-1)^{r+j}} = \frac{\kappa \tau}{n} \sum_{i=t+j}^n \binom{i-t}{j} \frac{1}{(i-1)^{r+j}}, \quad (4)$$

which concludes the proof. \blacktriangleleft

Lemma 3.2 provides the exact probability that a dominating item is accepted as $(j+1)$ -th item. However, it is more meaningful to consider the asymptotic setting where $n \rightarrow \infty$. Here, we assume $t-1 = cn$ for some constant $c \in (0, 1)$. For this setting, we obtain the following lemma.

► **Lemma 3.3.** *Let E_j be defined as in Lemma 3.2. In the asymptotic setting described above,*

(A) *For $r = 1$ it holds that $\Pr[E_j] = c \left(\ln \frac{1}{c} + \sum_{\ell=1}^j \beta_\ell \frac{c^\ell - 1}{\ell} \right)$, where $\beta_\ell = (-1)^{\ell+1} \binom{j}{\ell}$ for $\ell \in [j]$.*

(B) *For $r \geq 2$ it holds that $\Pr[E_j] = \frac{c}{r-1} - \frac{c^r(1-c)^j}{r-1} \sum_{\ell=0}^j \alpha_\ell \left(\frac{c}{1-c} \right)^\ell$, where $\alpha_\ell = \binom{j+r-1}{\ell+r-1}$ for $\ell \in [0..j]$.*

The proof of Lemma 3.3 relies on a sequence of technical lemmas and is given in Appendix A.

► **Remark.** As described in Section 2.1, SINGLE-REF generalizes the optimal strategy for the secretary problem ($k = 1$). Note that the combinatorial analysis from Lemma 3.2 as well as the asymptotic bound from Lemma 3.3 give exactly the respective terms from the secretary problem. To see this, we set $r = 1$ and consider the probability that the dominating item v_1 is accepted as first item. By Lemma 3.2 (with $j = 0$), the success probability is $\frac{t-1}{n} \sum_{i=t}^n \frac{1}{i-1}$. Moreover, Lemma 3.3(A) provides the asymptotic bound of $c \ln(1/c)$ for this case.

3.2 Non-Dominating Items

It remains to consider the acceptance probabilities of the non-dominating items v_{r+1}, \dots, v_k . Fortunately, there exist some interesting connections to the probabilities for dominating items.

► **Lemma 3.4.** *Let $i \in [1..k-r]$ and $j \in [1..i]$. For the non-dominating item v_{r+i} it holds that $\Pr[v_{r+i} \text{ is } j\text{-th accept}] = \Pr[v_{r+i} \text{ is } (i+1)\text{-th accept}]$.*

Proof. First we argue that there are in total at least $i+1$ accepts if v_{r+i} is accepted. Assuming that v_{r+i} is accepted, we have $s_r < v_{r+i}$. Let \mathcal{S} be the set of elements which the algorithm may accept, i.e. $\mathcal{S} = \{v_1, \dots, v_{r+i}\}$. Since s_r is the r -th best element in the sampling, at most $r-1$ elements from \mathcal{S} can be part of the sampling and thus at least $r+i-(r-1) = i+1$ elements from \mathcal{S} , including v_{r+i} , are accepted.

As described in Section 2.2, we construct a bijective function $f: P \rightarrow Q$ where P (resp. Q) is the set of permutations where v_{r+i} is the j -th (resp. $(i+1)$ -th) accept. For each input sequence $\pi \in P$, let a_1, \dots, a_{i+1} with $a_j = v_{r+i}$ denote the first $i+1$ accepts. The function f swaps the positions of a_1, \dots, a_{i+1} in a cyclic shift, such that $a_j = v_{r+i}$ is at the former position of a_{i+1} . In other words, the relative order of the first $i+1$ accepted elements in $f(\pi)$ is changed in a way that v_{r+i} is the $(i+1)$ -th accept in $f(\pi)$. Note that the cyclic shift can be reversed, thus f is bijective. ◀

While Lemma 3.4 relates the acceptance probabilities of a single non-dominating item, the claim of Lemma 3.5 is in a way orthogonal by relating probabilities of non-dominating items to those for dominating items.

► **Lemma 3.5.** *Let $i \in [1..k-r]$ and $j \in [1..k-i]$. For the non-dominating item v_{r+i} and any dominating item v^+ it holds that $\Pr[v_{r+i} \text{ is } (i+j)\text{-th accept}] = \Pr[v^+ \text{ is } (i+j)\text{-th accept}]$.*

Proof. Let P be the set of permutations where v_{r+i} is the $(i+j)$ -th accept and let Q contain those where v^+ is the $(i+j)$ -th accept. We prove the claim by defining a bijective function $f: P \rightarrow Q$. Let f be the function that swaps v_{r+i} with v^+ in the input sequence.

Consider any input sequence $\pi \in P$. As v_{r+i} is accepted, $s_r < v_{r+i}$. We can argue that in $f(\pi)$ element s_r is still the r -th best element of the sampling: This holds clearly if no item is moved out of or into the sampling. Otherwise, f moves v_{r+i} into the sampling and v^+ outside. But since $s_r < v_{r+i} < v^+$, this does not change the role of s_r as the r -th best sampling element. Thus f is injective.

18:10 New Results for the k -Secretary Problem

To prove that f is surjective, let $\pi' \in Q$ be any input sequence where v^+ is the $(i+j)$ -th accept. We next consider the rank z of $s_r = v_z$. As there must be sampling elements s_1, \dots, s_{r-1} and accepted elements $a_1, \dots, a_{i+j-1}, v^+$ of rank smaller than z , we have $z > (r-1) + (i+j-1) + 1 \geq r+i$. Hence, $s_r < v_{r+i}$. The inverse function of f consists in swapping back v^+ with v_{r+i} . For the same reason as above, this maintains s_r . As $s_r < v_{r+i}$, element v_{r+i} gets accepted, thus $f^{-1}(\pi') \in P$. \blacktriangleleft

Using the previous results for dominating and non-dominating items we are now ready to state the main result of this section, namely the competitive ratio of SINGLE-REF. Due to the complex expressions from Lemma 3.3 we give numerical results for small values of k .

► Theorem 3.6. *In the asymptotic setting of $n \rightarrow \infty$ and assuming that $t-1 = cn$ for a constant $c \in (0, 1)$, SINGLE-REF achieves the competitive ratios given in Table 1.*

Proof. For an item v_i , let $p_i^{(j)}$ be the probability that v_i is the j -th accept (with $1 \leq j \leq k$). The total acceptance probability of v_i is denoted by $p_i = \sum_{j=1}^k p_i^{(j)}$. According to Lemma 3.2, each dominating item has the same acceptance probability for a fixed acceptance position. Therefore, in the following we simply write p_1 (resp. $p_1^{(j)}$) for the acceptance probability of any dominating item.

By Lemma 2.1 the competitive ratio can be obtained by summing over the acceptance probabilities of all items divided by k . Clearly, $\sum_{i=1}^r p_i = rp_1$. Now consider any non-dominating item v_{r+i} . According to Lemmas 3.4 and 3.5, p_{r+i} can be related to respective probabilities $p_1^{(j)}$: It holds that $p_{r+i}^{(j)} = p_1^{(z)}$ with $z = \max\{j, i+1\}$. Therefore $p_{r+i} = \sum_{j=1}^k p_{r+i}^{(j)} = \sum_{j=1}^i p_1^{(i+1)} + \sum_{j=i+1}^k p_1^{(j)} = ip_1^{(i+1)} + \sum_{j=i+1}^k p_1^{(j)}$. Hence, we obtain the competitive ratio

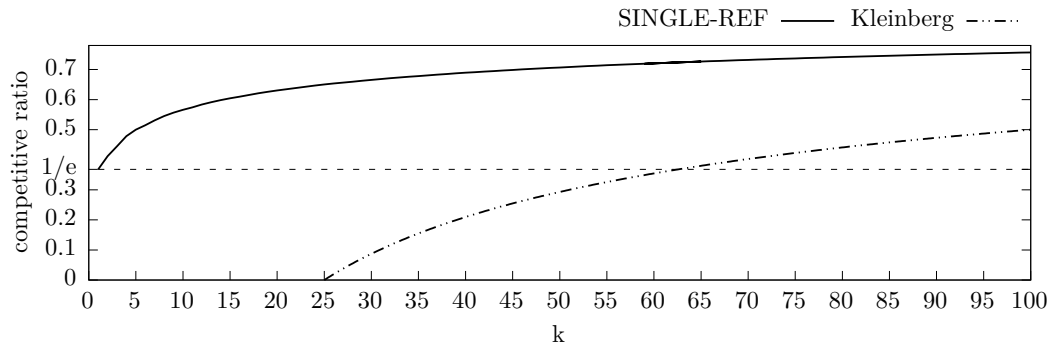
$$\frac{1}{k} \sum_{i=1}^k p_i = \frac{1}{k} \left(rp_1 + \sum_{i=1}^{k-r} \left(ip_1^{(i+1)} + \sum_{j=i+1}^k p_1^{(j)} \right) \right) \quad (5)$$

with $p_1^{(j)} = \Pr[E_{j-1}]$ for the event E_j considered in Lemmas 3.2 and 3.3. To evaluate the performance of our algorithm, we maximized Equation (5) over the parameters r and c using a computer algebra system. This yields the competitive ratios shown in Table 1 (p. 3). \blacktriangleleft

For completeness, we evaluated the competitive ratio of SINGLE-REF in the interval $k \in [1..100]$ using the optimization procedure mentioned in the previous proof. Figure 2 shows the performance of SINGLE-REF in comparison with Kleinberg's result [21]; our algorithm reaches competitive ratios of up to 0.75 and outperforms the algorithm from [21] on this interval. In Appendix A, we provide the full list of optimal parameters for $k \in [1..100]$ (see Table 3, p. 19).

4 Analysis of OPTIMISTIC for $k = 2$

In this section we sketch the analysis of OPTIMISTIC for $k = 2$. Due to space constraints, for some proofs we refer to the full version of this paper. Let \mathcal{A}_2 denote OPTIMISTIC algorithm with $k = 2$ in the following. As implied by Lemma 2.1 the competitive ratio is determined by p_1 and p_2 , the probabilities that \mathcal{A}_2 accepts v_1 and v_2 , respectively. To find these probabilities, we make use of the relation between probabilities and sets (see Section 2.2). Let P_i be the set of permutations in which \mathcal{A}_2 accepts v_i .



■ **Figure 2** Comparison of our algorithm SINGLE-REF and the algorithm by Kleinberg [21], for $k \in [1..100]$. The parameters r and c for SINGLE-REF are chosen optimally.

Probability p_2 . In the next lemma, we show a surprising relation between OPTIMISTIC for $k = 2$ and the algorithm for the classical (1-)secretary problem (see Section 2.1). The proof of Lemma 4.1 uses a sophisticatedly tailored bijection between two respective sets of permutations. We sketch the proof method here and give the entire proof in the full version of this paper.

► **Lemma 4.1.** *Let \mathcal{A}_1 be the algorithm for the classical secretary problem. Assuming that both algorithms $\mathcal{A}_1, \mathcal{A}_2$ are parameterized with the same t , we have that $p_2 = \Pr[\mathcal{A}_2 \text{ accepts } v_2] = \Pr[\mathcal{A}_1 \text{ accepts } v_1]$.*

Sketch of proof. Equivalently, we prove that the corresponding complementary events happen with the same probability. For this purpose, we define for each permutation π where \mathcal{A}_2 does not accept v_2 a unique permutation $f(\pi)$ where \mathcal{A}_1 does not accept v_1 . Different situations where \mathcal{A}_2 does not accept v_2 lead to a total number of five cases. If v_2 is in the sampling of π , we define $f(\pi)$ such that the positions of v_1 and v_2 are swapped. Then, \mathcal{A}_1 clearly does not accept v_1 in $f(\pi)$. Another case is when v_2 comes behind two accepted elements a_1, a_2 in π and $v_1 = a_1$ is the first accept. Note that since a_2 is accepted, $a_2 > s_1$. In this case, $f(\pi)$ can be defined by swapping the positions of both accepts v_1 and a_2 . Recall that \mathcal{A}_1 accepts the first item better than s_1 following the sampling phase which is a_2 in $f(\pi)$, thus v_1 is not selected.

In the full proof, we consider all five cases according to π . In each case it is enough to define f such that the positions of at most three elements are swapped. Finally, we have to argue that the function f is indeed bijective. ◀

Probability p_1 . In this part, we argue that $p_1 = p_2 + \delta$ holds for some $\delta > 0$. To obtain δ , we again consider cardinalities of sets instead of probabilities. First, we observe that P_2 can be related to a set $P'_1 \subset P_1$ such that P_2 and P'_1 have equal size.

► **Lemma 4.2.** *Let $P'_1 = \{\pi \in P_1 \mid \text{pos}_\pi(v_2) < t \Rightarrow \mathcal{A}_2 \text{ accepts } v_1 \text{ as first item}\}$. It holds that $|P'_1| = |P_2|$.*

Proof. Let $f: P_2 \rightarrow P'_1$ be the function that swaps v_1 with v_2 in the given sequence. We first have to argue that in fact $f: P_2 \rightarrow P'_1$, therefore let $\pi \in P_2$ be given. Then, v_1 gets accepted by \mathcal{A}_2 in $f(\pi)$ at the position $\text{pos}_{f(\pi)}(v_1) = \text{pos}_\pi(v_2)$, as v_1 is an item of higher value. So far we have $f(\pi) \in P_1$. If $\text{pos}_{f(\pi)}(v_2) \geq t$, there is nothing to show. Assuming that $\text{pos}_{f(\pi)}(v_2) < t$, it follows $\text{pos}_\pi(v_1) < t$, i.e. v_1 was the best element in the sampling of π . Since no item (particularly not v_2) can beat v_1 , but v_2 was accepted by \mathcal{A}_2 in π , we get that v_2 was the first accept in π . Hence v_1 is the first accept in $f(\pi)$.

Clearly, f is injective. For surjectivity, let $\pi' \in P'_1$ and let π the permutation obtained from π' by swapping (back) v_1 with v_2 . If $\text{pos}_{\pi'}(v_2) < t$, by definition of P'_1 we know that v_1 is the first accept in π' , implying that no item before $\text{pos}_{\pi'}(v_1) = \text{pos}_{\pi'}(v_2)$ is chosen by \mathcal{A}_2 . In the case $\text{pos}_{\pi'}(v_2) \geq t$, since $\text{pos}_{\pi'}(v_1) \geq t$, the smallest rank in the sampling of π' is 3 or greater. Therefore, v_2 gets accepted if not more than one item before v_2 gets accepted. This is the case in π , as $\text{pos}_{\pi}(v_2) = \text{pos}_{\pi'}(v_1)$. \blacktriangleleft

Since $|P_1| = |P'_1| + |P_1 \setminus P'_1| = |P_2| + |P_1 \setminus P'_1|$, we therefore get $\delta = |P_1 \setminus P'_1|/n!$, i.e., δ is the probability that a random permutation is in the set $|P_1 \setminus P'_1|$. This probability is considered in Lemma 4.3.

► **Lemma 4.3.** *Let $\delta = \Pr[\pi \in P_1 \setminus P'_1]$ where π is drawn uniformly from the set of all permutations and P'_1 is defined like in Lemma 4.2. It holds that $\delta = \frac{t-1}{n} \frac{t-2}{n-1} \sum_{i=t}^{n-1} \frac{n-i}{(i-2)(i-1)}$.*

The proof of Lemma 4.3 relies on a counting argument similar to the proof of Lemma 3.2. We prove Lemma 4.3 in the full version of this paper.

Competitive ratio. From Lemmas 4.1 and 4.3, we know the exact probabilities p_2 and p_1 . For particular n , the term $(p_1 + p_2)/2$ can be optimized over t to find the optimal sampling size. In the following theorem we consider the asymptotic setting $n \rightarrow \infty$. Here, we assume that the sampling size is a constant fraction of the input size, i.e., $t - 1 = cn$ for some constant $c \in (0, 1)$.

► **Theorem 4.4.** *For $k = 2$, the algorithm OPTIMISTIC is 0.4168-competitive in the limit $n \rightarrow \infty$ and assuming that the sampling size is $t - 1 = cn$ for $c = 0.3521$.*

Proof. According to Lemma 4.1, p_2 is the probability that the classical secretary algorithm accepts the best item, i.e., $p_2 = \frac{t-1}{n} \sum_{i=t}^n \frac{1}{i-1}$. This term approaches $c \ln(1/c)$ asymptotically. From Lemma 4.3 we know $p_1 = p_2 + \delta$, where $\delta = \frac{t-1}{n} \frac{t-2}{n-1} \sum_{i=t}^{n-1} \frac{n-i}{(i-2)(i-1)}$. For $n \rightarrow \infty$, the sum $\sum_{i=t}^{n-1} \frac{n-i}{(i-2)(i-1)}$ is bounded from above and below by $\frac{1}{c} - \ln \frac{1}{c} - 1$. This can be seen by bounding the sum by two corresponding integrals. Further, $\lim_{n \rightarrow \infty} \frac{t-1}{n} \frac{t-2}{n-1} = c^2$. Therefore, $\delta = c^2 \left(\frac{1}{c} - \ln \frac{1}{c} - 1 \right)$ for large n . According to Lemma 2.1, \mathcal{A}_2 is $\alpha(c)$ -competitive with

$$\alpha(c) = \frac{1}{2} (p_1 + p_2) = \frac{1}{2} (p_2 + \delta + p_2) = c \ln \frac{1}{c} + \frac{c^2}{2} \left(\frac{1}{c} - \ln \frac{1}{c} - 1 \right).$$

Setting $c = 1/e$, we obtain a competitive ratio of $\alpha(1/e) = \frac{3e-2}{2e^2} \approx 0.4164$. However, the optimal choice for c is around $c^* = 0.3521 < 1/e$, improving the competitive ratio slightly to $\alpha(c^*) \approx 0.4168$. \blacktriangleleft

5 Conclusion and Future Work

We investigated two algorithms for the k -secretary problem with a focus on small values for $k \geq 2$. Aside from a tight analysis of the OPTIMISTIC algorithm [3] for $k = 2$, we introduced and analyzed the algorithm SINGLE-REF. For any value of k , the competitive ratio of SINGLE-REF can be obtained by numerical optimization.

We see various directions of future work. For SINGLE-REF, it remains to find the right dependency between the parameters r , c , and k in general and to find a closed formula for the competitive ratio for any value of k . OPTIMISTIC seems a promising and elegant algorithm, however no tight analysis for general $k \geq 3$ is known so far. For $k = 2$, we identified a key property in Lemma 4.1. Similar properties may hold in the general case. Lastly, to the best of our knowledge, no hardness results for the k -secretary problem are known (apart from the cases $k \leq 2$).

References

- 1 S. Agrawal, Z. Wang, and Y. Ye. A Dynamic Near-Optimal Algorithm for Online Linear Programming. *Operations Research*, 62(4):876–890, 2014.
- 2 M. Ajtai, N. Megiddo, and O. Waarts. Improved algorithms and analysis for secretary problems and generalizations. *SIAM Journal on Discrete Mathematics*, 14(1):1–27, 2001.
- 3 M. Babaioff, N. Immorlica, D. Kempe, and R. Kleinberg. A Knapsack Secretary Problem with Applications. In *Proc. 10th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems and 11th International Workshop on Randomization and Computation (APPROX/RANDOM)*, pages 16–28, 2007.
- 4 M. Babaioff, N. Immorlica, D. Kempe, and R. Kleinberg. Online auctions and generalized secretary problems. *SIGecom Exchanges*, 7(2), 2008.
- 5 M. Babaioff, N. Immorlica, D. Kempe, and R. Kleinberg. Matroid Secretary Problems. *Journal of the ACM (JACM)*, 65(6):35:1–35:26, 2018.
- 6 M. Babaioff, N. Immorlica, and R. Kleinberg. Matroids, secretary problems, and online mechanisms. In *Proc. 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 434–443, 2007.
- 7 B. Bahmani, A. Mehta, and R. Motwani. A 1.43-Competitive Online Graph Edge Coloring Algorithm in the Random Order Arrival Model. In *Proc. 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 31–39, 2010.
- 8 N. Buchbinder, K. Jain, and M. Singh. Secretary Problems via Linear Programming. *Mathematics of Operations Research*, 39(1):190–206, 2014.
- 9 T.-H. H. Chan, F. Chen, and S. H.-C. Jiang. Revealing Optimal Thresholds for Generalized Secretary Problem via Continuous LP: Impacts on Online K -Item Auction and Bipartite K -Matching with Random Arrival Order. In *Proc. 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1169–1188, 2015.
- 10 E. B Dynkin. The optimum choice of the instant for stopping a Markov process. *Soviet Mathematics*, 4:627–629, 1963.
- 11 M. Feldman, O. Svensson, and R. Zenklusen. A Simple $O(\log \log(\text{rank}))$ -Competitive Algorithm for the Matroid Secretary Problem. In *Proc. 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1189–1201, 2015.
- 12 M. Feldman, O. Svensson, and R. Zenklusen. A Framework for the Secretary Problem on the Intersection of Matroids. In *Proc. 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 735–752, 2018.
- 13 T. S. Ferguson. Who Solved the Secretary Problem? *Statistical Science*, 4(3):282–289, 1989.
- 14 P.R. Freeman. The secretary problem and its extensions: A review. *International Statistical Review/Revue Internationale de Statistique*, pages 189–206, 1983.
- 15 O. Göbel, T. Kesselheim, and A. Tönnis. Online Appointment Scheduling in the Random Order Model. In *Proc. 23rd Annual European Symposium on Algorithms (ESA)*, pages 680–692, 2015.
- 16 R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete mathematics - a foundation for computer science (2. ed.)*. Addison-Wesley, 1994.
- 17 M. Hoefer and B. Kodric. Combinatorial Secretary Problems with Ordinal Information. In *44th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 133:1–133:14, 2017.
- 18 C. Kenyon. Best-Fit Bin-Packing with Random Order. In *Proc. 7th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 359–364, 1996.
- 19 T. Kesselheim, K. Radke, A. Tönnis, and B. Vöcking. Primal beats dual on online packing LPs in the random-order model. In *Proc. 46th Annual ACM Symposium on Theory of Computing (STOC)*, pages 303–312, 2014.
- 20 T. Kesselheim and A. Tönnis. Submodular Secretary Problems: Cardinality, Matching, and Linear Constraints. In *Proc. 20th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems and 21st International Workshop on Randomization and Computation (APPROX/RANDOM)*, pages 16:1–16:22, 2017.

- 21 R. D. Kleinberg. A multiple-choice secretary algorithm with applications to online auctions. In *Proc. 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 630–631, 2005.
- 22 O. Lachish. $O(\log \log \text{Rank})$ Competitive Ratio for the Matroid Secretary Problem. In *Proc. 55th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 326–335, 2014.
- 23 D. V Lindley. Dynamic programming and decision theory. *Applied Statistics*, pages 39–51, 1961.
- 24 M. Mahdian and Q. Yan. Online bipartite matching with random arrivals: an approach based on strongly factor-revealing LPs. In *Proc. 43rd ACM Symposium on Theory of Computing (STOC)*, pages 597–606, 2011.

A Technical Proofs for SINGLE-REF

In several lemmas we need to find closed expressions for sums over values of a certain function. If the function is monotone, such sums can be bounded by corresponding integrals:

► **Fact A.1.** Let $f: \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$ and $a, b \in \mathbb{N}$.

- (A) If f is monotonically decreasing, then $\int_a^{b+1} f(i) \, di \leq \sum_{i=a}^b f(i) \leq \int_{a-1}^b f(i) \, di$.
- (B) If f is monotonically increasing, then $\int_{a-1}^b f(i) \, di \leq \sum_{i=a}^b f(i) \leq \int_a^{b+1} f(i) \, di$.

In Lemma 3.3 we consider the acceptance probabilities of dominating items in the asymptotic setting $n \rightarrow \infty$ with $t - 1 = cn$ for $c \in (0, 1)$. We can assume further that $j, r \leq k = o(n)$. In the following, we prove Lemma 3.3 using some technical lemmas, stated and proven below the main proof.

Proof of Lemma 3.3. We first consider the sum $S := \sum_{i=t+j}^n \binom{i-t}{j} \frac{1}{(i-1)^{r+j}}$ from Equation (4) and obtain the following lower bound:

$$\begin{aligned} S &= \sum_{i=t+j}^n \binom{i-t}{j} \frac{1}{(i-1)^{r+j}} = \frac{1}{j!} \sum_{i=t+j}^n \frac{(i-t)^j}{(i-1)^{r+j}} \geq \frac{1}{j!} \sum_{i=t+j}^n \frac{(i-t-j+1)^j}{(i-1)^{r+j}} \\ &= \frac{1}{j!} \sum_{i=1}^{n-t-j+1} \frac{i^j}{(i+t+j-2)^{r+j}}. \end{aligned}$$

Let $f(i) = i^j / (i+y)^{r+j}$ for $y = t+j-2$. Note that y can be seen as a constant independent from i . Let $m = n - t - j + 1$, now the above inequality reads as $S \geq (1/j!) \sum_{i=1}^m f(i)$. In the following we investigate the function f .

Unfortunately, f is in general not monotone, hence we can not apply Fact A.1A or Fact A.1B directly in order to bound the sum by an integral. However, we can split the sum into two monotone parts. Let d be defined like in Lemma A.2 (following this proof). Now we can apply Fact A.1 as follows:

$$\begin{aligned} \sum_{i=1}^m f(i) &= \sum_{i=1}^d f(i) + \sum_{i=d+1}^m f(i) \geq \int_0^d f(i) \, di + \int_{d+1}^{m+1} f(i) \, di \\ &= \int_0^{m+1} f(i) \, di - \int_d^{d+1} f(i) \, di. \end{aligned} \tag{6}$$

Finding the indefinite integral $\int f(i) di$ turns out to be a technical task and is therefore moved to separate lemmas (see Lemmas A.3 and A.4). If $F(i)$ is a function with $F'(i) = f(i)$, we have for κ, τ defined like in Equation (4)

$$\Pr[E_j] = \frac{\kappa\tau}{n} S \geq \frac{\kappa\tau}{n j!} (F(m+1) - F(0) - F(d+1) + F(d)). \tag{7}$$

In the remainder of the proof we consider the two cases $r = 1$ and $r \geq 2$ separately.

Case A: $r = 1$. Let $F(i)$ and β_ℓ be defined like in Lemma A.3. In Equation (7), the factor $\frac{\kappa\tau}{n j!}$ resolves to c as $\kappa = (j+r-1)^{\underline{j}} = j^{\underline{j}} = j!$ and $\tau = (t-1)^{\underline{x}} = (t-1)^{\underline{1}} = t-1$. Further it holds that

$$\begin{aligned} \lim_{n \rightarrow \infty} F(m+1) &= \lim_{n \rightarrow \infty} \left(\ln((m+1) + y) + \sum_{\ell=1}^j \beta_\ell \frac{y^\ell}{\ell((m+1) + y)^\ell} \right) \\ &= \lim_{n \rightarrow \infty} \left(\ln n + \sum_{\ell=1}^j \beta_\ell \frac{(t+j-2)^\ell}{\ell n^\ell} \right) = \lim_{n \rightarrow \infty} \left(\ln n + \sum_{\ell=1}^j \beta_\ell \frac{c^\ell}{\ell} \right) \end{aligned}$$

and moreover

$$\begin{aligned} \lim_{n \rightarrow \infty} F(0) &= \lim_{n \rightarrow \infty} \left(\ln y + \sum_{\ell=1}^j \beta_\ell \frac{y^\ell}{\ell y^\ell} \right) = \lim_{n \rightarrow \infty} \left(\ln(t+j-2) + \sum_{\ell=1}^j \beta_\ell \frac{1}{\ell} \right) \\ &= \lim_{n \rightarrow \infty} \left(\ln t + \sum_{\ell=1}^j \beta_\ell \frac{1}{\ell} \right). \end{aligned}$$

Hence, $\lim_{n \rightarrow \infty} (F(m+1) - F(0)) = \ln \frac{1}{c} + \sum_{\ell=1}^j \beta_\ell \frac{c^\ell - 1}{\ell}$. It remains to consider $F(d) - F(d+1)$ in the limit of $n \rightarrow \infty$. It holds that

$$\begin{aligned} F(d) - F(d+1) &= \ln(d+y) + \sum_{\ell=1}^j \beta_\ell \frac{y^\ell}{\ell(d+y)^\ell} - \ln(d+1+y) - \sum_{\ell=1}^j \beta_\ell \frac{y^\ell}{\ell(d+1+y)^\ell} \\ &= \ln \left(\frac{d+y}{d+1+y} \right) + \sum_{\ell=1}^j \frac{\beta_\ell}{\ell} \left(\left(\frac{y}{d+y} \right)^\ell - \left(\frac{y}{d+1+y} \right)^\ell \right) \end{aligned}$$

and since $y = t+j-2 = \Theta(n)$ and $d = (j/r)y = \Theta(y)$, we get that $\lim_{n \rightarrow \infty} (F(d) - F(d+1)) = 0$.

Case B: $r \geq 2$. In this case let $F(i)$ and α_ℓ be defined according to Lemma A.4. Further, let $G(i) = -\alpha_0(r-1)F(i)$. Using Equation (7) we obtain

$$\begin{aligned} &= \frac{\kappa\tau}{n j! \alpha_0 (r-1)} (G(0) - G(m+1) + G(d+1) - G(d)) \\ &= \frac{\tau}{n(r-1)} (G(0) - G(m+1) + G(d+1) - G(d)), \end{aligned} \tag{8}$$

where the last equality follows from the definition of $\alpha_0 = \binom{j+r-1}{r-1} = \kappa/j!$. We first notice

$$\lim_{n \rightarrow \infty} \frac{\tau}{n(r-1)} = \frac{1}{r-1} \lim_{n \rightarrow \infty} \frac{(t-1)^{\underline{x}}}{n} = \frac{1}{r-1} \lim_{n \rightarrow \infty} \frac{(t-1)^r}{n} = \frac{1}{r-1} c^r \lim_{n \rightarrow \infty} n^{r-1}.$$

Further it holds that

$$G(m+1) = \frac{\sum_{\ell=0}^j \alpha_\ell (m+1)^{j-\ell} (t+j-2)^\ell}{(m+1+t+j-2)^{r+j-1}} = \frac{\sum_{\ell=0}^j \alpha_\ell (m+1)^{j-\ell} (t+j-2)^\ell}{n^{r+j-1}}.$$

18:16 New Results for the k -Secretary Problem

Note that $\lim_{n \rightarrow \infty} (m+1) = \lim_{n \rightarrow \infty} (n - (t-1)) = \lim_{n \rightarrow \infty} (n - cn) = \lim_{n \rightarrow \infty} (1-c)n$ and similarly $\lim_{n \rightarrow \infty} (t+j-2) = \lim_{n \rightarrow \infty} (t-1) = \lim_{n \rightarrow \infty} cn$. Hence we get

$$\lim_{n \rightarrow \infty} G(m+1) = \lim_{n \rightarrow \infty} \frac{\sum_{\ell=0}^j \alpha_\ell (1-c)^{j-\ell} n^{j-\ell} c^\ell n^\ell}{n^{r+j-1}} = \lim_{n \rightarrow \infty} \frac{\sum_{\ell=0}^j \alpha_\ell (1-c)^{j-\ell} c^\ell}{n^{r-1}}.$$

For the term $G(0)$ we obtain

$$G(0) = \frac{\sum_{\ell=0}^j \alpha_\ell 0^{j-\ell} y^\ell}{y^{r+j-1}} = \frac{\alpha_j y^j}{y^{r+j-1}} = \frac{1}{y^{r-1}}$$

and thus $\lim_{n \rightarrow \infty} G(0) = \lim_{n \rightarrow \infty} \frac{1}{y^{r-1}} = \lim_{n \rightarrow \infty} \frac{1}{(t-1)^{r-1}} = \frac{1}{c^{r-1}} \lim_{n \rightarrow \infty} \frac{1}{n^{r-1}}$.

In Equation (8) it remains to consider $G(d+1) - G(d)$. Similarly to case A we can show that this term approaches 0 for $n \rightarrow \infty$:

$$\begin{aligned} G(d+1) - G(d) &= \frac{\sum_{\ell=0}^j \alpha_\ell (d+1)^{j-\ell} y^\ell}{(d+1+y)^{r+j-1}} - \frac{\sum_{\ell=0}^j \alpha_\ell d^{j-\ell} y^\ell}{(d+y)^{r+j-1}} \\ &\leq \frac{\sum_{\ell=0}^j \alpha_\ell y^\ell ((d+1)^{j-\ell} - d^{j-\ell})}{(d+y)^{r+j-1}} \end{aligned}$$

where the numerator approaches 0 since $d = \Theta(y) = \Theta(n)$. Using Equation (8) and all limits stated above, we get finally

$$\begin{aligned} \lim_{n \rightarrow \infty} \Pr[E_j] &= \lim_{n \rightarrow \infty} \frac{1}{r-1} c^r n^{r-1} \left(\frac{1}{c^{r-1}} \frac{1}{n^{r-1}} - \frac{\sum_{\ell=0}^j \alpha_\ell (1-c)^{j-\ell} c^\ell}{n^{r-1}} \right) \\ &= \frac{1}{r-1} \left(c - \sum_{\ell=0}^j \alpha_\ell c^{r+\ell} (1-c)^{j-\ell} \right) \\ &= \frac{c}{r-1} - \frac{c^r (1-c)^j}{r-1} \sum_{\ell=0}^j \alpha_\ell \left(\frac{c}{1-c} \right)^\ell. \end{aligned}$$

This concludes the proof. \blacktriangleleft

► **Lemma A.2.** Let $f: \mathbb{R} \rightarrow \mathbb{R}$ with $f(i) = i^j / (i+y)^{r+j}$ and $j \geq 0$, $r \geq 1$, and $y > 0$ does not depend on i . The function f is monotonically increasing for $i \leq d$ and monotonically decreasing for $i > d$ where $d = (jy)/r$.

Proof. Let $g(i) = i^j$ and $h(i) = (i+y)^{r+j}$. We consider the first derivative $f'(i) = \frac{g'(i)h(i) - g(i)h'(i)}{h(i)^2}$. Since $h(i)^2$ is nonnegative, f grows monotonically if

$$g'(i)h(i) \geq g(i)h'(i) \Leftrightarrow j i^{j-1} (i+y)^{r+j} \geq i^j (r+j) (i+y)^{r+j-1} \Leftrightarrow j(i+y) \geq i(r+j).$$

It is easy to see that the last inequality is equivalent to $i \leq \frac{jy}{r} = d$. \blacktriangleleft

► **Lemma A.3.** Let $f: \mathbb{R} \rightarrow \mathbb{R}$ with $f(i) = i^j / (i+y)^{r+j}$ and $r = 1$, $j \geq 0$, and $y > 0$ does not depend on i . The following function F fulfills $F'(i) = f(i)$:

$$F(i) = \ln(i+y) + \sum_{\ell=1}^j \beta_\ell \frac{y^\ell}{\ell (i+y)^\ell}$$

where $\beta_\ell = (-1)^{\ell+1} \binom{j}{\ell}$ for $1 \leq \ell \leq j$.

Proof. We need to show $F'(i) = f(i)$ and observe first that

$$\begin{aligned} F'(i) &= \frac{1}{i+y} + \sum_{\ell=1}^j \beta_{\ell} \frac{-\ell y^{\ell}}{\ell(i+y)^{\ell+1}} = \frac{1}{i+y} + \sum_{\ell=1}^j \beta_{\ell} y^{\ell} \frac{-(i+y)^{j-\ell}}{(i+y)^{j+1}} \\ &= \frac{1}{(i+y)^{j+1}} \left((i+y)^j + \sum_{\ell=1}^j \beta_{\ell} y^{\ell} (-(i+y)^{j-\ell}) \right) \end{aligned}$$

and since $\beta_0 = (-1)^{0+1} \binom{j}{0} = -1$ we get further

$$F'(i) = \frac{1}{(i+y)^{j+1}} \sum_{\ell=0}^j \beta_{\ell} y^{\ell} (-(i+y)^{j-\ell}) = \frac{1}{(i+y)^{j+1}} \sum_{\ell=0}^j (-1)^{\ell+2} \binom{j}{\ell} y^{\ell} (i+y)^{j-\ell}.$$

Finally, note that $(-1)^{\ell+2} y^{\ell} = (-y)^{\ell}$, thus by the binomial theorem the last sum evaluates to $((i+y) + (-y))^j = i^j$ which concludes the proof. \blacktriangleleft

► **Lemma A.4.** Let $f: \mathbb{R} \rightarrow \mathbb{R}$ with $f(i) = i^j / (i+y)^{r+j}$ and $j \geq 0$, $r \geq 2$, and $y > 0$ does not depend on i . The following function F fulfills $F'(i) = f(i)$:

$$F(i) = -\frac{\sum_{\ell=0}^j \alpha_{\ell} i^{j-\ell} y^{\ell}}{\alpha_0 (r-1) (i+y)^{r+j-1}},$$

where $\alpha_{\ell} = \binom{j+r-1}{\ell+r-1}$ for $0 \leq \ell \leq j$.

Proof. Let $G(i)$ and $H(i)$ be the numerator and denominator of $F(i)$. It holds that $G'(i) = -\sum_{\ell=0}^j \alpha_{\ell} (j-\ell) i^{j-\ell-1} y^{\ell}$ and $H'(i) = \alpha_0 (r-1) (r+j-1) (i+y)^{r+j-2} = H(i) r(i)$ where $r(i) = \frac{r+j-1}{i+y}$. In order to prove the claim, we show

$$G'(i)(i+y) - G(i)(r+j-1) = i^j \alpha_0 (r-1) \quad (9)$$

since then we have

$$\begin{aligned} F'(i) &= \frac{G'(i)H(i) - G(i)H'(i)}{H(i)^2} = \frac{G'(i) - G(i)r(i)}{H(i)} = \frac{G'(i) - G(i)r(i)}{\frac{\alpha_0 (r-1)}{i+y} (i+y)^{r+j}} \\ &= \frac{(i+y)(G'(i) - G(i)r(i))}{\alpha_0 (r-1) (i+y)^{r+j}} = \frac{(i+y)G'(i) - (r+j-1)G(i)}{\alpha_0 (r-1) (i+y)^{r+j}} \end{aligned}$$

With Equation (9), the last term resolves to $= \frac{i^j \alpha_0 (r-1)}{\alpha_0 (r-1) (i+y)^{r+j}} = f(i)$. It remains to show Equation (9):

$$\begin{aligned} &G'(i)(i+y) - G(i)(r+j-1) \\ &= -\left(\sum_{\ell=0}^j \alpha_{\ell} (j-\ell) i^{j-\ell-1} y^{\ell} \right) (i+y) + \left(\sum_{\ell=0}^j \alpha_{\ell} i^{j-\ell} y^{\ell} \right) (r+j-1) \\ &= -\left(\sum_{\ell=0}^j \alpha_{\ell} (j-\ell) i^{j-\ell} y^{\ell} \right) - \left(\sum_{\ell=0}^j \alpha_{\ell} (j-\ell) i^{j-\ell-1} y^{\ell+1} \right) \\ &\quad + \left(\sum_{\ell=0}^j \alpha_{\ell} i^{j-\ell} y^{\ell} \right) (r+j-1) \\ &= \left(\sum_{\ell=0}^j \alpha_{\ell} i^{j-\ell} y^{\ell} (r-1+\ell) \right) - \left(\sum_{\ell=0}^{j-1} \alpha_{\ell} (j-\ell) i^{j-\ell-1} y^{\ell+1} \right). \end{aligned}$$

18:18 New Results for the k -Secretary Problem

Note that the first sum contains all powers of i from i^0 to i^j , while the latter sum only powers from i^0 to i^{j-1} . Therefore, we can split up the part for i^j from the first sum and group equal powers of i to obtain

$$\alpha_0(r-1)i^j + \sum_{\ell=1}^j (\alpha_\ell(r-1+\ell) - \alpha_{\ell-1}(j-\ell+1)) i^{j-\ell} y^\ell.$$

The claim follows if we can show that the last sum evaluates to zero. This is true, since by definition of α_ℓ it holds that

$$\begin{aligned} \alpha_\ell(r-1+\ell) &= \binom{j+r-1}{\ell+r-1} (r-1+\ell) = \frac{(j+r-1)!}{(\ell+r-1)!(j-\ell)!} (r-1+\ell) \\ &= \frac{(j+r-1)!}{(\ell+r-2)!(j-\ell+1)!} \frac{(j-\ell+1)}{(j-\ell)!} = \binom{j+r-1}{(\ell-1)+r-1} (j-\ell+1) = \alpha_{\ell-1}(j-\ell+1). \quad \blacktriangleleft \end{aligned}$$

■ **Table 3** Optimal parameters and corresponding competitive ratios of SINGLE-REF for $k \in [1..100]$. For readability, the numeric values are truncated after the fourth decimal place.

k	r	c	competitive ratio	k	r	c	competitive ratio
1	1	0.3678	0.3678	51	10	0.1662	0.7082
2	1	0.2545	0.4119	52	10	0.1635	0.7098
3	2	0.3475	0.4449	53	10	0.1608	0.7113
4	2	0.2928	0.4785	54	10	0.1582	0.7127
5	2	0.2525	0.4999	55	10	0.1557	0.7141
6	2	0.2217	0.5148	56	10	0.1532	0.7155
7	3	0.2800	0.5308	57	10	0.1509	0.7168
8	3	0.2549	0.5453	58	10	0.1486	0.7180
9	3	0.2338	0.5567	59	11	0.1597	0.7193
10	3	0.2159	0.5660	60	11	0.1574	0.7206
11	4	0.2570	0.5740	61	11	0.1551	0.7219
12	4	0.2410	0.5834	62	11	0.1529	0.7231
13	4	0.2267	0.5914	63	11	0.1508	0.7243
14	4	0.2140	0.5983	64	11	0.1487	0.7255
15	4	0.2026	0.6043	65	11	0.1467	0.7266
16	4	0.1924	0.6096	66	11	0.1447	0.7277
17	5	0.2231	0.6155	67	11	0.1428	0.7287
18	5	0.2133	0.6211	68	12	0.1527	0.7298
19	5	0.2042	0.6261	69	12	0.1508	0.7309
20	5	0.1959	0.6306	70	12	0.1489	0.7320
21	5	0.1882	0.6347	71	12	0.1470	0.7330
22	5	0.1811	0.6384	72	12	0.1452	0.7340
23	6	0.2054	0.6426	73	12	0.1434	0.7350
24	6	0.1985	0.6465	74	12	0.1417	0.7360
25	6	0.1919	0.6502	75	12	0.1400	0.7369
26	6	0.1858	0.6535	76	12	0.1384	0.7378
27	6	0.1800	0.6566	77	13	0.1473	0.7387
28	6	0.1746	0.6595	78	13	0.1456	0.7397
29	7	0.1947	0.6625	79	13	0.1440	0.7406
30	7	0.1893	0.6655	80	13	0.1424	0.7415
31	7	0.1842	0.6684	81	13	0.1408	0.7424
32	7	0.1793	0.6711	82	13	0.1393	0.7433
33	7	0.1747	0.6736	83	13	0.1378	0.7441
34	7	0.1703	0.6760	84	13	0.1363	0.7449
35	7	0.1662	0.6782	85	13	0.1349	0.7457
36	8	0.1830	0.6805	86	14	0.1429	0.7465
37	8	0.1788	0.6829	87	14	0.1415	0.7473
38	8	0.1748	0.6851	88	14	0.1400	0.7482
39	8	0.1710	0.6873	89	14	0.1386	0.7490
40	8	0.1673	0.6893	90	14	0.1372	0.7497
41	8	0.1638	0.6912	91	14	0.1359	0.7505
42	8	0.1605	0.6930	92	14	0.1346	0.7512
43	9	0.1750	0.6948	93	14	0.1333	0.7520
44	9	0.1716	0.6968	94	14	0.1320	0.7527
45	9	0.1683	0.6986	95	14	0.1307	0.7534
46	9	0.1651	0.7004	96	15	0.1381	0.7541
47	9	0.1621	0.7021	97	15	0.1368	0.7548
48	9	0.1592	0.7037	98	15	0.1356	0.7555
49	9	0.1563	0.7052	99	15	0.1343	0.7562
50	9	0.1536	0.7067	100	15	0.1331	0.7569

Triangle Estimation Using Tripartite Independent Set Queries

Anup Bhattacharya

Indian Statistical Institute, Kolkata, India

Arijit Bishnu

Indian Statistical Institute, Kolkata, India

Arijit Ghosh

Indian Statistical Institute, Kolkata, India

Gopinath Mishra

Indian Statistical Institute, Kolkata, India

Abstract

Estimating the number of triangles in a graph is one of the most fundamental problems in sublinear algorithms. In this work, we provide an approximate triangle counting algorithm using only polylogarithmic queries when *the number of triangles on any edge in the graph is polylogarithmically bounded*. Our query oracle *Tripartite Independent Set* (TIS) takes three disjoint sets of vertices A , B and C as input, and answers whether there exists a triangle having one endpoint in each of these three sets. Our query model generally belongs to the class of *group queries* (Ron and Tsur, ACM ToCT, 2016; Dell and Lapinskas, STOC 2018) and in particular is inspired by the *Bipartite Independent Set* (BIS) query oracle of Beame et al. (ITCS 2018). We extend the algorithmic framework of Beame et al., with TIS replacing BIS, for triangle counting using ideas from color coding due to Alon et al. (J. ACM, 1995) and a concentration inequality for sums of random variables with bounded dependency (Janson, Rand. Struct. Alg., 2004).

2012 ACM Subject Classification Theory of computation \rightarrow Models of computation; Theory of computation \rightarrow Streaming, sublinear and near linear time algorithms; Mathematics of computing \rightarrow Probabilistic algorithms

Keywords and phrases Triangle estimation, query complexity, sublinear algorithm

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.19

Related Version A full version of the paper is available at <https://arxiv.org/abs/1808.00691>.

1 Introduction

Counting the number of triangles in a graph is a fundamental algorithmic problem in the RAM model [4, 10, 22], streaming [1, 2, 5, 11, 13, 24, 25, 26, 27, 28, 33] and the query model [17, 21]. In this work, we provide the first approximate triangle counting algorithm using only polylogarithmic queries to a query oracle named *Tripartite Independent Set* (TIS).

Notations, the query model, the problem and the result

We denote the set $\{1, \dots, n\}$ by $[n]$. Let $V(G)$, $E(G)$ and $T(G)$ denote the set of vertices, edges and triangles in the input graph G , respectively. Let $t(G) = |T(G)|$. The statement A, B, C are disjoint, means A, B, C are pairwise disjoint. For three non-empty disjoint sets $A, B, C \subseteq V(G)$, $G(A, B, C)$, termed as a *tripartite subgraph* of G , denotes the induced subgraph of $A \cup B \cup C$ in G minus the edges having both endpoints in A or B or C . $t(A, B, C)$ denotes the number of triangles in $G(A, B, C)$. We use the triplet (a, b, c) to denote the triangle having a, b, c as its vertices. Let Δ_u denote the number of triangles having u as one of its vertices. Let $\Delta_{(u,v)}$ be the number of triangles having (u, v) as one of its edges



© Anup Bhattacharya, Arijit Bishnu, Arijit Ghosh, and Gopinath Mishra; licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 19; pp. 19:1–19:17

Leibniz International Proceedings in Informatics



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

19:2 Triangle Estimation Using Tripartite Independent Set Queries

and $\Delta_E = \max_{(u,v) \in E(G)} \Delta_{(u,v)}$. For a set \mathcal{U} , “ \mathcal{U} is COLORED with $[n]$ ”, means that each member of \mathcal{U} is assigned a color out of $[n]$ colors independently and uniformly at random. Let $\mathbb{E}[X]$ and $\mathbb{V}[X]$ denote the expectation and variance of a random variable X . For an event \mathcal{E} , \mathcal{E}^c denotes the complement of \mathcal{E} . The statement “ a is an $1 \pm \epsilon$ multiplicative approximation of b ” means $|b - a| \leq \epsilon \cdot b$. Next, we describe the query oracle.

Tripartite independent set oracle (TIS). Given three non-empty disjoint subsets $V_1, V_2, V_3 \subseteq V(G)$ of a graph G , TIS query oracle answers “YES” if and only if $t(V_1, V_2, V_3) \neq 0$.

Notice that the query oracle looks at only those triangles that have vertices in all of these sets V_1, V_2, V_3 . The TRIANGLE-ESTIMATION problem is to report an $1 \pm \epsilon$ multiplicative approximation of $t(G)$ where the input is $V(G)$, TIS oracle for graph G and $\epsilon \in (0, 1)$.

► **Theorem 1.** *Let G be a graph with $\Delta_E \leq d$, $|V(G)| = n \geq 64$. For any $\epsilon > 0$, TRIANGLE-ESTIMATION can be solved using $\mathcal{O}(\epsilon^{-12} d^{12} \log^{25} n)$ TIS queries with probability $1 - \mathcal{O}(n^{-2})$.*

Note that the query complexity stated in Theorem 1 is $\text{poly}(\log n, \frac{1}{\epsilon})$, even if d is $\mathcal{O}(\log^c n)$, where c is a positive constant. We reiterate that the only bound we require is on the number of triangles on an edge; neither do we require any bound on the maximum degree of the graph, nor do we require any bound on the number of triangles incident on a vertex.

Query models and TIS

Query models for graphs are essentially of two types:

Local Queries: This query model was initiated by Feige [19] and Goldreich and Ron [20] and used even recently by [17, 18]. The queries on the graphs are (i) degree query: the oracle reports the degree of a vertex; (ii) neighbor query: the oracle reports the i^{th} neighbor of v , if it exists; and (iii) edge existence query: the oracle reports whether there exists an edge between a given pair of vertices.

Group Queries or Subset queries and Subset samples: These queries were implicitly initiated in the works of Stockmeyer [31, 32] and formalized by Ron and Tsur [29]. *Group queries* can be viewed as a generalization of membership queries in sets. The essential idea of the group queries is to estimate the size of an unknown set $S \subseteq U$ by using a YES/NO answer from the oracle to the existence of an intersection between sets S and $T \subseteq U$; and give a uniformly selected item of $S \cap T$, if $S \cap T \neq \emptyset$ in the *subset sample* query. Subset sample queries are at least as powerful as group queries. The *cut query* by Rubinfeld et al. [30], though motivated by submodular function minimization problem, can also be seen in the light of group queries – we seek the number of edges that intersect both the vertex sets that form a cut. Choi and Kim [12] used a variation of group queries for *graph reconstruction*. Dell and Lapinskas [14] essentially used this class of queries for estimating the number of edges in a bipartite graph. *Bipartite independent set* (BIS) queries for a graph, initiated by Beame et al. [6], can also be seen in the light of group queries. It provides a YES/NO answer to the existence of an edge in $E(G)$ that intersects with both $V_1, V_2 \subset V(G)$ of G , where V_1 and V_2 are disjoint. A subset sample version of BIS oracle was used in [9].

In TIS, we seek a YES/NO answer about the existence of an intersection between the set of triangles, that we want to estimate, and three disjoint sets of vertices. Thus TIS belongs to the class of group queries, as does BIS. A bone of contention for any newly introduced

query oracle is its worth¹. Beame et al. [6] had given a subjective justification in favor of BIS to establish it as a query oracle. It is easy to verify that TIS, being in the same class of group queries, have the interesting connections to group testing and computational geometry as BIS. We provide justifications in favor of considering $\Delta_E \leq d$ in Appendix A. Intuitively, TIS is to triangle counting what BIS is to edge estimation.

Prior works

Eden et al. [17] used $\tilde{O}\left(\frac{|V(G)|}{\hat{t}(G)^{1/3}} + \min\left\{\frac{|E(G)|^{3/2}}{\hat{t}(G)}, |E(G)|\right\}\right)^2$ local queries to estimate the number of triangles. Their algorithmic results aided by an almost matching lower bound have almost closed this line of study. Matching upper and lower bounds on k -clique counting in G using local query model have also been reported [18]. A precursor to triangle estimation in graphs is edge estimation. The number of edges in a graph can be estimated by using $\tilde{O}(n/\sqrt{m})$ many degree and neighbor queries, and $\Omega(n/\sqrt{m})$ queries are necessary to estimate the number of edges even if we allow all the three local queries [20]. This result would almost have closed the edge estimation problem but for having a relook at the problem with stronger query models and hoping for polylogarithmic number of queries. Beame et al. [6] precisely did that by estimating the number of edges in a graph using $\mathcal{O}(\epsilon^{-4} \log^{14} n)$ *bipartite independent set* (BIS) queries. Motivated by this result, we explore whether triangle estimation can be solved using only polylogarithmic queries to TIS.

Organization of the paper

We give a broad overview of the algorithm in Section 2. Sections 3, 4 and 5 give the details of sparsification, exact estimation and *coarse* estimation of the number of triangles, respectively. The final algorithm is given in Section 6. Appendix A provides justifications in favor of TIS. Appendix B has the probabilistic results used in this paper.

► **Remark 1.** Note that the TRIANGLE-ESTIMATION can also be thought of as HYPEREDGE ESTIMATION problem in a 3-uniform hypergraph. Very recently, Dell et al. [15] and Bhattacharya et al. [8], independently, showed that the bound on Δ_E is not necessary to solve TRIANGLE-ESTIMATION by using polylogarithmic many TIS queries. Also, both Dell et al. [15] and Bhattacharya et al. [8], independently, generalized our result to c -uniform hypergraphs, where $c \in \mathbb{N}$ is a constant.

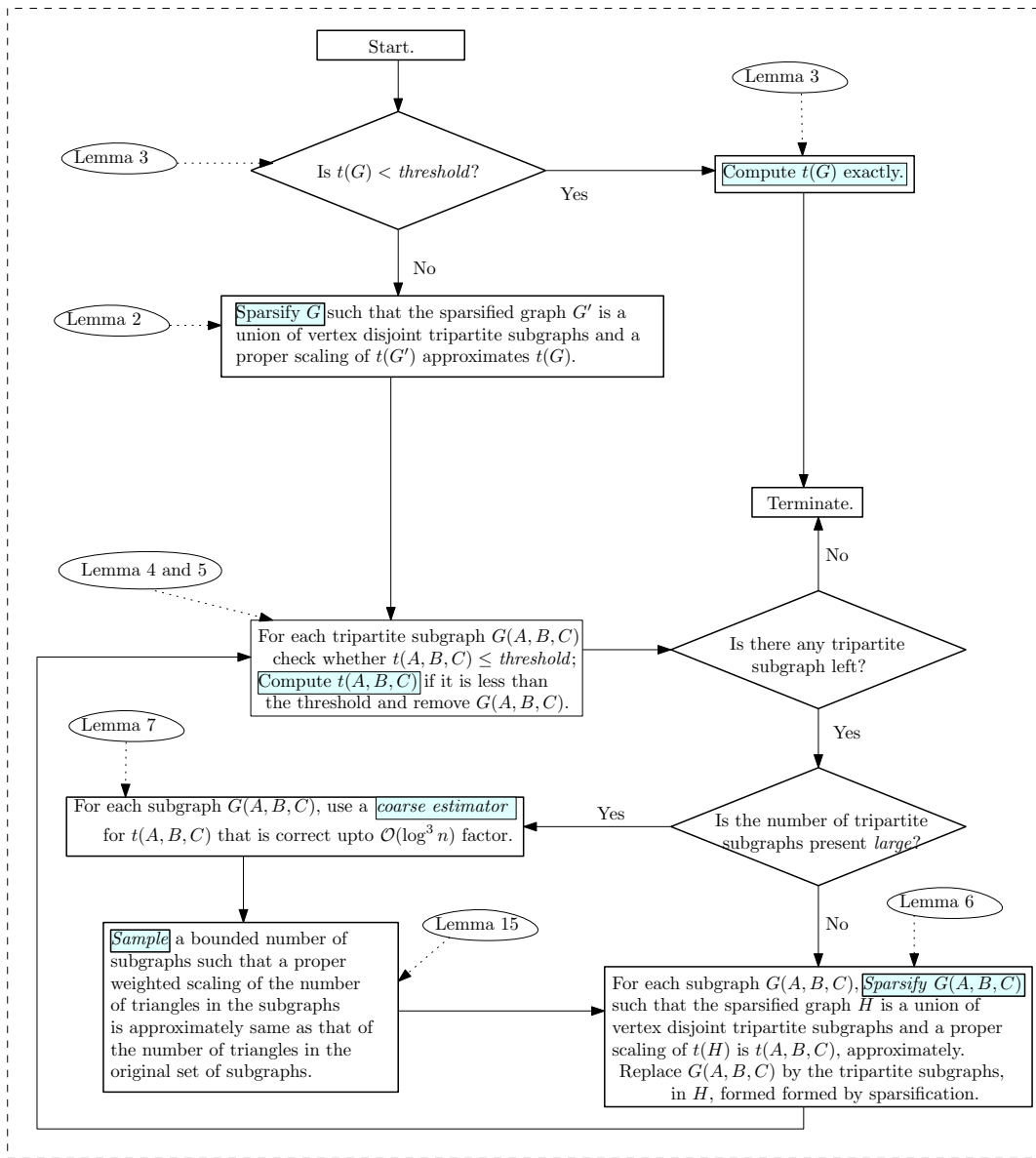
2 Overview of the algorithm

Our algorithmic framework is inspired by [6] but the detailed analysis is markedly different using *color coding* due to Alon et al. [3] and a relatively new concentration inequality, due to Janson [23], for handling sums of random variables with bounded dependency. Apart from Lemmas 5 and 13, all other proofs require different ideas.

We feel that the analysis in [6] does not go through in our case because of a subtle difference between counting the number of triangles and the number of edges in a graph. An edge is an explicit structure, whereas, a triangle is an implicit structure – triangles (a, b, x) , (b, c, y) and (a, c, z) in G imply the existence of the triangle (a, b, c) in G .

¹ See <http://www.wisdom.weizmann.ac.il/~oded/MC/237.html> for a comment on BIS.

² $\tilde{O}(\cdot)$ hides a polynomial factor of $\log n$ and $\frac{1}{\epsilon}$, where $\epsilon \in (0, 1)$ is such that $(1 - \epsilon)t \leq \hat{t} \leq (1 + \epsilon)t$; \hat{t} and t denote the estimated and actual number of triangles in G , respectively.



■ **Figure 1** Flow chart of the algorithm. The highlighted texts indicate the basic building blocks of the algorithm. We also indicate the corresponding lemmas that support the building blocks.

In Figure 1, we give a *flowchart* of the algorithm and show the corresponding lemmas that support the steps of the algorithm. The main idea of our algorithm is as follows. We can figure out for a given G , if the number of triangles $t(G)$ is greater than or equal to a threshold τ (Lemma 3). If $t(G) < \tau$, i.e., G is sparse in triangles, we compute $t(G)$ exactly (Lemma 3). Otherwise, we sparsify G to get a disjoint union of tripartite subgraphs of G that maintain $t(G)$ up to a scaling factor (Lemma 2). For each tripartite subgraph, if the subgraph is sparse (decided by Lemma 4), we count the number of triangles exactly (Lemma 5). Otherwise, we again sparsify (Lemma 6). This repeated process of sparsification may create a huge number of tripartite subgraphs. Counting the number of triangles in them is managed by doing a coarse estimation (Lemma 7) and taking a sample of the subgraph

that maintains the number of triangles *approximately*. Each time we sparsify, we ensure that the sum of the number of triangles in the subgraphs generated by sparsification is a constant fraction of the number of triangles in the graph before sparsification, making the number of iterations $O(\log n)$.

We sparsify G by considering the partition obtained when $V(G)$ is COLORED with $[3k]$. This sparsification is done such that: (i) the sparsified graph is a union of a set of vertex disjoint tripartite subgraphs and (ii) a proper scaling of the number of triangles in the sparsified graph is a *good* estimate of $t(G)$ with high probability³. The proof of the sparsification result stated next uses the *method of averaged bounded differences* and *Chernoff-Hoeffding type* inequality in bounded dependency setting by Janson [23]. The detailed proof is in Section 3. Recall that Δ_E is the maximum number of triangles on a particular edge.

► **Lemma 2 (General Sparsification).** *Let $k, d \in \mathbb{N}$. There exists a constant κ_1 such that for any graph G with $\Delta_E \leq d$, if V_1, \dots, V_{3k} is a random partition of $V(G)$ obtained by $V(G)$ being COLORED with $[3k]$, then*

$$\mathbb{P} \left(\left| \frac{9k^2}{2} \sum_{i=1}^k t(V_i, V_{k+i}, V_{2k+i}) - t(G) \right| > \kappa_1 dk^2 \sqrt{t(G) \log n} \right) \leq \frac{2}{n^4}.$$

We apply the sparsification corresponding to Lemma 2 only when $t(G)$ is above a threshold⁴ to ensure that the relative error is bounded. We can decide whether $t(G)$ is less than the threshold and if it is so, we compute the exact value of $t(G)$, using the following lemma, whose proof is inspired by color coding ideas [9] and given in Section 4.

► **Lemma 3 (Exact Counting).** *There exists an algorithm that for any graph G and a threshold parameter $\tau \in \mathbb{N}$, determines whether $t(G) < \tau$ using $\mathcal{O}(\tau^6 \log n)$ TIS queries with probability $1 - n^{-10}$. Moreover, the algorithm finds the exact value of $t(G)$ if $t(G) < \tau$.*

Assume that $t(G)$ is *large*⁵ and G has undergone sparsification. We initialize a data structure with a set of vertex disjoint tripartite graphs that are obtained after the sparsification step. For each tripartite graph $G(A, B, C)$ in the data structure, we check whether $t(A, B, C)$ is less than a threshold using the algorithm corresponding to Lemma 4. If it is less than a threshold, we compute the exact value of $t(A, B, C)$ using Lemma 5 and remove $G(A, B, C)$ from the data structure. The proofs of Lemma 4 and 5 are given in the full version [7].

► **Lemma 4 (Threshold for Tripartite Graph).** *There exists a deterministic algorithm that given any disjoint subsets $A, B, C \subset V(G)$ of any graph G and a threshold parameter $\tau \in \mathbb{N}$, can decide whether $t(A, B, C) \leq \tau$ using $\mathcal{O}(\tau \log n)$ TIS queries.*

► **Lemma 5 (Exact Counting in Tripartite Graphs).** *There exists a deterministic algorithm that given any disjoint subsets $A, B, C \subset V(G)$ of any graph G , can determine the exact value of $t(A, B, C)$ using $\mathcal{O}(t(A, B, C) \log n)$ TIS queries.*

Now we are left with some tripartite graphs such that the number of triangles in each graph is more than a threshold. If the number of such graphs is not large, then we sparsify each tripartite graph $G(A, B, C)$ in a fashion almost similar to the earlier sparsification. This

³ High probability means that the probability of success is at least $1 - \frac{1}{n^c}$ for some constant c .

⁴ The threshold is a fixed polynomial in $d, \log n$ and $\frac{1}{\epsilon}$.

⁵ Large refers to a fixed polynomial in $d, \log n$ and $\frac{1}{\epsilon}$.

19:6 Triangle Estimation Using Tripartite Independent Set Queries

sparsification result formally stated in the following Lemma, has a proof similar to Lemma 2. We replace $G(A, B, C)$ by a constant (say, k)⁶ many tripartite subgraphs formed after sparsification.

► **Lemma 6** (Sparsification for Tripartite Graphs). *Let $k, d \in \mathbb{N}$. There exists a constant κ_2 such that*

$$\mathbb{P} \left(\left| k^2 \sum_{i=1}^k t(A_i, B_i, C_i) - t(A, B, C) \right| > \kappa_2 d k^2 \sqrt{t(G)} \log n \right) \leq \frac{1}{n^8}$$

where A, B and C are disjoint subsets of $V(G)$ for any graph G with $\Delta_E \leq d$, and $A_1, \dots, A_k, B_1, \dots, B_k$ and C_1, \dots, C_k are the partitions of A, B, C formed uniformly at random, respectively.

If we have a large number of vertex disjoint tripartite subgraphs of G and each subgraph contains a large number of triangles, then we *coarsely* estimate the number of triangles in each subgraph which is correct up to $\mathcal{O}(\log^3 n)$ factor by using the algorithm corresponding to the following Lemma, whose proof is in Section 5. Our COARSE-ESTIMATE algorithm is similar in structure to the coarse estimation algorithm for edge estimation, but the analysis involves sophisticated calculations.

► **Lemma 7** (Coarse Estimation). *There exists an algorithm that given disjoint subsets $A, B, C \subset V(G)$ of any graph G , returns an estimate \hat{t} satisfying*

$$\frac{t(A, B, C)}{64 \log n} \leq \hat{t} \leq 64 t(A, B, C) \log^3 n$$

with probability $1 - n^{-9}$. Moreover, the query complexity of the algorithm is $\mathcal{O}(\log^4 n)$.

After estimating the number of triangles in each subgraph coarsely, we generate a bounded number of samples of the set of subgraphs using a sampling technique given by Beame et al. [6]. The sampling maintains the triangle count approximately. The Lemma corresponding to sampling is formally stated in Lemma 13 in Section 6. After getting the sample, we apply the sparsification algorithm corresponding to Lemma 6 for each subgraph in the sample.

Now again, for each tripartite graph $G(A, B, C)$, we check whether $t(A, B, C)$ is less than a threshold using the algorithm corresponding to Lemma 4. If yes, then we can compute the exact value of $t(A, B, C)$ using Lemma 5 and remove $G(A, B, C)$ from the data structure. Otherwise, we iterate on all the required steps discussed above as shown in Figure 1. Observe that the query complexity of each iteration is polylogarithmic⁷. Now, note that the number of triangles reduces by a constant factor after each sparsification step. So, the number of iterations is bounded by $\mathcal{O}(\log n)$. Hence, the query complexity of our algorithm is polylogarithmic. This completes the high level description of our algorithm.

⁶ In our algorithm, k is a constant. However, Lemma 6 holds for any $k \in \mathbb{N}$.

⁷ Polylogarithmic refers to a polynomial in $d, \log n$ and $\frac{1}{\epsilon}$

3 Sparsification Lemma

In this Section, we prove Lemma 2. The proof of Lemma 6 is similar.

Proof of Lemma 2. $V(G)$ is COLORED with $[3k]$. Let V_1, \dots, V_{3k} be the resulting partition of $V(G)$. Let Z_i be the random variable that denotes the color assigned to the i^{th} vertex. For $i \in [3k]$, $\pi(i)$ is a set of three colors defined as follows: $\pi(i) = \{i, (1 + (i + k - 1) \bmod 3k), (1 + (i + 2k - 1) \bmod 3k)\}$.

► **Definition 8.** A triangle (a, b, c) is said to be properly colored if there exists a bijection in terms of coloring from $\{a, b, c\}$ to $\pi(i)$.

Let $f(Z_1, \dots, Z_n) = \sum_{i=1}^k t(V_i, V_{k+i}, V_{2k+i})$. Note that f is the number of triangles that are properly colored. The probability that a triangle is properly colored is $\frac{2}{9k^2}$. So, $\mathbb{E}[f] = \frac{2t(G)}{9k^2}$.

Let us focus on the instance when vertices $1, \dots, t-1$ are already colored and we are going to color vertex t . Let $S_\ell(S_r)$ be the set of triangles in G having t as one of the vertices and other two vertices are from $[t-1]$ ($[n] \setminus [t]$). $S_{\ell r}$ be the set of triangles in G such that t is a vertex and the second and third vertices are from $[t-1]$ and $[n] \setminus [t]$, respectively.

Given that the vertex t is colored with color $c \in [3k]$, let $N_\ell^c, N_r^c, N_{\ell r}^c$ be the random variables that denote the number of triangles in S_ℓ, S_r and $S_{\ell r}$ that are properly colored, respectively. Now, we can deduce the following about \mathbb{E}_f^t , the difference in the conditional expectation of the number of triangles that are properly colored whose t^{th} vertex is (possibly) differently colored, by considering the vertices in S_ℓ, S_r and $S_{\ell r}$ separately.

$$\begin{aligned} \mathbb{E}_f^t &= |\mathbb{E}[f \mid Z_1, \dots, Z_{t-1}, Z_t = a_t] - \mathbb{E}[f \mid Z_1, \dots, Z_{t-1}, Z_t = a'_t]| \\ &= \left| N_\ell^{a_t} - N_\ell^{a'_t} + \mathbb{E} \left[N_r^{a_t} - N_r^{a'_t} \right] + \mathbb{E} \left[N_{\ell r}^{a_t} - N_{\ell r}^{a'_t} \right] \right| \\ &\leq \left| N_\ell^{a_t} - N_\ell^{a'_t} \right| + \mathbb{E} \left[\left| N_r^{a_t} - N_r^{a'_t} \right| \right] + \mathbb{E} \left[\left| N_{\ell r}^{a_t} - N_{\ell r}^{a'_t} \right| \right] \end{aligned}$$

Now, consider the following claim, whose proof can be found in the full version [7].

- **Claim 9.** (a) $\mathbb{P}(|N_\ell^{a_t} - N_\ell^{a'_t}| < 8\sqrt{d\Delta_t \log n}) \geq 1 - 4n^{-8}$;
 (b) $\mathbb{E}[|N_r^{a_t} - N_r^{a'_t}|] \leq \sqrt{d\Delta_t}/k$;
 (c) $\mathbb{E}[|N_{\ell r}^{a_t} - N_{\ell r}^{a'_t}|] < 6d\sqrt{\Delta_t \log n}$.⁸

Let $c_t = 15d\sqrt{\Delta_t \log n}$. From the above claim, we have

$$\mathbb{E}_f^t < 8\sqrt{d\Delta_t \log n} + \frac{\sqrt{d\Delta_t}}{k} + 6d\sqrt{\Delta_t \log n} \leq 15d\sqrt{\Delta_t \log n} = c_t$$

with probability at least $1 - \frac{4}{n^8}$. Let \mathcal{B} be the event that there exists $t \in [n]$ such that $\mathbb{E}_f^t > c_t$. By the union bound over all $t \in [n]$, $\mathbb{P}(\mathcal{B}) \leq \frac{4}{n^7}$.

Using the method of *averaged bounded difference* [16] (See Lemma 15 in Appendix B), we have

$$\mathbb{P}(|f - \mathbb{E}[f]| > \delta + t(G)\mathbb{P}(\mathcal{B})) \leq e^{-\delta^2 / \sum_{t=1}^n c_t^2} + \mathbb{P}(\mathcal{B}).$$

⁸ Note that Δ_t is the number of triangles having t as one of its vertices and we are not assuming any bound on Δ_t . We assume Δ_E , that is number of triangles on any edge, is bounded.

19:8 Triangle Estimation Using Tripartite Independent Set Queries

We set $\delta = 60d\sqrt{t(G)}\log n$. Observe that $\sum_{t=1}^n c_t^2 = 225d^2 \log n \sum_{t=1}^n \Delta_t = 675d^2 t(G) \log n$. Hence,

$$\mathbb{P}\left(\left|f - \frac{2t(G)}{9k^2}\right| > 60d\sqrt{t(G)}\log n + t(G)\mathbb{P}(\mathcal{B})\right) \leq \frac{1}{n^4} + \frac{1}{n^7},$$

that is,

$$\mathbb{P}\left(\left|\frac{9k^2}{2}f - t(G)\right| > 270dk^2\sqrt{t(G)}\log n + \frac{9k^2}{2} \cdot \frac{t(G)}{n^7}\right) \leq \frac{1}{n^4} + \frac{1}{n^7}.$$

Since, $\frac{9k^2}{2} \cdot \frac{t(G)}{n^7} < dk^2\sqrt{t(G)}\log n$, we get

$$\mathbb{P}\left(\left|\frac{9k^2}{2}f - t(G)\right| > 271dk^2\sqrt{t(G)}\log n\right) \leq \frac{2}{n^4}. \quad \blacktriangleleft$$

4 Proof of the Lemmas corresponding to exact estimation

In this Section, we prove Lemma 3. The proofs of 4 and 5 can be found in the full version [7].

Proof of Lemma 3. We color $V(G)$ with $[100\tau^2]$ colors. Let $h : V(G) \rightarrow [100\tau^2]$ be the coloring function and $V_i = \{v \in V(G) : h(v) = i\}$, i.e., the vertices with color i , where $i \in [100\tau^2]$. Note that $V_1, \dots, V_{100\tau^2}$ forms a partition of $V(G)$. We make TIS queries with input V_i, V_j, V_k for each $1 \leq i < j < k \leq 100\tau^2$. Observe that we make $\mathcal{O}(\tau^6)$ TIS queries. We construct a 3-uniform hypergraph \mathcal{H} , where $U(\mathcal{H}) = \{V_1, \dots, V_{100\tau^2}\}$ ⁹ and $(V_i, V_j, V_k) \in \mathcal{F}(\mathcal{H})$ if and only if TIS oracle answers yes with V_i, V_j, V_k given as input. We repeat the above procedure γ times, where $\gamma = 50 \log n$. Let $\mathcal{H}_1, \dots, \mathcal{H}_\gamma$ be the set of corresponding hypergraphs and h_i be the coloring function to form the hypergraph \mathcal{H}_i , where $i \in [\gamma]$. Then we compute $A = \max\{|\mathcal{F}(\mathcal{H}_1)|, \dots, |\mathcal{F}(\mathcal{H}_\gamma)|\}$. If $A \geq \tau$, we report $t(G) \geq \tau$. Otherwise, we report A as $t(G)$. Note that the total number of TIS queries is $\mathcal{O}(\tau^6 \log n)$. Now, we analyze the cases $t(G) \geq \tau$ and $t(G) < \tau$ separately.

(i) $t(G) \geq \tau$: Consider a fixed set T of τ triangles. Let T_v be the set of vertices that is present in some triangle in T . Observe that $|T_v| \leq 3\tau$. Let \mathcal{E}_i be the event that the vertices in T_v are uniquely colored by the function h_i , i.e., $\mathcal{E}_i : h_i(u) = h_i(v)$ if and only if $u = v$, where $u, v \in T_v$. First we prove that $\mathbb{P}(\mathcal{E}) \geq \frac{9}{10}$ by computing $\mathbb{P}(\mathcal{E}_i^c)$.

$$\mathbb{P}(\mathcal{E}_i^c) \leq \sum_{u, v \in T_v} \mathbb{P}(h_i(u) = h_i(v)) \leq \sum_{u, v \in T_v} \frac{1}{100\tau^2} \leq \frac{|T_v|^2}{100\tau^2} < \frac{1}{10}.$$

Let PROP_i be the property that for each triangle $z \in T$, there is a corresponding hyperedge in $\mathcal{F}(\mathcal{H}_i)$, where $i \in [\gamma]$. Specifically, for each triangle $(a_1, a_2, a_3) \in T$ there exists a hyperedge $(a'_1, a'_2, a'_3) \in \mathcal{F}(\mathcal{H}_i)$ such that $h_i(a_j) = h_i(a'_j)$ for each $j \in [3]$. Note that, if PROP_i holds, then $|\mathcal{F}(\mathcal{H}_i)| \geq |T| \geq \tau$. By the definition of TIS oracle, PROP_i holds when the event \mathcal{E}_i occurs, i.e., PROP_i holds with probability at least $\frac{9}{10}$. This implies, with probability $\frac{9}{10}$, $|\mathcal{F}(\mathcal{H}_i)| \geq \tau$. Recall that $A = \max\{|\mathcal{F}(\mathcal{H}_1)|, \dots, |\mathcal{F}(\mathcal{H}_\gamma)|\}$ and $\gamma = 50 \log n$. So, $\mathbb{P}(A < \tau) = \left(1 - \frac{9}{10}\right)^{50 \log n} \leq \frac{1}{n^{10}}$. Hence, if $t(G) \geq \tau$, our algorithm detects it with probability at least $1 - \frac{1}{n^{10}}$.

⁹ $U(\mathcal{H})$ and $\mathcal{F}(\mathcal{H})$ denote the set of vertices and hyperedges in a hypergraph \mathcal{H} , respectively.

(ii) $t(G) < \tau$: Let T be the set of all $t(G)$ triangles in G and T_v be the set of vertices that is present in some triangle in T . Observe that $|T_v| \leq 3 \cdot t(G) < 3\tau$. Let \mathcal{E}_i be the event that the vertices in T_v are uniquely colored by the function h_i , i.e., $\mathcal{E}_i : h_i(u) = h_i(v)$ if and only if $u = v$, where $u, v \in T_v$. First we prove that $\mathbb{P}(\mathcal{E}_i) \geq \frac{9}{10}$ by computing $\mathbb{P}(\mathcal{E}_i^c)$.

$$\mathbb{P}(\mathcal{E}_i^c) \leq \sum_{u,v \in T_v} \mathbb{P}(h_i(u) = h_i(v)) \leq \sum_{u,v \in T_v} \frac{1}{100\tau^2} \leq \frac{|T_v|^2}{100\tau^2} < \frac{1}{10}.$$

Let PROP_i be the property that for each triangle $z \in T$, there is a corresponding hyperedge in $\mathcal{F}(\mathcal{H}_i)$, where $i \in [\gamma]$. Specifically, for each triangle $(a_1, a_2, a_3) \in T$ there exists a hyperedge $(a'_1, a'_2, a'_3) \in \mathcal{F}(\mathcal{H}_i)$ such that $h_i(a_j) = h_i(a'_j)$ for each $j \in [3]$. Note that, if PROP_i holds, then $|\mathcal{F}(\mathcal{H}_i)| = t(G)$. By the definition of TIS oracle, PROP_i holds when the event \mathcal{E}_i occurs, i.e., PROP_i holds with probability at least $\frac{9}{10}$. This implies, with probability $\frac{9}{10}$, $|\mathcal{F}(\mathcal{H}_i)| = t(G)$. Recall that $A = \max\{|\mathcal{F}(\mathcal{H}_1), \dots, \mathcal{F}(\mathcal{H}_\gamma)|\}$ and $\gamma = 50 \log n$. By the construction of \mathcal{H}_i , $|\mathcal{F}(\mathcal{H}_i)| \leq t(G)$. So, $A \leq t(G)$ and $\mathbb{P}(A \neq t(G)) = \mathbb{P}(A < t(G)) \leq (1 - \frac{9}{10})^{50 \log n} \leq \frac{1}{n^{10}}$. Hence, if $t(G) < \tau$, our algorithm outputs the exact value of $t(G)$ with probability at least $1 - \frac{1}{n^{10}}$. \blacktriangleleft

5 Proof of the Lemma corresponding to coarse estimation

We now prove Lemma 7. Algorithm 2 corresponds to Lemma 7. Algorithm 1 is a subroutine in Algorithm 2. Algorithm 1 determines whether a given estimate \hat{t} is correct upto a $\mathcal{O}(\log^3 n)$ factor. Lemmas 10 and 11 are intermediate results needed to prove Lemma 7.

Algorithm 1 VERIFY-ESTIMATE (A, B, C, \hat{t}) .

Input: Three pairwise disjoint set $A, B, C \subseteq V(G)$ and \hat{t} .
Output: If \hat{t} is a good estimate, then ACCEPT. Otherwise, REJECT.

```

1 begin
2   for ( $i = 2 \log n$  to 0) do
3     for ( $j = \log n$  to 0) do
4       Find  $A_{ij} \subseteq A, B_{ij} \subseteq B, C_{ij} \subseteq C$  by sampling each element of  $A, B$  and  $C$ ,
           respectively with probability  $\min\{\frac{2^i}{\hat{t}}, 1\}, \min\{\frac{2^j}{2^i} \log n, 1\}, \frac{1}{2^j}$ ,
           respectively.
5       if ( $t(A_{ij}, B_{ij}, C_{ij}) \neq 0$ ) then
6         ACCEPT
7   REJECT

```

Lemma 10. If $\hat{t} \geq 64t(A, B, C) \log^3 n$, $\mathbb{P}(\text{VERIFY-ESTIMATE}(A, B, C, \hat{t}) \text{ accepts}) \leq \frac{1}{20}$.

Proof. Let $T(A, B, C)$ denote the set of triangles having vertices $a \in A, b \in B$ and $c \in C$, where A, B and C are disjoint subsets of $V(G)$. For $(a, b, c) \in T(A, B, C)$ such that $a \in A, b \in B, c \in C$, let $X_{(a,b,c)}^{ij}$ denote the indicator random variable such that $X_{(a,b,c)}^{ij} = 1$ if and only if $(a, b, c) \in T(A_{ij}, B_{ij}, C_{ij})$ and $X_{ij} = \sum_{(a,b,c) \in T(A,B,C)} X_{(a,b,c)}^{ij}$. Note that $t(A_{ij}, B_{ij}, C_{ij}) = X_{ij}$. (a, b, c) is present in $T(A_{ij}, B_{ij}, C_{ij})$ if $a \in A_{ij}, b \in B_{ij}$ and $c \in C_{ij}$. So,

$$\mathbb{P}(X_{(a,b,c)}^{ij} = 1) \leq \frac{2^i}{\hat{t}} \cdot \frac{2^j}{2^i} \log n \cdot \frac{1}{2^j} = \frac{\log n}{\hat{t}} \text{ and } \mathbb{E}[X_{ij}] \leq \frac{t(A, B, C)}{\hat{t}} \log n.$$

19:10 Triangle Estimation Using Tripartite Independent Set Queries

As $X_{ij} \geq 0$,

$$\mathbb{P}(X_{ij} \neq 0) = \mathbb{P}(X_{ij} \geq 1) \leq \mathbb{E}[X_{ij}] \leq \frac{t(A, B, C)}{\hat{t}} \log n.$$

Now using the fact that $\hat{t} \geq 64t(A, B, C) \log^3 n$, we have $\mathbb{P}(X_{ij} \neq 0) \leq \frac{1}{64 \log^2 n}$. Observe that VERIFY-ESTIMATE accepts if and only if there exists $i, j \in \{0, \dots, \log n\}$ such that $X_{ij} \neq 0$. Using the union bound, we get

$$\begin{aligned} \mathbb{P}(\text{VERIFY-ESTIMATE accepts}) &\leq \sum_{0 \leq i \leq 2 \log n} \sum_{0 \leq j \leq \log n} \mathbb{P}(X_{ij} \neq 0) \\ &\leq \frac{(2 \log n + 1)(\log n + 1)}{32 \log^2 n} \\ &\leq \frac{1}{20}. \quad \blacktriangleleft \end{aligned}$$

► **Lemma 11.** *If $\hat{t} \leq \frac{t(A, B, C)}{32 \log n}$, $\mathbb{P}(\text{VERIFY-ESTIMATE}(A, B, C, \hat{t}) \text{ accepts}) \geq \frac{1}{5}$.*

Proof. For $p \in \{0, \dots, 2 \log n\}$, let $A^p \subseteq A$ be the set of vertices such that for each $a \in A^p$, the number of triangles of the form (a, b, c) with $(b, c) \in B \times C$, lies between 2^p and $2^{p+1} - 1$.

For $a \in A^p$ and $q \in \{0, \dots, \log n\}$, let $B^{pq}(a) \subseteq B$ is the set of vertices such that for each $b \in B$, the number of triangles of the form (a, b, c) with $c \in C$ lies between 2^q and $2^{q+1} - 1$. We need the following Claim to proceed further.

▷ **Claim 12.**

- (i) There exists $p \in \{0, \dots, 2 \log n\}$ such that $|A^p| > \frac{t(A, B, C)}{2^{p+1}(2 \log n + 1)}$.
- (ii) For each $a \in A^p$, there exists $q \in \{0, \dots, \log n\}$ such that $|B^{pq}(a)| > \frac{2^p}{2^{q+1}(\log n + 1)}$.

Proof.

- (i) Observe that $t(A, B, C) = \sum_{p=0}^{2 \log n} t(A^p, B, C)$ as the sum takes into account all incidences of vertices in A . So, there exists $p \in \{0, \dots, 2 \log n\}$ such that $t(A^p, B, C) \geq \frac{t(A, B, C)}{2 \log n + 1}$. From the definition of A^p , $t(A^p, B, C) < |A^p| \cdot 2^{p+1}$. Hence, there exists $p \in \{0, \dots, 2 \log n\}$ such that

$$|A^p| > \frac{t(A^p, B, C)}{2^{p+1}} \geq \frac{t(A, B, C)}{2^{p+1}(2 \log n + 1)}.$$

- (ii) Observe that $\sum_{q=0}^{\log n} t(\{a\}, B^{pq}(a), C) = t(\{a\}, B, C)$. So, there exists $q \in \{0, \dots, \log n\}$ such that $t(\{a\}, B^{pq}(a), C) \geq \frac{t(\{a\}, B, C)}{\log n + 1}$. From the definition of $B^{pq}(a)$, $t(\{a\}, B^{pq}(a), C) < |B^{pq}(a)| \cdot 2^{q+1}$. Hence, there exists $q \in \{0, \dots, \log n\}$ such that

$$|B^{pq}(a)| > \frac{t(\{a\}, B^{pq}(a), C)}{2^{q+1}} \geq \frac{t(\{a\}, B, C)}{2^{q+1}(\log n + 1)} \geq \frac{2^p}{2^{q+1}(\log n + 1)}. \quad \blacktriangleleft$$

We come back to the proof of Lemma 11. We will show that VERIFY-ESTIMATE accepts with probability at least $\frac{1}{5}$ when loop executes for $i = p$, where p is such that $|A^p| > \frac{t(A, B, C)}{2^{p+1}(2 \log n + 1)}$. The existence of such a p is evident from (i) of Claim 12.

Recall that $A_{pq} \subseteq A, B_{pq} \subseteq B$ and $C_{pq} \subseteq C$ are the samples obtained when the loop variables i and j in Algorithm 1 attain values p and q , respectively. Observe that

$$\mathbb{P}(A_{pq} \cap A^p = \emptyset) \leq \left(1 - \frac{2^p}{\hat{t}}\right)^{|A^p|} \leq e^{-\frac{2^p}{\hat{t}}|A^p|} \leq e^{-\frac{2^p}{\hat{t}} \frac{t(A, B, C)}{2^{p+1} \log n}} = e^{-\frac{t(A, B, C)}{2\hat{t}(2 \log n + 1)}}.$$

Now using the fact that $\hat{t} \leq \frac{t(A,B,C)}{32 \log n}$ and $n \geq 64$,

$$\mathbb{P}(A_{pq} \cap A^p = \emptyset) \leq \frac{1}{e^6}.$$

Assume that $A_{pq} \cap A^p \neq \emptyset$ and $a \in A_{pq} \cap A^p$. By (ii) of Claim 12, there exists $q \in \{0, \dots, \log n\}$, such that $B^{pq}(a) \geq \frac{2^p}{2^{q+1}(\log n + 1)}$. Observe that we will be done, if we can show that VERIFY-ESTIMATE accepts when loop executes for $i = p$ and $j = q$. Now,

$$\mathbb{P}(B_{pq} \cap B^{pq}(a) = \emptyset \mid A_{pq} \cap A^p \neq \emptyset) \leq \left(1 - \frac{2^q}{2^p} \log n\right)^{|B^{pq}(a)|} \leq \frac{1}{e^{3/7}}.$$

Assume that $A_{pq} \cap A^p \neq \emptyset$, $B_{pq} \cap B^{pq}(a) \neq \emptyset$ and $b \in B_{pq} \cap B^{pq}(a)$. Let S be the set such that (a, b, s) is a triangle in G for each $s \in S$. Note that $|S| \geq 2^q$. So,

$$\mathbb{P}(C_{pq} \cap S = \emptyset \mid A_{pq} \cap A^p \neq \emptyset \text{ and } B_{pq} \cap B^{pq}(a) \neq \emptyset) \leq \left(1 - \frac{1}{2^q}\right)^{2^q} \leq \frac{1}{e}.$$

Observe that VERIFY-ESTIMATE accepts if $t(A_{pq}, B_{pq}, C_{pq}) \neq 0$. Also, $t(A_{pq}, B_{pq}, C_{pq}) \neq 0$ if $A_{pq} \cap A^p \neq \emptyset$, $B_{pq} \cap B^{pq}(a) \neq \emptyset$ and $C_{pq} \cap S \neq \emptyset$. Hence,

$$\begin{aligned} \mathbb{P}(\text{VERIFY-ESTIMATE accepts}) &\geq \mathbb{P}(A_{pq} \cap A^p \neq \emptyset, B_{pq} \cap B^{pq}(a) \neq \emptyset \text{ and } C_{pq} \cap S \neq \emptyset) \\ &= \mathbb{P}(A_{pq} \cap A^p \neq \emptyset) \cdot \mathbb{P}(B_{pq} \cap B^{pq}(a) \neq \emptyset \mid A_{pq} \cap A^p \neq \emptyset) \\ &\quad \cdot \mathbb{P}(C_{pq} \cap S \neq \emptyset \mid A_{pq} \cap A^p \neq \emptyset \text{ and } B_{pq} \cap B^{pq}(a) \neq \emptyset) \\ &> \left(1 - \frac{1}{e^6}\right) \left(1 - \frac{1}{e^{3/7}}\right) \left(1 - \frac{1}{e}\right) > \frac{1}{5}. \quad \blacktriangleleft \end{aligned}$$

■ **Algorithm 2** COARSE-ESTIMATE (A, B, C) .

Input: Three pairwise disjoint sets $A, B, C \subset V(G)$.

Output: An estimate \hat{t} for $t(A, B, C)$.

```

1 begin
2   for ( $\hat{t} = n^3, n^3/2, \dots, 1$ ) do
3     Repeat VERIFY-ESTIMATE  $(A, B, C, \hat{t})$  for  $\Gamma = 2000 \log n$  times. If at least  $\frac{\Gamma}{10}$ 
       many VERIFY-ESTIMATE accepts, then output  $\hat{t}$ .

```

Proof of Lemma 7. Note that an execution of COARSE-ESTIMATE for a particular \hat{t} , repeats VERIFY-ESTIMATE for $\Gamma = 2000 \log n$ times and gives output \hat{t} if at least $\frac{\Gamma}{10}$ many VERIFY-ESTIMATE accepts. For a particular \hat{t} , let X_i be the indicator random variable such that $X_i = 1$ if and only if the i^{th} execution of VERIFY-ESTIMATE accepts. Also take $X = \sum_{i=1}^{\Gamma} X_i$. COARSE-ESTIMATE gives output \hat{t} if $X > \frac{\Gamma}{10}$.

Consider the execution of COARSE-ESTIMATE for a particular \hat{t} . If $\hat{t} \geq 32t(A, B, C) \log^3 n$, we first show that COARSE-ESTIMATE accepts with probability $1 - \frac{1}{n^5}$. Recall Lemma 10. If $\hat{t} \geq 64t(A, B, C) \log^3 n$, $\mathbb{P}(X_i = 1) \leq \frac{1}{20}$ and hence $\mathbb{E}[X] \leq \frac{\Gamma}{20}$. By using Chernoff-Hoeffding's inequality (See (i) of Lemma 17 in Appendix B),

$$\mathbb{P}\left(X > \frac{\Gamma}{10}\right) = \mathbb{P}\left(X > \frac{\Gamma}{20} + \frac{\Gamma}{20}\right) \leq \frac{1}{n^{10}}.$$

By using the union bound for all \hat{t} , the probability that COARSE-ESTIMATE outputs some $\hat{t} \geq 16t(A, B, C) \log^3 n$, is at most $\frac{3 \log n}{n^{10}}$.

19:12 Triangle Estimation Using Tripartite Independent Set Queries

Now consider the instance when the for loop in COARSE-ESTIMATE executes for a \hat{t} such that $\hat{t} \leq \frac{t(A,B,C)}{32 \log n}$. In this situation, $\mathbb{P}(X_i = 1) \geq \frac{1}{5}$. So, $\mathbb{E}[X] \geq \frac{\Gamma}{5}$. By using Chernoff-Hoeffding's inequality (See (ii) of Lemma 17 in Appendix B),

$$\mathbb{P}\left(X \leq \frac{\Gamma}{10}\right) \leq \mathbb{P}\left(X < \frac{3\Gamma}{20}\right) = \mathbb{P}\left(X < \frac{\Gamma}{5} - \frac{\Gamma}{20}\right) \leq \frac{1}{n^{10}}.$$

By using the union bound for all \hat{t} , the probability that COARSE-ESTIMATE outputs some $\hat{t} \leq \frac{t(A,B,C)}{16 \log n}$, is at most $\frac{3 \log n}{n^{10}}$.

Observe that, COARSE-ESTIMATE gives output \hat{t} that satisfies either $\hat{t} \geq 64t(A, B, C) \log^3 n$ or $\hat{t} \leq \frac{t(A,B,C)}{32 \log n}$ is at most $\frac{3 \log n}{n^{10}} + \frac{3 \log n}{n^{10}} \leq \frac{1}{n^9}$.

Putting everything together, COARSE-ESTIMATE gives some \hat{t} as output with probability at least $1 - \frac{1}{n^9}$ satisfying

$$\frac{t(A, B, C)}{64 \log n} \leq \hat{t} \leq 64t(A, B, C) \log^3 n.$$

From the description of VERIFY-ESTIMATE and COARSE-ESTIMATE, the query complexity of VERIFY-ESTIMATE is $\mathcal{O}(\log^2 n)$ and COARSE-ESTIMATE calls VERIFY-ESTIMATE $\mathcal{O}(\log^2 n)$ times. Hence, COARSE-ESTIMATE makes $\mathcal{O}(\log^4 n)$ many queries. \blacktriangleleft

6 The final triangle estimation algorithm: Proof of Theorem 1

Now we design our algorithm for $1 \pm \epsilon$ multiplicative approximation of $t(G)$. If $\epsilon \leq \frac{d \log^2 n}{n^{1/4}}$, we query for $t(\{a\}, \{b\}, \{c\})$ for all distinct $a, b, c \in V(G)$ and compute the exact value of $t(G)$. So, we assume that $\epsilon > \frac{d \log^2 n}{n^{1/4}}$.

We build a data structure such that it maintains two things at any point of time. (i) An accumulator ψ for the number of triangles. We initialize $\psi = 0$. (ii) A set of tuples $(A_1, B_1, C_1, w_1), \dots, (A_\zeta, B_\zeta, C_\zeta, w_\zeta)$, where tuple (A_i, B_i, C_i) corresponds to the tripartite subgraph $G(A_i, B_i, C_i)$ and w_i is the weight associated to $G(A_i, B_i, C_i)$. Initially, there is no tuple in our data structure. The algorithm will proceed as follows.

- (1) **(Exact Counting)** Fix the threshold τ as $\frac{36\kappa_1^2 d^2 \log^4 n}{\epsilon^2}$. Decide whether $t(G) < \tau$ by using the result of Lemma 3, where κ_1 is the constant mentioned in Lemma 2. If yes, we terminate by reporting the exact value of $t(G)$. Otherwise, we go to Step-2. The query complexity of Step-1 is $\mathcal{O}(\tau^6 \log n) = \mathcal{O}\left(\frac{d^{12} \log^{25} n}{\epsilon^{12}}\right)$.
- (2) **(General Sparsification)** $V(G)$ is COLORED with $[3k]$ for $k = 1$. Let A, B, C be the partition generated by the coloring of $V(G)$. We initialize the data structure by setting $\psi = 0$ and adding the tuple $(A, B, C, 9/2)$ to the data structure. Note that no query is required in this step. The constant $9/2$ is obtained by putting $k = 1$ in Lemma 2.
- (3) We repeat steps 4 to 7 until there is no tuple left in the data structure. We maintain an invariant that the number of tuples stored in the data structure, is at most $\frac{10\kappa_3 \log^{16} n}{\epsilon^2}$, where κ_3 is a constant to be fixed later.
- (4) **(Threshold for Tripartite Graph and Exact Counting in Tripartite Graphs)** For each tuple (A, B, C, w) in the data structure, we determine whether $t(A, B, C) \leq \frac{36\kappa_2^2 d^2 \log^4 n}{\epsilon^2}$, the threshold, by using the deterministic algorithm corresponding to Lemma 3 with $\mathcal{O}\left(\frac{d^2 \log^4 n}{\epsilon^2} \cdot \log n\right) = \mathcal{O}\left(\frac{d^2 \log^5 n}{\epsilon^2}\right)$ many queries, where κ_2 is the constant mentioned in Lemma 6. If yes, we find $t(A, B, C)$ using $\mathcal{O}\left(\frac{d^2 \log^5 n}{\epsilon^2}\right)$ many queries and add $w \cdot t(A, B, C)$ to ψ . We remove all (A, B, C) 's for which the algorithm found that $t(A, B, C)$ is below the threshold. As there are at most $\mathcal{O}\left(\frac{\log^{16} n}{\epsilon^2}\right)$ many triples at any time, the number of queries made in each iteration of the algorithm is $\mathcal{O}\left(\frac{d^2 \log^5 n}{\epsilon^2} \cdot \frac{\log^{16} n}{\epsilon^2}\right) = \mathcal{O}\left(\frac{d^2 \log^{21} n}{\epsilon^4}\right)$.

- (5) Note that each tuple (A, B, C, w) in this step is such that $t(A, B, C) > \frac{36\kappa_2^2 d^2 \log^4 n}{\epsilon^2}$. Let $(A_1, B_1, C_1, w_1), \dots, (A_r, B_r, C_r, w_r)$ be the set of tuples stored at the current instant. If $r > \frac{10\kappa_3 \log^{16} n}{\epsilon^2}$, we go to Step 6. Otherwise, we go to Step 7.
- (6) **(Coarse Estimation and Sampling)** For each tuple (A, B, C, w) in the data structure, we find an estimate \hat{t} such that $\frac{t(A, B, C)}{64 \log^3 n} < \hat{t} < 64t(A, B, C) \log^3 n$. This can be done due to Lemma 7 and the number of queries is $\mathcal{O}(\log^4 n)$ per tuple. As the algorithm executes the current step, the number of tuples in our data structure is *large*. We take a sample from the set of tuples such that the sample maintains the required estimate *approximately* by using Lemma 13, that follows from a Lemma by Beame et al. [6]. The original statement of Beame et al. is given in Lemma 20 in Appendix B.
- **Lemma 13** ([6]). *Let $(A_1, B_1, C_1, w_1), \dots, (A_r, B_r, C_r, w_r)$ be the tuples present in the data structure and e_i be the corresponding coarse estimation for $t(A_i, B_i, C_i)$, $i \in [r]$, such that (i) $w_i, e_i \geq 1, \forall i \in [r]$; (ii) $\frac{e_i}{\rho} \leq t(A_i, B_i, C_i) \leq e_i \rho$ for some $\rho > 0$ and $\forall i \in [r]$; and (iii) $\sum_{i=1}^r w_i \cdot t(A_i, B_i, C_i) \leq M$. Note that the exact values $t(A_i, B_i, C_i)$'s are not known to us. Then there exists an algorithm that finds $(A'_1, B'_1, C'_1, w'_1), \dots, (A'_s, B'_s, C'_s, w'_s)$ such that all of the above three conditions hold and $\left| \sum_{i=1}^s w'_i \cdot t(A'_i, B'_i, C'_i) - \sum_{i=1}^r w_i \cdot t(A_i, B_i, C_i) \right| \leq \lambda S$ with probability $1 - \delta$; where $S = \sum_{i=1}^r w_i \cdot t(A_i, B_i, C_i)$ and $\lambda, \delta > 0$. Also, $s = \mathcal{O}(\lambda^{-2} \rho^4 \log M (\log \log M + \log \frac{1}{\delta}))$. We use the algorithm corresponding to Lemma 13 with $\lambda = \frac{\epsilon}{61 \log n}$, $\rho = 64 \log^3 n$ and $\delta = \frac{1}{n^{10}}$ to find a new set of tuples $(A'_1, B'_1, C'_1, w'_1), \dots, (A'_s, B'_s, C'_s, w'_s)$ such that $|S - \sum_{i=1}^s w'_i t(A'_i, B'_i, C'_i)| \leq \lambda S$ with probability $1 - \frac{1}{n^{10}}$, where $S = \sum_{i=1}^r w_i t(A_i, B_i, C_i)$ and $s = \frac{\kappa_3 \log^{16} n}{\epsilon^2}$ for some constant $\kappa_3 > 0$. This κ_3 is same as the one mentioned in Step 3. No query is required to execute the algorithm of Lemma 13. Recall that the number of tuples present at any time is $\mathcal{O}\left(\frac{\log^{16} n}{\epsilon^2}\right)$. Hence, the number of queries in this step in each iteration, is $\mathcal{O}\left(\frac{\log^{16} n}{\epsilon^2} \cdot \log^4 n\right) = \mathcal{O}\left(\frac{\log^{20} n}{\epsilon^2}\right)$.*
- (7) **(Sparsification for Tripartite Graphs)** We partition each of A, B and C into 3 parts uniformly at random. Let $A = U_1 \uplus U_2 \uplus U_3$; $V = V_1 \uplus V_2 \uplus V_3$ and $W = W_1 \uplus W_2 \uplus W_3$. We delete (A, B, C, w) from the data structure and add $(U_i, V_i, W_i, 9w)$ for each $i \in [3]$ to our data structure. Note that no query is made in this step.
- (8) Report ψ as the estimate for the number of triangles in G , when no tuples are left.

First, we prove that the above algorithm produces a $(1 \pm \epsilon)$ multiplicative approximation to $t(G)$ for any $\epsilon > 0$ with high probability. If $t(G) \leq \frac{36\kappa_1^2 d^2 \log^4 n}{\epsilon^2}$, then the algorithm terminates in Step-1 and reports the exact number of triangles with probability $1 - \frac{1}{n^{10}}$ by Lemma 3. Otherwise, the algorithm proceeds to Step-2. In Step-2, the algorithm colors $V(G)$ using three colors and incurs a multiplicative error of $1 \pm \epsilon_0$ to $t(G)$, where $\epsilon_0 = \frac{\kappa_1 d \log n}{\sqrt{t(G)}}$. As $t(G) > \frac{36\kappa_1^2 d^2 \log^4 n}{\epsilon^2}$ and $n \geq 64$, $\epsilon_0 \leq \lambda = \frac{\epsilon}{61 \log n}$. Note that the algorithm possibly performs Step-4 to Step-7 multiple times, but not more than $O(\log n)$ times, as explained below.

Let $(A_1, B_1, C_1, w_1), \dots, (A_\zeta, B_\zeta, C_\zeta, w_\zeta)$ are the set of tuples present in the data structure currently. We define $\sum_{i=1}^\zeta t(A_i, B_i, C_i)$ as the number of active triangles. Let ACTIVE_i be the number of triangles that are active in the i^{th} iteration. Note that $\text{ACTIVE}_1 \leq t(G) \leq n^3$. By Lemma 6 and Step-7, observe that $\text{ACTIVE}_{i+1} \leq \frac{\text{ACTIVE}_i}{2}$. So, after $3 \log n$ many iterations there will be at most constant number of active triangles and then we can compute the exact number of active triangles and add it to ψ . In each iteration, there can be a multiplicative error of $1 \pm \lambda$ in Step-5 and $1 \pm \epsilon_0$ due to Step-4. So, using the fact that $\epsilon_0 \leq \lambda$, the multiplicative approximation factor lies between $(1 - \lambda)^{3 \log n + 1}$ and $(1 + \lambda)^{3 \log n + 1}$. As $\lambda = \frac{\epsilon}{61 \log n}$, the required approximation factor is $1 \pm \epsilon$.

19:14 Triangle Estimation Using Tripartite Independent Set Queries

The query complexity of Step 1 is $\mathcal{O}(\epsilon^{-12}d^{12}\log^{25}n)$. The query complexity of Steps 4 to 6 is $\mathcal{O}(\epsilon^{-4}\log^{21}n)$ in each iteration and the total number of iterations is $\mathcal{O}(\log n)$. Hence, the total query complexity of the algorithm is $\mathcal{O}(\epsilon^{-12}d^{12}\log^{25}n)$.

Now, we bound the failure probability of the algorithm. The algorithm can fail in Step-1 with probability at most $\frac{1}{n^{10}}$, Step-2 with probability at most $\frac{2}{n^4}$, Step-6 with probability at most $\frac{10\kappa_3\log^{16}n}{\epsilon^4} \cdot \frac{1}{n^9} + \frac{1}{n^{10}}$, and Step-7 with probability at most $\frac{10\kappa_3\log^{16}n}{\epsilon^4} \cdot \frac{1}{n^8}$. As the algorithm might execute Steps 4 to 6 for $3\log n$ times, the total failure probability is bounded by $\frac{1}{n^{10}} + \frac{2}{n^4} + 3\log n \left(\frac{10\kappa_3\log^{16}n}{\epsilon^4} \cdot \frac{1}{n^8} + \frac{10\kappa_3\log^{16}n}{\epsilon^4} \cdot \frac{1}{n^9} + \frac{1}{n^{10}} \right) \leq \frac{c}{n^2}$. Note that the above inequality holds because $\epsilon > \frac{d\log^2 n}{n^{1/4}}$ and $n \geq 64$.

References

- 1 Nesreen K Ahmed, Nick Duffield, Jennifer Neville, and Ramana Kompella. Graph sample and hold: A framework for big-graph analytics. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1446–1455. ACM, 2014.
- 2 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI symposium on Principles of Database Systems*, pages 5–14. ACM, 2012.
- 3 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.
- 4 Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997.
- 5 Ziv Bar-Yossef, Ravi Kumar, and D Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 623–632. Society for Industrial and Applied Mathematics, 2002.
- 6 Paul Beame, Sariel Har-Peled, Sivaramkrishnan Natarajan Ramamoorthy, Cyrus Rashtchian, and Makrand Sinha. Edge Estimation with Independent Set Oracles. In *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, pages 38:1–38:21, 2018. doi:10.4230/LIPIcs.ITCS.2018.38.
- 7 Anup Bhattacharya, Arijit Bishnu, Arijit Ghosh, and Gopinath Mishra. Triangle Estimation using Tripartite Independent Set Queries. *CoRR*, abs/1808.00691, 2018. arXiv:1808.00691.
- 8 Anup Bhattacharya, Arijit Bishnu, Arijit Ghosh, and Gopinath Mishra. Hyperedge Estimation using Polylogarithmic Subset Queries. *arXiv preprint*, 2019. arXiv:1908.04196.
- 9 Arijit Bishnu, Arijit Ghosh, Sudeshna Kolay, Gopinath Mishra, and Saket Saurabh. Parameterized Query Complexity of Hitting Set Using Stability of Sunflowers. In *29th International Symposium on Algorithms and Computation, ISAAC 2018, December 16-19, 2018, Jiaoxi, Yilan, Taiwan*, pages 25:1–25:12, 2018.
- 10 Andreas Björklund, Rasmus Pagh, Virginia Vassilevska Williams, and Uri Zwick. Listing triangles. In *International Colloquium on Automata, Languages, and Programming*, pages 223–234. Springer, 2014.
- 11 Luciana S Buriol, Gereon Frahling, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Christian Sohler. Counting triangles in data streams. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 253–262. ACM, 2006.
- 12 Sung-Soon Choi and Jeong Han Kim. Optimal query complexity bounds for finding graphs. *Artificial Intelligence*, 174(9-10):551–569, 2010.
- 13 Graham Cormode and Hossein Jowhari. A second look at counting triangles in graph streams (corrected). *Theoretical Computer Science*, 683:22–30, 2017.

- 14 Holger Dell and John Lapinskas. Fine-grained reductions from approximate counting to decision. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 281–288. ACM, 2018.
- 15 Holger Dell, John Lapinskas, and Kitty Meeks. Approximately counting and sampling small witnesses using a colourful decision oracle. *arXiv preprint*, 2019. [arXiv:1907.04826](https://arxiv.org/abs/1907.04826).
- 16 Devdatt P Dubhashi and Alessandro Panconesi. *Concentration of measure for the analysis of randomized algorithms*. Cambridge University Press, 2009.
- 17 Talya Eden, Amit Levi, Dana Ron, and C Seshadhri. Approximately counting triangles in sublinear time. *SIAM Journal on Computing*, 46(5):1603–1646, 2017.
- 18 Talya Eden, Dana Ron, and C Seshadhri. On approximating the number of k -cliques in sublinear time. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 722–734. ACM, 2018.
- 19 Uriel Feige. On sums of independent random variables with unbounded variance and estimating the average degree in a graph. *SIAM Journal on Computing*, 35(4):964–984, 2006.
- 20 Oded Goldreich and Dana Ron. Approximating average parameters of graphs. *Random Structures & Algorithms*, 32(4):473–493, 2008.
- 21 Mira Gonen, Dana Ron, and Yuval Shavitt. Counting stars and other small subgraphs in sublinear-time. *SIAM Journal on Discrete Mathematics*, 25(3):1365–1411, 2011.
- 22 Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. *SIAM Journal on Computing*, 7(4):413–423, 1978.
- 23 Svante Janson. Large deviations for sums of partly dependent random variables. *Random Structures & Algorithms*, 24(3):234–248, 2004.
- 24 Madhav Jha, Comandur Seshadhri, and Ali Pinar. A space efficient streaming algorithm for triangle counting using the birthday paradox. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 589–597. ACM, 2013.
- 25 Hossein Jowhari and Mohammad Ghodsi. New streaming algorithms for counting triangles in graphs. In *International Computing and Combinatorics Conference*, pages 710–716. Springer, 2005.
- 26 John Kallaugher and Eric Price. A hybrid sampling scheme for triangle counting. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1778–1797. Society for Industrial and Applied Mathematics, 2017.
- 27 Daniel M Kane, Kurt Mehlhorn, Thomas Sauerwald, and He Sun. Counting arbitrary subgraphs in data streams. In *International Colloquium on Automata, Languages, and Programming*, pages 598–609. Springer, 2012.
- 28 A. Pavan, Kanat Tangwongsan, Srikanta Tirthapura, and Kun-Lung Wu. Counting and Sampling Triangles from a Graph Stream. *PVLDB*, 6(14):1870–1881, 2013.
- 29 Dana Ron and Gilad Tsur. The power of an example: Hidden set size approximation using graph queries and conditional sampling. *ACM Transactions on Computation Theory (TOCT)*, 8(4):15, 2016.
- 30 Aviad Rubinfeld, Tselil Schramm, and S. Matthew Weinberg. Computing Exact Minimum Cuts Without Knowing the Graph. In *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, pages 39:1–39:16, 2018.
- 31 Larry Stockmeyer. The complexity of approximate counting. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 118–126. ACM, 1983.
- 32 Larry Stockmeyer. On approximation algorithms for $\#P$. *SIAM Journal on Computing*, 14(4):849–861, 1985.
- 33 Kanat Tangwongsan, Aduri Pavan, and Srikanta Tirthapura. Parallel triangle counting in massive streaming graphs. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 781–786. ACM, 2013.

A Scenario where Δ_E is bounded

In this Section, we discuss some scenarios where the number of triangles sharing an edge is bounded. An obvious example for such graphs are graphs with bounded degree. We explore some other scenarios.

- (i) Consider a graph $G(P, E)$ such that the vertex set P corresponds to a subset of \mathbb{R}^2 and $(u, v) \in E$ if and only if the distance between u and v is exactly 1. The objective is to compute the number of triples of points from P forming an equilateral triangle having side length 1, that is, the number of triangles in G . Observe that there can be at most two triangles sharing an edge in G , that is, $\Delta_E \leq 2$.
- (ii) Consider a graph $G(P, E)$ such that the vertex set P corresponds to a set of points inside an $N \times N$ square in \mathbb{R}^2 and $(u, v) \in E$ if and only if the distance between u and v is at most 1. The objective is to compute the number of triples of points from P forming a triangle having each side length at most 1, that is, the number of triangles in G . For *large* enough N there can be bounded number of triangles sharing an edge in G with high probability.
- (iii) Consider a graph $G(V, E)$ representing a community sharing information. Each node has some information and two nodes are connected if and only if there exists an edge between the nodes. Nodes increase their information by sharing information among their neighbors in G . Observe that the information of a node is derived by the set of neighbors. So, if two nodes have *large* number of common neighbors in G , then there is no need of an edge between the two nodes. So, the number of triangles on any edge in the graph is bounded. The objective is to compute the number of triangles in G , that is, the number of triples of nodes in G such that each pair of vertices are connected.

In (i) and (ii), TIS oracle can be implemented very efficiently. We can report a TIS query by just running a standard plane sweep algorithm in Computational Geometry that takes $\mathcal{O}(n \log n)$ running time.

B Some probability results

► **Proposition 14.** *Let X be a random variable. Then $\mathbb{E}[X] \leq \sqrt{\mathbb{E}[X^2]}$.*

► **Lemma 15** (Theorem 7.1 from [16]). *Let f be a function of n random variables X_1, \dots, X_n such that*

- (i) *Each X_i takes values from a set A_i ,*
- (ii) *$\mathbb{E}[f]$ is bounded, i.e., $0 \leq \mathbb{E}[f] \leq M$,*
- (iii) *\mathcal{B} be any event satisfying the following for each $i \in [n]$.*

$$|\mathbb{E}[f \mid X_1, \dots, X_{i-1}, X_i = a_i, \mathcal{B}^c] - \mathbb{E}[f \mid X_1, \dots, X_{i-1}, X_i = a'_i, \mathcal{B}^c]| \leq c_i.$$

Then for any $\delta \geq 0$,

$$\mathbb{P}(|f - \mathbb{E}[f]| > \delta + M \mathbb{P}(\mathcal{B})) \leq e^{-\delta^2 / \sum_{i=1}^n c_i^2} + \mathbb{P}(\mathcal{B}).$$

► **Lemma 16** (Hoeffding's inequality [16]). *Let X_1, \dots, X_n be n independent random variables such that $X_i \in [a_i, b_i]$. Then for $X = \sum_{i=1}^n X_i$, the following is true for any $\delta > 0$.*

$$\mathbb{P}(|X - \mathbb{E}[X]| \geq \delta) \leq 2 \cdot e^{-\frac{2\delta^2}{\sum_{i=1}^n (b_i - a_i)^2}}.$$

► **Lemma 17** (Chernoff-Hoeffding bound [16]). *Let X_1, \dots, X_n be independent random variables such that $X_i \in [0, 1]$. For $X = \sum_{i=1}^n X_i$ and $\mu_l \leq \mathbb{E}[X] \leq \mu_h$, the followings hold for any $\delta > 0$.*

- (i) $\mathbb{P}(X > \mu_h + \delta) \leq e^{-2\delta^2/n}$.
- (ii) $\mathbb{P}(X < \mu_l - \delta) \leq e^{-2\delta^2/n}$.

► **Lemma 18** (Theorem 3.2 from [16]). *Let X_1, \dots, X_n be random variables such that $a_i \leq X_i \leq b_i$ and $X = \sum_{i=1}^n X_i$. Let \mathcal{D} be the dependent graph, where $V(\mathcal{D}) = \{X_1, \dots, X_n\}$ and $E(\mathcal{D}) = \{(X_i, X_j) : X_i \text{ and } X_j \text{ are dependent}\}$. Then for any $\delta > 0$,*

$$\mathbb{P}(|X - \mathbb{E}[X]| \geq \delta) \leq 2e^{-2\delta^2/\chi^*(\mathcal{D}) \sum_{i=1}^n (b_i - a_i)^2},$$

where $\chi^*(\mathcal{D})$ denotes the fractional chromatic number of \mathcal{D} .

The following lemma directly follows from Lemma 18.

► **Lemma 19.** *Let X_1, \dots, X_n be indicator random variables such that there are at most d many X_j 's on which an X_i depends and $X = \sum_{i=1}^n X_i$. Then for any $\delta > 0$,*

$$\mathbb{P}(|X - \mathbb{E}[X]| \geq \delta) \leq 2e^{-2\delta^2/(d+1)n}.$$

► **Lemma 20** (Importance sampling [6]). *Let $(D_1, w_1, e_1), \dots, (D_r, w_r, e_r)$ are the given structures and each D_i has an associated weight $c(D_i)$ satisfying*

- (i) $w_i, e_i \geq 1, \forall i \in [r]$;
- (ii) $\frac{e_i}{\rho} \leq c(D_i) \leq e_i \rho$ for some $\rho > 0$ and all $i \in [r]$; and
- (iii) $\sum_{i=1}^r w_i \cdot c(D_i) \leq M$.

Note that the exact values $c(D_i)$'s are not known to us. Then there exists an algorithm that finds $(D'_1, w'_1, e'_1), \dots, (D'_s, w'_s, e'_s)$ such that all of the above three conditions hold and $\left| \sum_{i=1}^s w'_i \cdot c(D'_i) - \sum_{i=1}^r w_i \cdot c(D_i) \right| \leq \lambda S$ with probability $1 - \delta$; where $S = \sum_{i=1}^r w_i \cdot c(D_i)$ and $\lambda, \delta > 0$. The time complexity of the algorithm is $\mathcal{O}(r)$ and $s = \mathcal{O}\left(\frac{\rho^4 \log M (\log \log M + \log \frac{1}{\delta})}{\lambda^2}\right)$.

Step-By-Step Community Detection in Volume-Regular Graphs

Luca Becchetti 

Sapienza Università di Roma, Italy
<https://www.diag.uniroma1.it/~becchett>
becchetti@diag.uniroma1.it

Emilio Cruciani 

Gran Sasso Science Institute, L'Aquila, Italy
<https://sites.google.com/view/emiliocruciani>
emilio.cruciani@gssi.it

Francesco Pasquale 

Università di Roma "Tor Vergata", Italy
<https://www.mat.uniroma2.it/~pasquale>
pasquale@mat.uniroma2.it

Sara Rizzo 

Gran Sasso Science Institute, L'Aquila, Italy
<https://sites.google.com/view/sararizzo>
sara.rizzo@gssi.it

Abstract

Spectral techniques have proved amongst the most effective approaches to graph clustering. However, in general they require explicit computation of the main eigenvectors of a suitable matrix (usually the Laplacian matrix of the graph).

Recent work (e.g., Becchetti et al., SODA 2017) suggests that observing the temporal evolution of the power method applied to an initial random vector may, at least in some cases, provide enough information on the space spanned by the first two eigenvectors, so as to allow recovery of a hidden partition without explicit eigenvector computations. While the results of Becchetti et al. apply to perfectly balanced partitions and/or graphs that exhibit very strong forms of regularity, we extend their approach to graphs containing a hidden k partition and characterized by a milder form of volume-regularity. We show that the class of k -volume regular graphs is the largest class of undirected (possibly weighted) graphs whose transition matrix admits k "stepwise" eigenvectors (i.e., vectors that are constant over each set of the hidden partition). To obtain this result, we highlight a connection between volume regularity and lumpability of Markov chains. Moreover, we prove that if the stepwise eigenvectors are those associated to the first k eigenvalues and the gap between the k -th and the $(k+1)$ -th eigenvalues is sufficiently large, the AVERAGING dynamics of Becchetti et al. recovers the underlying community structure of the graph in logarithmic time, with high probability.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Community detection, Distributed algorithms, Dynamics, Markov chains, Spectral analysis

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.20

Related Version <https://arxiv.org/abs/1907.07149>

Funding *Luca Becchetti*: Partially supported by ERC Advanced Grant 788893 AMDROMA "Algorithmic and Mechanism Design Research in Online Markets" and MIUR PRIN project ALGADIMAR "Algorithms, Games, and Digital Markets".

Francesco Pasquale: Partially supported by the University of "Tor Vergata" under research program "Mission: Sustainability" project ISIDE (grant no. E81I18000110005).



© Luca Becchetti, Emilio Cruciani, Francesco Pasquale, and Sara Rizzo;
licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 20; pp. 20:1–20:23

Leibniz International Proceedings in Informatics



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

1 Introduction

Clustering a graph in a way that reflects underlying community structure is a very important mining task [18]. Informally speaking, in the classical setting, we are given a possibly weighted graph G and an integer k . Our goal is to partition the vertex set of $G = (V, E)$ into k disjoint subsets, so that the k induced subgraphs have high inner and low outer expansion. Spectral techniques have proved amongst the most effective approaches to graph clustering [37, 41, 45]. The general approach to spectral graph clustering [45] normally implies embedding the vertices of G into the k -dimensional subspace spanned by the main k eigenvectors of a matrix defined in terms of G 's adjacency matrix, typically its (normalized) Laplacian. Intuitively, one expects that, for a well-clustered graph with k communities, the profiles of the first k eigenvectors are correlated with the underlying community structure of G . Recent work has provided theoretical support to this approach. In particular, [27] showed that, given the first k orthonormal eigenvectors of the normalized Laplacian, it is possible to produce a k -partition of the vertex set, corresponding to k suitably-defined indicator vectors, such that the associated values of the Rayleigh quotient are relatively small. More recently, [38] proved that, under suitable hypotheses on the spectral gap between the k -th and $(k+1)$ -th eigenvalue of the normalized Laplacian of G , the span of the first k eigenvectors largely overlaps with the span of $\{D^{\frac{1}{2}}\mathbf{g}_1, \dots, D^{\frac{1}{2}}\mathbf{g}_k\}$, where D is the diagonal degree matrix of G , while the \mathbf{g}_i 's are indicator vectors describing a k -way partition $\{S_i\}_{i=1}^k$ of V such that, for every i , the conductance of S_i is at most the k -way expansion constant $\rho(k)$ [27]. Note that, if \mathbf{v} is an eigenvector associated to the i -th smallest eigenvalue of the normalized Laplacian, $D^{-\frac{1}{2}}\mathbf{v}$ is an eigenvector corresponding to the i -th largest eigenvalue of the random walk's transition matrix associated to G . Hence, when G is well-clustered, one might reasonably expect the first k eigenvectors of P to exhibit almost-“stepwise” profiles reflecting G 's underlying community structure. The aforementioned spectral approaches require explicit computation of the k main eigenvectors of a (generally symmetric) matrix.

In [6], the authors considered the case $k = 2$ for which they proposed the following distributed algorithm (AVERAGING dynamics, Algorithm 1): “At the outset, every node picks an initial value, independently and uniformly at random in $\{-1, 1\}$; then, in each synchronous round, every node updates its value to the average of those held by its neighbors. A node also tags itself **blue** if the last update increased its value, **red** otherwise” [6]. The authors showed that, under a variety of graph models exhibiting sparse balanced cuts, including the *stochastic block model* [20], the process resulting from the above simple local rule converges, in logarithmic time, to a coloring that, depending on the model, exactly or approximately reflects the underlying cut. They further elaborated on how to extend the proposed approach to the case of multiple communities, providing an analysis for a strongly regular version of the stochastic block model with multiple communities. While results like those presented in [27, 38] provide further theoretical justification for spectral clustering, the approach proposed in [6] suggests that observing the temporal evolution of the power method applied to an initial random vector may, at least in some cases, provide equivalent information, without requiring explicit eigenvector computations.

1.1 Our contributions

The goal of this work is to take a further step in this direction by considering a more general class of graphs, even if still relatively “regular”, than the one considered in [6]. The analysis of the AVERAGING dynamics on this class is considerably harder, but it is likely to provide insights into the challenges of analyzing the general case, without all the intricacies of the latter. Our contribution is as follows:

- We define the class of *k-volume-regular* graphs. This class of edge-weighted graphs includes those considered in [6] and it is the largest class of undirected, possibly weighted graphs that admit *k* “stepwise” eigenvectors (i.e., having constant values over the *k* steps that identify the hidden partition). This result uses a connection between volume regularity and lumpability of Markov chains [22, 44].
- If the stepwise eigenvectors are those associated to the first *k* eigenvalues and the gap between the *k*-th and the (*k*+1)-th eigenvalues is sufficiently large, we show that running the AVERAGING dynamics for a suitable number of steps allows recovery of the underlying community structure of the graph, with high probability.¹ To prove this, we provide a family of mutually orthonormal vectors which, when the graph is volume-regular, span the eigenspace of the main *k* eigenvectors of the normalized adjacency matrix of the graph. It should be noted that the first and second of these vectors are respectively the main eigenvector and the Fiedler vector [17] associated to the normalized adjacency matrix.
- While the results of [6] apply when the underlying communities are of the same size, our results do not require this assumption and they apply to weighted graphs. It should also be noted that volume regularity is a weaker notion than regularity of the graph.
- We further show that variants of the AVERAGING dynamics (and/or its labeling rule) can address different problems (e.g., identifying bipartiteness) and/or other graph classes.

We finally note that the overall algorithm we consider can be viewed as a fully decentralized, synchronous algorithm that works in *anonymous* networks,² with a completely local clustering criterion, though it cannot be considered a *dynamics* in the sense of [6] since it requires a bound on the number of nodes in the underlying network.

1.2 Further related work

We briefly discuss further work that bears some relationship to this paper, either because it adopts simple and/or decentralized heuristics to uncover community structure, or because it relies on the use of spectral techniques.

Decentralized heuristics for block reconstruction. *Label propagation algorithms* [39] are dynamics based on majority updating rules [3] and have been applied for detecting communities in complex networks. Several papers present experimental results for such protocols on specific classes of clustered graphs [4, 29, 39]. The only available rigorous analysis of a label propagation algorithm on planted partition graphs is the one presented in [24], where the authors analyze a label propagation algorithm on $\mathcal{G}_{2n,p,q}$ graphs in the case of dense topologies. In particular, their analysis considers the case where $p = \Omega(1/n^{\frac{1}{4}-\epsilon})$ and $q = \mathcal{O}(p^2)$, a parameter range in which very dense clusters of constant diameter separated by a sparse cut occur w.h.p. In this setting, characterized by a polynomial gap between *p* and *q*, simple combinatorial and concentration arguments show that the protocol converges in constant expected time. A logarithmic bound for sparser topologies is conjectured in [24].

Following [6], a number of recent papers analyze simple distributed algorithms for community detection that rely on elementary dynamics. In the AVERAGING dynamics considered in this paper, every node communicates in parallel with all its neighbors in each round. While this might be too expensive in scenarios characterized by dense topologies, it is

¹ An event \mathcal{E}_n holds *with high probability (w.h.p.)* if $\mathbf{P}(\mathcal{E}_n) = 1 - \mathcal{O}(n^{-\gamma})$, for some constant $\gamma > 0$.

² Nodes do not possess distinguished identities.

simply infeasible in other settings (for instance, when links represent opportunistic meetings that occur asynchronously). Motivated by similar considerations, a first line of follow-up work considered “sparsified”, asynchronous variants of the AVERAGING dynamics [5, 31, 43].

Another interesting direction is the rigorous analysis of well-known (non-linear) dynamics based on *majority rules* on graphs that exhibit community structure. In [12], Cruciani et al. consider the *2-Choices* dynamics where, in each round, every node picks two random neighbors and updates its value to the most frequent among its value and those held by its sampled neighbors. They show that if the underlying graph has a suitable core-periphery structure and the process starts in a configuration where nodes in core and periphery have different states, the system either rapidly converges to the core’s state or reaches a metastable regime that reflects the underlying graph structure. Similar results have been also obtained for clustered regular graphs with dense communities in [13], where the *2-Choices* dynamics is proposed as a distributed algorithm for community detection.

Although based on the AVERAGING dynamics and thus extremely simple and fully decentralized, the algorithm we consider in this paper is not itself a dynamics in the sense proposed in [6], since its clustering criterion is applied within a time window, which in turn requires (at least approximate) knowledge of the network size.

Because of their relevance for the reconstruction problem, we also briefly discuss the class of *belief propagation algorithms*, best known as message-passing algorithms for performing inference in graphical models [30]. Though not a dynamics, belief propagation is still a simple approach. Moreover, there is non-rigorous, strong supporting evidence that some *belief propagation algorithms* might be optimal for the reconstruction problem [14]. A rigorous analysis is a major challenge; in particular, convergence to the correct value of belief propagation is far from being fully-understood on graphs which are not trees [34, 46]. As we discuss in the next subsection, more complex algorithms inspired by belief propagation have been rigorously shown to perform reconstruction optimally.

General algorithms for block reconstruction. Several algorithms for community detection are *spectral*: They typically consider the eigenvector associated to the second largest eigenvalue of the adjacency matrix A of G , or the eigenvector corresponding to the largest eigenvalue of the matrix $A - \frac{d}{n}J$ [7, 10, 11, 32],³ since these are correlated with the hidden partition. More recently spectral algorithms have been proposed [2, 8, 11, 25, 36, 38] that find a weak reconstruction even in the sparse, tight regime.

Interestingly, spectral algorithms turn out to be a feasible approach also in distributed settings. In particular, Kempe and McSherry [23] show that eigenvalue computations can be performed in a distributed fashion, yielding distributed algorithms for community detection under various models, including the stochastic block model. However, their algorithm does not match any simple decentralized computing model. In particular, the algorithm of Kempe and McSherry as well as any distributed version of the above mentioned centralized algorithms are neither dynamics, nor do they correspond to the notion of *light-weight* algorithm of Hassin and Peleg [19]. Moreover, the mixing time of the simple random walk on the graph is a bottleneck for the distributed algorithm of Kempe and McSherry and for any algorithm that performs community detection in a graph G by employing the power method or the Lanczos method [26] as a subroutine. This is not the case for the AVERAGING dynamics, since it removes the component of the state in the span of the main eigenvector.

³ A is the adjacency matrix of G , J is the matrix having all entries equal to 1, d is the average degree, and n is the number of vertices.

In general, the reconstruction problem has been studied extensively using a multiplicity of techniques, which include combinatorial algorithms [15], belief propagation [14] and variants of it [35], spectral-based techniques [11, 32], Metropolis approaches [21], and semidefinite programming [1], among others.

1.3 Roadmap

The rest of this paper is organized as follows. In Section 2, we formally define the AVERAGING dynamics and briefly recall how it is connected with the transition matrix of a random walk on the underlying graph. We also define the notion of *community-sensitive algorithm* and the class of *clustered volume-regular graphs*. In Section 3 we show the relation between lumpability of Markov chains and volume-regular graphs. In Section 4 we state the main result of the paper (see Theorem 9) on the analysis of the AVERAGING for clustered volume-regular graphs: We give the two main technical lemmas and show how the main theorem derives from them. In Section 5 we show how slightly modified versions of the AVERAGING dynamics can be used to identify the hidden partition of other non-clustered volume-regular graphs, e.g., bipartite graphs. In Section 6 we draw some conclusions and point to some open problems. Full proofs of technical lemmas are given in the Appendix.

2 Preliminaries

Notation. Consider an undirected edge-weighted graph $G = (V, E, w)$ with nonnegative weights. For each node $u \in V$, we denote by $\delta(u)$ the *volume*, or *weighted degree*, of node u , namely $\delta(u) = \sum_{v:(u,v) \in E} w(u, v)$. D denotes the diagonal matrix, such that $D_{uu} = \delta(u)$ for each $u \in V$. Without loss of generality we assume $\min_u \delta(u) = 1$, since the behavior of the AVERAGING dynamics (and the corresponding analysis) is not affected by a normalization of the weights. We refer to the maximum volume of a node as $\Delta := \max_u \delta(u)$.

In the remainder, W denotes the *weighted adjacency matrix* of G , while $P = D^{-1}W$ is the *transition matrix* of a random walk on G , in which a transition from node u to node v occurs with probability proportional to $w(u, v)$. We call $\lambda_1, \dots, \lambda_n$ the eigenvalues of P , in non-increasing order, and $\mathbf{v}_1, \dots, \mathbf{v}_n$ a family of eigenvectors of P , such that $P\mathbf{v}_i = \lambda_i\mathbf{v}_i$. We let $N = D^{-\frac{1}{2}}WD^{-\frac{1}{2}} = D^{\frac{1}{2}}PD^{-\frac{1}{2}}$ denote the *normalized weighted adjacency matrix* of G . Note that N is symmetric and that its spectrum is the same as that of P . We denote by $\mathbf{w}_1, \dots, \mathbf{w}_n$ a family of eigenvectors of N , such that $N\mathbf{w}_i = \lambda_i\mathbf{w}_i$. It is important to note that \mathbf{w}_i is an eigenvector of N if and only if $D^{-\frac{1}{2}}\mathbf{w}_i$ is an eigenvector of P .

2.1 Averaging dynamics

The simple algorithm we consider in this paper, named AVERAGING dynamics (Algorithm 1) after [6] in which the algorithm was first proposed, can be seen as an application of the power method, augmented with a Rademacher initialization and a suitable labeling scheme. In this form, it is best described as a distributed process, executed by the nodes of an underlying edge-weighted graph. The AVERAGING dynamics can be used as a building-block to achieve “community detection” in some classes of “regular” and “almost regular” graphs. Herein, we extend its use and analysis to broader graph classes and, in one case, to a different problem.

■ **Algorithm 1** AVERAGING dynamics.

Rademacher initialization: At round $t = 0$, every node $v \in V$ independently samples its value $\mathbf{x}^{(0)}(v)$ from $\{-1, +1\}$ uniformly at random.

Update rule: At each subsequent round $t \geq 1$, every node $v \in V$:

1. *Averaging:* updates its value $\mathbf{x}^{(t)}(v)$ to the weighted average of the values of its neighbors at the end of the previous round.
2. *Labeling:* if $\mathbf{x}^{(t)}(v) \geq \mathbf{x}^{(t-1)}(v)$ then v sets $\text{label}^{(t)}(v) = 1$; otherwise v sets $\text{label}^{(t)}(v) = 0$.

Spectral decomposition of the transition matrix. Let $\mathbf{x}^{(t)}$ denote the *state vector* at time t , i.e., the vector whose u -th entry is the value held by node u at time t . We let $\mathbf{x}^{(0)} = \mathbf{x}$ denote the initial state vector. Globally, the *averaging* update rule of Algorithm 1 corresponds to one iteration of the power method, in this case an application of the transition matrix P to the current state vector, i.e., $\mathbf{x}^{(t)} = P\mathbf{x}^{(t-1)}$. We can write

$$\mathbf{x}^{(t)} = P^t \mathbf{x} = D^{-\frac{1}{2}} N^t D^{\frac{1}{2}} \mathbf{x} \stackrel{(a)}{=} D^{-\frac{1}{2}} \sum_{i=1}^n \lambda_i^t \mathbf{w}_i \mathbf{w}_i^\top \sum_{i=1}^n \beta_i \mathbf{w}_i = \sum_{i=1}^n \lambda_i^t \beta_i D^{-\frac{1}{2}} \mathbf{w}_i,$$

where in (a) we spectrally decomposed the matrix N^t and expressed the vector $D^{\frac{1}{2}} \mathbf{x}$ as a linear combination of the eigenvectors of N , i.e., $D^{\frac{1}{2}} \mathbf{x} = \sum_{i=1}^n \beta_i \mathbf{w}_i$, with $\beta_i = \langle D^{\frac{1}{2}} \mathbf{x}, \mathbf{w}_i \rangle$. By explicitly writing the β_i s and by noting that $\mathbf{w}_i = \frac{D^{\frac{1}{2}} \mathbf{v}_i}{\|D^{\frac{1}{2}} \mathbf{v}_i\|}$ we conclude that

$$\mathbf{x}^{(t)} = \sum_{i=1}^n \lambda_i^t \frac{\langle D^{\frac{1}{2}} \mathbf{x}, D^{\frac{1}{2}} \mathbf{v}_i \rangle}{\|D^{\frac{1}{2}} \mathbf{v}_i\|} D^{-\frac{1}{2}} \frac{D^{\frac{1}{2}} \mathbf{v}_i}{\|D^{\frac{1}{2}} \mathbf{v}_i\|} = \sum_{i=1}^n \lambda_i^t \alpha_i \mathbf{v}_i, \quad (1)$$

where $\alpha_i := \frac{\langle D^{\frac{1}{2}} \mathbf{x}, D^{\frac{1}{2}} \mathbf{v}_i \rangle}{\|D^{\frac{1}{2}} \mathbf{v}_i\|^2} = \frac{\mathbf{x}^\top D \mathbf{v}_i}{\|D^{\frac{1}{2}} \mathbf{v}_i\|^2}$.

Note that $\lambda_1 = 1$ and $\mathbf{v}_1 = \mathbf{1}$ (where $\mathbf{1}$ denotes the vector whose entries are 1), since P is stochastic and, if G is connected and non bipartite, $\lambda_i \in (-1, 1)$ for every $i > 1$. The long term behavior of the dynamics can be written as

$$\lim_{t \rightarrow \infty} \mathbf{x}^{(t)} = \lim_{t \rightarrow \infty} \sum_{i=1}^n \lambda_i^t \alpha_i \mathbf{v}_i = \alpha_1 \mathbf{1}, \quad \text{with } \alpha_1 = \frac{\sum_{u \in V} \delta(u) \mathbf{x}(u)}{\sum_{u \in V} \delta(u)} = \sum_{u \in V} \frac{\delta(u)}{\text{vol}(V)} \mathbf{x}(u),$$

i.e., each node converges to the initial global weighted average of the network.

2.2 Community-sensitive algorithms

We give the following definition of *community sensitive algorithm*, that closely resembles that of locality-sensitive hashing (see, e.g., [28]).

► **Definition 1** (Community-sensitive algorithm). *Let \mathcal{A} be a randomized algorithm that takes in input a (possibly weighted) graph $G = (V, E)$ with a hidden partition $\mathcal{V} = \{V_1, \dots, V_k\}$ and assigns a Boolean value $\mathcal{A}(G)[v] \in \{0, 1\}$ to each node $v \in V$. We say \mathcal{A} is an (ε, δ) -Community-sensitive algorithm, for some $\varepsilon, \delta > 0$, if the following two conditions hold:*

1. *For each set V_i of the partition and for each pair of nodes $u, v \in V_i$ in that set, the probability that the algorithm assigns the same Boolean value to u and v is at least $1 - \varepsilon$,*

$$\forall i \in [k], \forall u, v \in V_i, \mathbf{P}(\mathcal{A}(G)[u] = \mathcal{A}(G)[v]) \geq 1 - \varepsilon.$$

2. For each pair V_i, V_j of distinct sets of the partition and for each pair of nodes $u \in V_i$ and $v \in V_j$, the probability that the algorithm assigns the same value to u and v is at most δ ,

$$\forall i, j \in [k] \text{ with } i \neq j, \forall u \in V_i, \forall v \in V_j, \mathbf{P}(\mathcal{A}(G)[u] = \mathcal{A}(G)[v]) \leq \delta.$$

For example, for $(\varepsilon, \delta) = (1/n, 1/2)$, an algorithm that simply assigns the same value to all nodes would satisfy the first condition but not the second one, while an algorithm assigning 0 or 1 to each node with probability 1/2, independently of the other nodes, would satisfy the second condition but not the first one.

Note that Algorithm 1 is a distributed algorithm that, at each round t , assigns one out of two labels to each node of a graph. In the next section (see Theorem 9) we prove that a time window $[T_1, T_2]$ exists, such that for all rounds $t \in [T_1, T_2]$, the assignment of the AVERAGING dynamics satisfies both conditions in Definition 1: The first condition with $\varepsilon = \varepsilon(n) = O(n^{-\frac{1}{2}})$, the second with $\delta = \delta(n) = 1 - \Omega(1)$.

Community-sensitive labeling. If we execute $\ell = \Theta(\log n)$ independent runs of an (ε, δ) -Community-sensitive algorithm \mathcal{A} , each node is assigned a *signature* of ℓ binary values, with pairwise Hamming distances probabilistically reflecting community membership of the nodes. More precisely, let \mathcal{A} be an (ε, δ) -Community-sensitive algorithm and let $\mathcal{A}_1, \dots, \mathcal{A}_\ell$ be $\ell = \Theta(\log n)$ independent runs of \mathcal{A} . For each node $u \in V$, let $\mathbf{s}(u) = (s_1(u), \dots, s_\ell(u))$ denote the *signature* of node u , where $s_i(u) = \mathcal{A}_i(G)[u]$. For each pair nodes u, v , let $h(u, v) = |\{i \in [\ell] : s_i(u) \neq s_i(v)\}|$ be the Hamming distance between $\mathbf{s}(u)$ and $\mathbf{s}(v)$. The following lemma follows from a straightforward application of Chernoff bounds.

► **Lemma 2** (From Community-sensitive algorithm to Community-sensitive labeling). *Let \mathcal{A} be an (ε, δ) -Community-sensitive algorithm with $\varepsilon = o(1)$ and $\delta = 1 - \Omega(1)$. For large enough $\ell = \Theta(\log n)$, two positive constants α, β exist, with $0 \leq \alpha < \beta \leq 1$, such that for each pair of nodes $u, v \in V$ it holds that:*

- *If u and v belong to the same community then $h(u, v) \leq \alpha\ell$, w.h.p.*
- *If u and v belong to different communities then $h(u, v) \geq \beta\ell$, w.h.p.*

Proof. If u and v belong to the same community, then $\mathbf{E}[h(u, v)] \leq \varepsilon\ell$. If they belong to different communities, then $\mathbf{E}[h(u, v)] \geq (1 - \delta)\ell$. The thesis follows by a standard application of Chernoff bounds, e.g., by choosing $\alpha = (1 - \delta)/4$ and $\beta = (1 - \delta)/2$. ◀

2.3 Volume-regular graphs

Recall that, for an undirected edge-weighted graph $G = (V, E, w)$, we denote by $\delta(u)$ the volume a node $u \in V$, i.e., $\delta(u) = \sum_{v:(u,v) \in E} w(u, v)$. Note that the transition matrix P of a random walk on G is such that $P_{uv} = w(u, v) / \delta(u)$. Given a partition $\mathcal{V} = \{V_1, \dots, V_k\}$ of the set of nodes V , for a node $u \in V$ and a partition index $i \in [k]$, $\delta_i(u)$ denotes the overall weight of edges connecting u to nodes in V_i , $\delta_i(u) = \sum_{v \in V_i : (u,v) \in E} w(u, v)$. Hence, $\delta(u) = \sum_{i=1}^k \delta_i(u)$.

► **Definition 3** (Volume-regular graph). *Let $G = (V, E, w)$ be an undirected edge-weighted graph with $|V| = n$ nodes and let $\mathcal{V} = \{V_1, \dots, V_k\}$ be a k -partition of the nodes, for some $k \in [n]$. We say that G is volume-regular with respect to \mathcal{V} if, for every pair of partition indexes $i, j \in [k]$ and for every pair of nodes $u, v \in V_i$, $\frac{\delta_j(u)}{\delta(u)} = \frac{\delta_j(v)}{\delta(v)}$. We say that G is k -volume-regular if there exists a k -partition \mathcal{V} of the nodes such that G is volume-regular with respect to \mathcal{V} .*

In other words, G is volume-regular if there exists a partition of the nodes such that the fraction of a node's volume toward a set of the partition is constant across nodes of the same set. Note that all graphs with n nodes are trivially 1- and n -volume-regular.

Let $G = (V, E, w)$ be a k -volume-regular graph and let P be the transition matrix of a random walk on G . In the next lemma we prove that the span of k linearly independent eigenvectors of P equals the span of the indicator vectors of the k communities of G . The proof makes use of the correspondence between random walks on volume-regular graphs and *ordinary lumpable* Markov chains [22]; in particular the result follows from Lemma 7 and Lemma 8 that can be found in Section 3.

► **Lemma 4.** *Let P be the transition matrix of a random walk on a k -volume-regular graph $G = (V, E, w)$ with k -partition $\mathcal{V} = \{V_1, \dots, V_k\}$. There exists a family $\{\mathbf{v}_1, \dots, \mathbf{v}_k\}$ of linearly independent eigenvectors of P such that $\text{Span}(\{\mathbf{v}_1, \dots, \mathbf{v}_k\}) = \text{Span}(\{\mathbf{1}_{V_1}, \dots, \mathbf{1}_{V_k}\})$, with $\mathbf{1}_{V_i}$ the indicator vector of the i -th set of the partition, for $i \in [k]$.*

In the rest of the paper we call “stepwise” the eigenvectors of P that can be written as linear combinations of the indicator vectors of the communities. In the next definition, we formalize the fact that a k -volume-regular graph is *clustered* if the k linearly independent stepwise eigenvectors of P , whose existence is guaranteed by the above lemma, are associated to the k largest eigenvalues of P .

► **Definition 5** (Clustered volume-regular graph). *Let $G = (V, E, w)$ be a k -volume-regular graph and let P be the transition matrix of a random walk on G . We say that G is a clustered k -volume-regular graph if the k stepwise eigenvectors of P are associated to the first k largest eigenvalues of P .*

3 Volume-regular graphs and lumpable Markov chains

The class of volume-regular graphs is deeply connected with the definition of *lumpability* [22] of Markov chains. We here first recall the definition of lumpable Markov chain and then show that a graph G is volume-regular if and only if the associated weighted random walk is a lumpable Markov chain.

► **Definition 6** (Ordinary lumpability of Markov Chains). *Let $\{X_t\}_t$ be a finite Markov chain with state space V and transition matrix $P = (P_{uv})_{u,v \in V}$ and let $\mathcal{V} = \{V_1, \dots, V_k\}$ be a partition of the state space. Markov chain $\{X_t\}_t$ is ordinary lumpable with respect to \mathcal{V} if, for every pair of partition indexes $i, j \in [k]$ and for every pair of nodes in the same set of the partition $u, v \in V_i$, it holds that*

$$\sum_{w \in V_i} P_{uw} = \sum_{w \in V_j} P_{vw}, \quad \forall u, v \in V_j. \quad (2)$$

We define the lumped matrix \hat{P} of the Markov Chain as the matrix such that $\hat{P}_{ij} = \sum_{w \in V_j} P_{uw}$, for any $u \in V_i$.

We first prove that random walks on volume-regular graphs define exactly the subset of reversible and ordinary lumpable Markov chains.

► **Lemma 7.** *A reversible Markov chain $\{X_t\}_t$ is ordinary lumpable if and only if it is a random walk on a volume-regular graph.*

Proof. Assume first that $\{X_t\}_t$ is ordinary lumpable and let P be the corresponding transition matrix. Consider the weighted graph $G = (V, E, w)$ obtained from P as follows: V corresponds to the set of states in P , while $w(u, v) = \pi(u)P_{uv}$, for every $u, v \in V$, with π the stationary distribution of P . Note that G is an undirected graph, i.e., $w(u, v) = \pi(u)P_{uv} \stackrel{(a)}{=} \pi(v)P_{vu} = w(v, u)$, where (a) holds because P is reversible. Moreover

$$\delta(u) = \sum_{z \in V} w(u, z) = \sum_{z \in V} \pi(u)P_{uz} = \pi(u) \sum_{z \in V} P_{uz} \stackrel{(b)}{=} \pi(u),$$

where (b) holds because P is stochastic. Thus G meets Definition 3 because, for any $u, v \in V_i$,

$$\frac{\delta_j(u)}{\delta(u)} = \frac{1}{\pi(u)} \sum_{z \in V_j} w(u, z) = \sum_{z \in V_j} P_{uz} = \sum_{z \in V_j} P_{vz} = \frac{1}{\pi(v)} \sum_{z \in V_j} w(v, z) = \frac{\delta_j(v)}{\delta(v)}.$$

Next, assume G is k -volume-regular with respect to the partition $\mathcal{V} = \{V_1, \dots, V_k\}$. Let P be the transition matrix of the corresponding random walk. For every $i, j \in [k]$ and for every $u, v \in V_i$ we have:

$$\sum_{z \in V_j} P_{uz} = \sum_{z \in V_j} \frac{w(u, z)}{\delta(u)} = \frac{\delta_j(u)}{\delta(u)} \stackrel{(a)}{=} \frac{\delta_j(v)}{\delta(v)} = \sum_{z \in V_j} \frac{w(v, z)}{\delta(v)} = \sum_{z \in V_j} P_{vz},$$

where (a) follows from Definition 3. Moreover note that P is reversible with respect to distribution π , where $\pi(u) = \frac{\delta(u)}{\text{vol}(G)}$. \blacktriangleleft

Note that infinitely many k -volume-regular graphs have the same k -ordinary lumpable random walk chain.

We next show that a Markov chain is k -ordinary lumpable if and only if the corresponding transition matrix P has k stepwise, linearly independent eigenvectors.

► **Lemma 8.** *Let P be the transition matrix of a Markov chain. Then P has k stepwise linearly independent eigenvectors if and only if P is ordinary lumpable.*

Proof. We divide the proof in two parts. First, we assume that P is ordinary lumpable and show that P has k stepwise linearly independent eigenvectors. Second, we assume that P has k stepwise linearly independent eigenvectors and show that P is ordinary lumpable.

1. Let P be ordinary lumpable and \hat{P} its lumped matrix. Let $\lambda_i, \hat{\mathbf{v}}_i$ be the eigenvalues and eigenvectors of \hat{P} , for each $i \in [k]$. Let $\mathbf{v}_i \in \mathbb{R}^n$ be a stepwise vector defined as

$$\mathbf{v}_i = (\hat{\mathbf{v}}_i(1), \dots, \hat{\mathbf{v}}_i(1), \hat{\mathbf{v}}_i(2), \dots, \hat{\mathbf{v}}_i(2), \dots, \hat{\mathbf{v}}_i(k), \dots, \hat{\mathbf{v}}_i(k))^\top,$$

where $\hat{\mathbf{v}}_i(j)$ indicates the j -th component of $\hat{\mathbf{v}}_i$, and then the n_j components relative to V_j are all equal to $\hat{\mathbf{v}}_i(j)$.

Since the eigenvectors $\hat{\mathbf{v}}_i$ of \hat{P} are linearly independent, the vectors \mathbf{v}_i are also linearly independent. Moreover, it is easy to see that $P\mathbf{v}_i = \lambda_i\mathbf{v}_i$ by just verifying the equation for every $i \in [k]$.

2. Assume P has k stepwise linearly independent eigenvectors \mathbf{v}_i , associated to k eigenvalues λ_i , for each $i \in [k]$. Let $\hat{\mathbf{v}}_i \in \mathbb{R}^k$ the vector that has as components the k constant values in the steps of \mathbf{v}_i . Since the \mathbf{v}_i are linearly independent, the $\hat{\mathbf{v}}_i$ also are.

For every eigenvector \mathbf{v}_i and for every two states $x, y \in V_l$, for every $l \in [k]$, we have that $\lambda_i\mathbf{v}_i(x) = \lambda_i\mathbf{v}_i(y)$ since \mathbf{v}_i is stepwise. Then, since $P\mathbf{v}_i = \lambda_i\mathbf{v}_i$, we have that

$$\sum_{j=1}^k \sum_{z \in V_j} P_{xz} \hat{\mathbf{v}}_i(j) = (P\mathbf{v}_i)(x) = (P\mathbf{v}_i)(y) = \sum_{j=1}^k \sum_{z \in V_j} P_{yz} \hat{\mathbf{v}}_i(j).$$

20:10 Step-By-Step Community Detection in Volume-Regular Graphs

Thus $\sum_{j=1}^k \hat{\mathbf{v}}_i(j) \sum_{z \in V_j} (P_{xz} - P_{yz}) = 0$ and then it follows that

$$\sum_{j=1}^k \hat{\mathbf{v}}_i(j) \mathbf{u}_{xy}(j) = \langle \mathbf{u}_{xy}, \hat{\mathbf{v}}_i \rangle = 0,$$

where $\mathbf{u}_{xy}(j) = \sum_{z \in V_j} (P_{xz} - P_{yz})$. Since the $\hat{\mathbf{v}}_i$ are k linearly independent vectors in a k -dimensional space, \mathbf{u}_{xy} cannot be orthogonal to all of them and then it has to be the null vector, i.e., $\mathbf{u}_{xy}(j) = 0$ for all $j \in [k]$. This implies that P is ordinary lumpable, i.e., $\sum_{z \in V_j} P_{xz} = \sum_{z \in V_j} P_{yz}$. It is easy to verify that the eigenvalues and eigenvectors of \hat{P} are exactly $\lambda_i, \hat{\mathbf{v}}_i$, with $i \in [k]$. ◀

4 Averaging dynamics on clustered volume-regular graphs

For a volume-regular graph $G = (V, E, w)$ with n nodes and k -partition $\mathcal{V} = \{V_1, \dots, V_k\}$ we name $N = \frac{\max_i |V_i|}{\min_i |V_i|}$ the ratio between the maximum and minimum sizes of the communities. In this section we prove the following result for volume-regular graphs.

► **Theorem 9.** *Let $G = (V, E, w)$ be a connected clustered k -volume-regular graph with n nodes and k -partition $\mathcal{V} = \{V_1, \dots, V_k\}$, with $k \leq \sqrt{n}$, maximum weighted degree $\Delta \leq \text{poly}(n)$, and $N = \mathcal{O}(\sqrt{k}/\Delta)$. If $\lambda_k > \frac{1}{2}$ and $(1 - \lambda_2) \geq (\lambda_2 - \lambda_k) \Delta^{\frac{3}{2}} n^{1+c}$, for an arbitrarily-small positive constant c , then a time interval $[T_1, T_2]$ exists, with $T_1 = \mathcal{O}(\log n / \log(\lambda_k/\lambda_{k+1}))$ and $T_2 = \Omega(n^{c/3})$, such that for each time $t \in [T_1, T_2]$ the AVERAGING dynamics truncated at round t is a $(\mathcal{O}(n^{-\frac{1}{2}}), 1 - \Omega(1))$ -community sensitive algorithm, w.h.p.*

In the remainder of this section, we first introduce further notation and then state the two main technical lemmas (Lemma 10 and Lemma 11), that will be used in the proof of Theorem 9, which concludes this section.

Let $G = (V, E, w)$ be a clustered k -volume-regular graph and, without loss of generality, let V_1, \dots, V_k be an arbitrary ordering of its communities. We introduce a family of stepwise vectors that generalize Fiedler vector [17], namely

$$\left\{ \boldsymbol{\chi}_i = \sqrt{\frac{\hat{m}_i}{m_i}} \mathbf{1}_{V_i} - \sqrt{\frac{m_i}{\hat{m}_i}} \mathbf{1}_{\hat{V}_i} : i \in [k-1] \right\},$$

where $\mathbf{1}_{V_i}$ is the indicator vector of the set V_i and, for convenience sake, we denoted by m_i the volume of the i -th community, \hat{V}_i the set of all nodes in communities $i+1, \dots, k$, and \hat{m}_i the volume of \hat{V}_i , i.e., $m_i = \sum_{u \in V_i} \delta(u)$, $\hat{V}_i = \bigcup_{h=i+1}^k V_h$, and $\hat{m}_i = \sum_{h=i+1}^k m_h$. Note that vectors $\boldsymbol{\chi}_i$ s are “stepwise” with respect to the communities of G (i.e., for every $i \in [k-1]$, $\boldsymbol{\chi}_i(u) = \boldsymbol{\chi}_i(v)$ whenever u and v belong to the same community).

Recall from Equation (1) that the initial state vector can be written as $\mathbf{x} = \sum_{i=1}^n \alpha_i \mathbf{v}_i$. Let $\mathbf{z} = \sum_{i=1}^k \alpha_i \mathbf{v}_i$ and note that $\mathbf{z} = \alpha_1 \mathbf{1} + \sum_{i=1}^{k-1} \gamma_i \boldsymbol{\chi}_i$ by applying Lemma 4 and because $\text{Span}(\{\mathbf{1}, \boldsymbol{\chi}_1, \dots, \boldsymbol{\chi}_{k-1}\}) = \text{Span}(\{\mathbf{1}_{V_1}, \dots, \mathbf{1}_{V_k}\})$. Let us now define the vector $\mathbf{y} = \mathbf{z} - \alpha_1 \mathbf{1}$ or, equivalently,

$$\mathbf{y} = \sum_{i=1}^{k-1} \gamma_i \boldsymbol{\chi}_i, \text{ where } \gamma_i = \frac{\mathbf{x}^\top D \boldsymbol{\chi}_i}{\|D^{1/2} \boldsymbol{\chi}_i\|^2}.$$

Note that the coefficients γ_i s are proportional to the length of the projection of the (inhomogeneously) contracted state vector on the (inhomogeneously) contracted, not anymore stepwise, $D^{\frac{1}{2}} \boldsymbol{\chi}_i$ s and can be computed since the vectors in the family $\{D^{\frac{1}{2}} \mathbf{1}\} \cup \{D^{\frac{1}{2}} \boldsymbol{\chi}_i : i \in [k-1]\}$ are mutually orthogonal.⁴

⁴ The mutual orthogonality of the vectors, including $D^{\frac{1}{2}} \mathbf{1}$, is also one of the reasons why other “simpler” families of stepwise vectors, e.g., the indicator vectors of the communities, are not used instead.

The binary coloring of each node only depends on the difference of its state in two consecutive rounds (see Algorithm 1). Essentially in Lemma 10 we show that, under suitable assumptions on the transition matrix of a random walk on G , there exists a time window where the the difference of the state vector in two consecutive rounds, i.e., $\mathbf{x}^{(t)} - \mathbf{x}^{(t+1)}$ can be approximated by the previously defined vector \mathbf{y} in a way that the sign of the two vectors is equal in any component, with high probability. Instead, in Lemma 11 we prove that with some constant probability (i.e., independent from the number of nodes n) the first two “steps” of the vector \mathbf{y} have different signs, i.e., the sign can be considered as a criterion to distinguish the first two communities.

► **Lemma 10 (Sign of the difference).** *Let $G = (V, E, w)$ be a clustered k -volume-regular graph. If $\lambda_k > \frac{1}{2}$ and $(1 - \lambda_2) \geq (\lambda_2 - \lambda_k) \Delta^{\frac{3}{2}} n^{1+c}$, for an arbitrarily-small positive constant c , then a time interval $[T_1, T_2]$ exists, with $T_1 = \mathcal{O}(\log n / \log(\lambda_k / \lambda_{k+1}))$ and $T_2 = \Omega(n^{c/3})$, such that for each node $u \in V$ it holds that $\text{sgn}(\mathbf{x}^{(t)}(u) - \mathbf{x}^{(t+1)}(u)) = \text{sgn}(\mathbf{y}(u))$ for every round $t \in [T_1, T_2]$ of the execution of the AVERAGING dynamics, w.h.p.*

Proof. Recall from Equation (1) that the state vector at time t , i.e., $\mathbf{x}^{(t)}$, can be written as the sum of the first k stepwise vectors of P and of the remaining ones, namely

$$\mathbf{x}^{(t)} = \alpha_1 \mathbf{1} + \sum_{i=2}^k \lambda_i^t \alpha_i \mathbf{v}_i + \sum_{i=k+1}^n \lambda_i^t \alpha_i \mathbf{v}_i = \alpha_1 \mathbf{1} + \mathbf{c}^{(t)} + \mathbf{e}^{(t)},$$

where we call $\mathbf{c}^{(t)} := \sum_{i=2}^k \lambda_i^t \alpha_i \mathbf{v}_i$ the *core contribution* and $\mathbf{e}^{(t)} := \sum_{i=k+1}^n \lambda_i^t \alpha_i \mathbf{v}_i$ the *error contribution*. If we look at the difference of the state vector between two consecutive rounds, for each node $u \in V$, the first term cancels out being constant over time and we get $\mathbf{x}^{(t)}(u) - \mathbf{x}^{(t+1)}(u) = \mathbf{c}^{(t)}(u) - \mathbf{c}^{(t+1)}(u) + \mathbf{e}^{(t)}(u) - \mathbf{e}^{(t+1)}(u)$. Note that the sign of the difference between two consecutive states of each node $u \in V$ is determined by the difference of the core contributions during the two consecutive rounds, i.e., $\mathbf{c}^{(t)}(u) - \mathbf{c}^{(t+1)}(u)$, whenever

$$\left| \mathbf{c}^{(t)}(u) - \mathbf{c}^{(t+1)}(u) \right| > \left| \mathbf{e}^{(t)}(u) - \mathbf{e}^{(t+1)}(u) \right|. \quad (3)$$

To find the conditions on t that make Equation (3) hold, we give a bound to both the left and right hand side of the inequality. In detail:

1. We know from Lemma 24 (see Appendix C) that $|\mathbf{c}^{(t)}(u) - \mathbf{c}^{(t+1)}(u)| > \frac{1}{2} \lambda_k^t (1 - \lambda_2) |\mathbf{y}(u)|$ for every $u \in V$ and for every time $t < T_2$, where $T_2 = \Omega(n^{\frac{c}{3}})$, since by hypothesis $\lambda_k > \frac{1}{2}$ and $(1 - \lambda_2) \geq (\lambda_2 - \lambda_k) \Delta^{\frac{3}{2}} n^{1+c}$.
2. We know from Lemma 25 (see Appendix D) that $|\mathbf{e}^{(t)}(u)| \leq \lambda_{k+1}^t \sqrt{\Delta n}$, for every $u \in V$, and thus it follows that $|\mathbf{e}^{(t)}(u) - \mathbf{e}^{(t+1)}(u)| \leq |\mathbf{e}^{(t)}(u)| + |\mathbf{e}^{(t+1)}(u)| \leq 2\lambda_{k+1}^t \sqrt{\Delta n}$.

Combining Lemma 24 and Lemma 25, we get that if the following inequality holds, i.e.,

$$\frac{1}{2} \lambda_k^t (1 - \lambda_2) |\mathbf{y}(u)| > 2\lambda_{k+1}^t \sqrt{\Delta n}, \quad (4)$$

then also Equation (3) holds. By moving the terms dependent from t on the left hand side and by taking the logarithm of both sides, we can finally find the conditions on t such that Equation (4) is satisfied, i.e., all times $t > T_1$ where

$$T_1 = \log \left(\frac{4\sqrt{\Delta n}}{(1 - \lambda_2) |\mathbf{y}(u)|} \right) \cdot \frac{1}{\log \left(\frac{\lambda_k}{\lambda_{k+1}} \right)}.$$

20:12 Step-By-Step Community Detection in Volume-Regular Graphs

Note that $T_1 = \mathcal{O}(\log n / \log(\frac{\lambda_k}{\lambda_{k+1}}))$ and that $T_1 = \mathcal{O}(\log n)$ when $\frac{\lambda_k}{\lambda_{k+1}} = \Omega(1)$. In fact:

1. We know by hypothesis that the maximum weighted degree of a node is at most polynomial in n , i.e., $\Delta \leq \text{poly}(n)$.
2. We know from the Cheeger's inequality for weighted graphs (Theorem 15) the relation between the spectral gap and the Cheeger's constant of G , i.e., $1 - \lambda_2 \geq \frac{1}{2\Delta n}$, given that $1 - \lambda_2 \geq \frac{h_G^2}{2} \geq \frac{1}{2\Delta n}$.
3. We know from Lemma 20 (see Appendix B) that the length of the projection of the state vector on the stepwise vectors is not too small, i.e., $|\mathbf{y}(u)| \geq \frac{k}{\Delta n}$, w.h.p.

Since Lemma 24 holds for every time $t < T_2$, we conclude that there exists a time window $[T_1, T_2]$ such that, for every time $t \in [T_1, T_2]$ of the AVERAGING dynamics, it holds that $\text{sgn}(\mathbf{x}^{(t)}(u) - \mathbf{x}^{(t+1)}(u)) = \text{sgn}(\mathbf{c}^{(t)}(u) - \mathbf{c}^{(t+1)}(u))$, with high probability. Moreover, Lemma 24 tells us that $\text{sgn}(\mathbf{c}^{(t)}(u) - \mathbf{c}^{(t+1)}(u)) = \text{sgn}(\mathbf{y}(u))$, for every $u \in V$ and for every $t \in [T_1, T_2]$. Thus, $\text{sgn}(\mathbf{x}^{(t)}(u) - \mathbf{x}^{(t+1)}(u)) = \text{sgn}(\mathbf{y}(u))$, concluding the proof. \blacktriangleleft

► Lemma 11 (Different communities, different signs). *Let $G = (V, E, w)$ be a clustered k -volume-regular graph with maximum weighted degree $\Delta \leq \text{poly}(n)$ and $N = \mathcal{O}(\sqrt{k}/\Delta)$. For each pair of nodes $u \in V_i, v \in V_j$, with $i \neq j$, it holds that $\mathbf{P}(\text{sgn}(\mathbf{y}(u)) \neq \text{sgn}(\mathbf{y}(v))) = \Omega(1)$.*

Proof. Since the ordering of the communities (and consequent definition of the χ_i 's) is completely arbitrary, we can without loss of generality assume $i = 1$ and $j = 2$. From Lemma 10 we have that $\text{sgn}(\mathbf{x}^{(t)}(u) - \mathbf{x}^{(t+1)}(u)) = \text{sgn}(\mathbf{y}(u))$, for every $u \in V$, during a time interval $[T_1, T_2]$, w.h.p. Let us define $X(V_i) := \sum_{w \in V_i} \delta(w) \mathbf{x}(w)$.

Note that $\mathbf{y}(u) = \gamma_1 \chi_1(u)$ and $\mathbf{y}(v) = \gamma_1 \chi_1(v) + \gamma_2 \chi_2(v)$, since the other terms of the χ_i s are equal to 0 on the components relative to u and v . Thus, with some algebra, we get

$$\begin{aligned} \mathbf{y}(u) &= \frac{1}{m} \left[\frac{\hat{m}_1}{m_1} X(V_1) - X(V_2) - X(\hat{V}_2) \right], \\ \mathbf{y}(v) &= \frac{1}{m} \left[\frac{m_1 m_2 + m \hat{m}_2}{\hat{m}_1 m_2} X(V_2) - X(V_1) - X(\hat{V}_2) \right]. \end{aligned}$$

Note that, by linearity of expectation, $\mathbf{E}[X(V_i)] = 0$. Moreover, since the terms $\mathbf{x}(w)$ s are independent Rademacher random variables, we can write the standard deviation of $X(V_i)$ as

$$\sigma(X(V_i)) = \sqrt{\sum_{w \in V_i} \sigma^2(\mathbf{x}(w))} = \sqrt{\sum_{w \in V_i} \left(\mathbf{E}[\delta(w)^2 \mathbf{x}(w)^2] - \mathbf{E}[\delta(w) \mathbf{x}(w)]^2 \right)} = \sqrt{\sum_{w \in V_i} \delta(w)^2}.$$

Then we can upper and lower bound the standard deviation $\sigma(X(V_i))$ getting $\frac{m_i}{\sqrt{|V_i|}} \leq \sigma(X(V_i)) \leq \Delta \sqrt{|V_i|}$, where the lower bound follows from $\|\mathbf{d}\|_2 \geq \|\mathbf{d}\|_1 / \sqrt{|V_i|}$, where \mathbf{d}_i is the vector of weighted degrees of nodes in community V_i , and for the upper bound we used that $\delta(w) \leq \Delta$, for each $w \in V$.

Let us now define the following three events:

1. $E_1: X(V_1) \geq \sigma(X(V_1)) \implies X(V_1) \geq \frac{m_1}{\sqrt{|V_1|}} \geq \frac{\min_i m_i}{\sqrt{\max_i |V_i|}}$;
 2. $E_2: X(V_2) \leq -\sigma(X(V_2)) \implies X(V_2) \leq -\frac{m_2}{\sqrt{|V_2|}} \leq -\frac{\min_i m_i}{\sqrt{\max_i |V_i|}}$;
 3. $E_3: 0 \leq X(\hat{V}_2) \leq \varepsilon \sigma(X(\hat{V}_2)) \implies 0 \leq X(\hat{V}_2) \leq \varepsilon \Delta \sqrt{\sum_{i=3}^k |V_i|} \leq \varepsilon \Delta \sqrt{k \max_i |V_i|}$,
- with ε a suitable positive constant. When E_1, E_2, E_3 are true, i.e., with some constant probability, it holds that $\mathbf{y}(v) < 0$; as for $\mathbf{y}(u)$ we have that

$$\frac{\hat{m}_1}{m_1} X(V_1) - X(V_2) - X(\hat{V}_2) \geq \frac{\hat{m}_1}{m_1} \sigma(X(V_1)) + \sigma(X(V_2)) - \varepsilon \sigma(X(\hat{V}_2))$$

$$\geq \frac{k \min_i |V_i|}{\sqrt{\max_i |V_i|}} - \varepsilon \Delta \sqrt{k \max_i |V_i|}.$$

The previous inequality is greater than 0 whenever $\varepsilon < \frac{\sqrt{k}}{\Delta N}$. By hypothesis $\Delta N = \mathcal{O}(\sqrt{k})$ and thus $\frac{\sqrt{k}}{\Delta N} = \Omega(1)$, i.e., there is an $\varepsilon = \Omega(1)$ such that $\mathbf{y}(u) > 0$.

By approximating the random variables with Gaussian ones and using Berry-Esseen's theorem (Theorem 16), it is possible to show that all three events have probability at least constant; moreover, being the events independent, also $\mathbf{P}(E_1, E_2, E_3)$ is constant. ◀

Proof of Theorem 9. The proof proceeds by showing that the binary labeling of the nodes of G produced by the AVERAGING dynamics during the time window $[T_1, T_2]$ is such that the two conditions required by the definition of (ε, δ) -community sensitive algorithm (Definition 1) are met. The first condition follows directly from Lemma 10 and from the fact that \mathbf{y} is a “stepwise” vector, with $\varepsilon = \mathcal{O}(n^{-\frac{1}{2}})$ (see Lemma 20 for details on the probability). The second condition follows directly from Lemma 11. ◀

5 Extensions

In this section, we discuss extensions to bipartite graphs (Section 5.1) and to other non-clustered graph classes (Section 5.2).

5.1 Bipartite Graphs

Assume $G = (V, E, w)$ is a bipartite 2-volume-regular graph, i.e., $V = V_1 \cup V_2$, $E \subseteq V_1 \times V_2$ and G is volume-regular w.r.t. the bipartition (V_1, V_2) . In this case, basic properties of random walks imply that the AVERAGING dynamics does not converge to the global (weighted) average of the values, but it periodically oscillates. This follows since the state vector is mainly affected by the eigenvectors associated to the two eigenvalues of absolute value 1 (for bipartite graphs, λ_1 and λ_n). As a result, after a number of rounds depending on $1/\lambda_2$, the following happens: in even rounds, all nodes in V_i ($i = 1, 2$) have a state that is close to some local average μ_i ; in odd rounds these values are swapped, as shown in Equation (5). In even rounds (or, equivalently, in odd rounds) however, the states of nodes in V_1 would converge to μ_1 and those of nodes in V_2 would converge to μ_2 . Unfortunately, convergence to local averages does not eventually become monotonic in this case, since the eigenvector associated to λ_2 is no longer stepwise in general.⁵ However, we can easily modify the labeling scheme of the AVERAGING dynamics to perform *bipartiteness detection* as follows: Nodes apply the labeling rule every two time steps and they do it between the states of two consecutive rounds, i.e., each node $v \in V$ sets $\text{label}^{(2t)}(v) = 1$ if $\mathbf{x}^{(2t)}(v) \geq \mathbf{x}^{(2t-1)}(v)$ and $\text{label}^{(2t)}(v) = 0$ otherwise. We call this new protocol AVERAGING BIPARTITE dynamics.

Let $G = (V, E, w)$ be an edge-weighted undirected bipartite volume-regular graph. We denote with $W \in \mathbb{R}^{n \times n}$ the weighted adjacency matrix of G . Since G is undirected and bipartite, the matrix W can be written as

$$W = \begin{pmatrix} 0 & W_1 \\ W_2 & 0 \end{pmatrix} = \begin{pmatrix} 0 & W_1 \\ W_1^\top & 0 \end{pmatrix}.$$

⁵ This in turn follows since lumpable classes are already associated to $\mathbf{1}$ and χ .

20:14 Step-By-Step Community Detection in Volume-Regular Graphs

Thus, the transition matrix of a simple random walk on G , i.e., $P = D^{-1}W$ where D^{-1} is a diagonal matrix and $D_{ii} = \frac{1}{\delta(i)}$, has the form

$$P = \begin{pmatrix} 0 & P_1 \\ P_1^\top & 0 \end{pmatrix}.$$

Claim 12 shows that the spectrum of P is symmetric and it gives a relation between the eigenvectors of symmetric eigenvalues.

▷ **Claim 12.** Let $G = (V_1 \cup V_2, E, w)$ be an edge-weighted undirected bipartite graph with bipartition (V_1, V_2) and such that $|V_i| = n_i$. If $\mathbf{v} = (\mathbf{v}_1, \mathbf{v}_2)^\top$, with $\mathbf{v}_i \in \mathbb{R}^{n_i}$, is an eigenvector of P with eigenvalue λ , then $\mathbf{v}' = (\mathbf{v}_1, -\mathbf{v}_2)^\top$ is an eigenvector of P with eigenvalue $-\lambda$.

Proof. If $P\mathbf{v} = \lambda\mathbf{v}$ then we have that $P_1\mathbf{v}_2 = \lambda\mathbf{v}_1$ and $P_1^\top\mathbf{v}_2 = \lambda\mathbf{v}_2$. Using these two equalities we get that $P\mathbf{v}' = -\lambda\mathbf{v}'$. In fact,

$$P\mathbf{v}' = \begin{pmatrix} 0 & P_1 \\ P_1^\top & 0 \end{pmatrix} \begin{pmatrix} \mathbf{v}_1 \\ -\mathbf{v}_2 \end{pmatrix} = \begin{pmatrix} -P_1\mathbf{v}_2 \\ P_1^\top\mathbf{v}_1 \end{pmatrix} = -\lambda \begin{pmatrix} \mathbf{v}_1 \\ -\mathbf{v}_2 \end{pmatrix}.$$

◁

The transition matrix P is stochastic, thus the vector $\mathbf{1}$ (i.e., the vector of all ones) is an eigenvector associated to $\lambda_1 = 1$, that is the first largest eigenvalue of P . Claim 12 implies that $\boldsymbol{\chi} = \mathbf{1}_{V_1} - \mathbf{1}_{V_2}$ is an eigenvector of P with eigenvalue $\lambda_n = -1$.

As in Section 2, we write the state vector at time t using the spectral decomposition of P . Let $1 = \lambda_1 > \lambda_2 \geq \dots > \lambda_n = -1$ be the eigenvalues of P . We denote by $\mathbf{1} = \mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n = \boldsymbol{\chi}$ a family of n linearly independent eigenvectors of P , where each \mathbf{v}_i is the eigenvector associated to λ_i . Thus, we have that

$$\mathbf{x}^{(t)} = P^t \mathbf{x} = \sum_{i=1}^n \lambda_i^t \alpha_i \mathbf{v}_i = \alpha_1 \mathbf{1} + (-1)^t \alpha_n \boldsymbol{\chi} + \sum_{i=2}^{n-1} \lambda_i^t \alpha_i \mathbf{v}_i \quad (5)$$

where $\alpha_i = \frac{\langle D^{\frac{1}{2}} \mathbf{x}, D^{\frac{1}{2}} \mathbf{v}_i \rangle}{\|D^{\frac{1}{2}} \mathbf{v}_i\|^2}$. The last equation implies that $\mathbf{x}^{(t)} = P^t \mathbf{x}$ does not converge to some value as t tends to infinity, but oscillates. In particular, nodes in V_1 on even rounds and nodes in V_2 on odd rounds, converge to $\alpha_1 + \alpha_n$. Instead in the symmetric case, i.e., odd rounds for nodes in V_1 and even rounds for nodes in V_2 , the process converges to $\alpha_1 - \alpha_n$. These quantities are proportional to the weighted average of the initial values in the first and in the second partition, respectively.

Lemma 13 shows that AVERAGING BIPARTITE dynamics performs bipartiteness detection in $\mathcal{O}(\log n / \log(1/\lambda_2))$ rounds. Note that if $\log(1/\lambda_2) = \Omega(1)$, then the AVERAGING BIPARTITE dynamics takes logarithmic time to find the bipartition.

► **Lemma 13.** *Let $G = (V, E, w)$ be an edge-weighted bipartite volume-regular graph with bipartition V_1, V_2 and maximum weighted degree $\Delta \leq \text{poly}(n)$. Then for every time $t > T$, with $T = \mathcal{O}(\log n / \log(1/\lambda_2))$, the AVERAGING BIPARTITE dynamics is a $(\mathcal{O}(n^{-\frac{1}{2}}), \mathcal{O}(1))$ -community sensitive algorithm, w.h.p.*

Proof of Lemma 13. We assume that the coloring rule is applied between every even and every odd round (conversely, the signs of the nodes in the analysis are swapped). Recall the definition of the *error contribution*, namely $\mathbf{e}^{(t)}(u) = \sum_{i=2}^{n-1} \lambda_i^t \alpha_i \mathbf{v}_i(u)$. We compute the difference between the state vectors of two consecutive steps by using Equation (5), namely

$$\begin{aligned}\mathbf{x}^{(2t)} - \mathbf{x}^{(2t+1)} &= \alpha_1 \mathbf{1} + (-1)^{2t} \alpha_n \boldsymbol{\chi} + \mathbf{e}^{(2t)} - \alpha_1 \mathbf{1} - (-1)^{2t+1} \alpha_n \boldsymbol{\chi} - \mathbf{e}^{(2t+1)} \\ &= 2\alpha_n \boldsymbol{\chi} + \mathbf{e}^{(2t)} - \mathbf{e}^{(2t+1)}.\end{aligned}$$

We want to find a time T such that for every $t > T$ the sign of a node $u \in V$ depends only on $\boldsymbol{\chi}(u)$. Formally, $\text{sgn}(\mathbf{x}^{(2t)}(u) - \mathbf{x}^{(2t+1)}(u)) = \text{sgn}(\alpha_n \boldsymbol{\chi})$. The last equation holds whenever

$$\begin{aligned}2|\alpha_n \boldsymbol{\chi}(u)| &> |\mathbf{e}^{(2t)}(u) - \mathbf{e}^{(2t+1)}(u)| \\ 2|\alpha_n| &> |\mathbf{e}^{(2t)}(u) - \mathbf{e}^{(2t+1)}(u)|.\end{aligned}\tag{6}$$

We upper bound $|\mathbf{e}^{(2t)}(u) - \mathbf{e}^{(2t+1)}(u)|$ by using Lemma 25. We get that $|\mathbf{e}^{(2t)}(u) - \mathbf{e}^{(2t+1)}(u)| \leq 2\lambda_2^{2t} \sqrt{\Delta n}$. We get that Equation (6) holds if the following holds:

$$\begin{aligned}|\alpha_n| &> \lambda_2^{2t} \sqrt{\Delta n} \\ \left(\frac{1}{\lambda_2}\right)^{2t} &> \frac{\sqrt{\Delta n}}{|\alpha_n|} \\ 2t &> \log\left(\frac{\sqrt{\Delta n}}{|\alpha_n|}\right) \frac{1}{\log(1/\lambda_2)}.\end{aligned}$$

In order to find the time t which makes the last inequality hold, we provide a lower bound on $|\alpha_n|$, showing that it is not too small, with high probability. Recall that $\alpha_i = \frac{\langle D^{\frac{1}{2}} \mathbf{x}, D^{\frac{1}{2}} \mathbf{v}_i \rangle}{\|D^{\frac{1}{2}} \mathbf{v}_i\|^2}$ and thus

$$\alpha_n = \frac{\langle D^{\frac{1}{2}} \mathbf{x}, D^{\frac{1}{2}} \boldsymbol{\chi} \rangle}{\|D^{\frac{1}{2}} \boldsymbol{\chi}\|^2} = \frac{1}{\text{vol}(V)} \sum_{v \in V} \delta(v) \mathbf{x}(v) \boldsymbol{\chi}(v),$$

where $\text{vol}(V) = \sum_{v \in V} \delta(v)$. We get the lower bound, with high probability, by showing that

$$\mathbf{P}\left(|\alpha_n| \leq \frac{1}{\Delta n}\right) \leq \mathbf{P}\left(|\alpha_n| \leq \frac{1}{\text{vol}(V)}\right) = \mathbf{P}\left(\left|\sum_{v \in V} \delta(v) \mathbf{x}(v) \boldsymbol{\chi}(v)\right| \leq 1\right) \stackrel{(a)}{=} \mathcal{O}\left(\frac{1}{\sqrt{n}}\right)$$

where in (a) we apply Theorem 17. Indeed this last inequality implies that $|\alpha_n| > \frac{1}{\Delta n}$ with high probability. The thesis then follows from the above bound on $|\alpha_n|$ and from the hypothesis on $\Delta \leq \text{poly}(n)$. \blacktriangleleft

5.2 Other non-clustered volume-regular graphs

Consider k -volume-regular graphs whose k stepwise eigenvectors are associated to the k largest eigenvalues, in absolute value. These graphs include many k -partite graphs (e.g., regular ones), graphs that are “close” to being k -partite (i.e., ones that would become k -partite upon removal of a few edges). Differently from the clustered case (Theorem 9) some of the k eigenvalues can in general be negative.

Consider the following variant of the labeling scheme of the AVERAGING dynamics, in which nodes apply their labeling rule only on even rounds, comparing their value with the one they held at the end of the last even round, i.e., each node $v \in V$ sets $\text{label}^{(2t)}(v) = 1$ if $\mathbf{x}^{(2t)}(v) \geq \mathbf{x}^{(2t-2)}(v)$ and $\text{label}^{(2t)}(v) = 0$ otherwise.

Since the above protocol amounts to only taking even powers of eigenvalues, the analysis of this modified protocol proceeds along the same lines as the clustered case, while the results of Theorem 9 seamlessly extend to this class of graphs.

6 Conclusions

The focus of this work is on heuristics that implicitly perform spectral graph clustering, without explicitly computing the main eigenvectors of a matrix describing connectivity properties of the underlying network (typically, its Laplacian or a related matrix). In this perspective, we extended the work of Becchetti et al. [6] in several ways. In particular, for k communities, [6] considered an extremely regular case, in which the second eigenvalue of the (normalized) Laplacian has algebraic and geometric multiplicities $k - 1$ and the corresponding eigenspace is spanned by a basis of indicator vectors. We considered a more general case in which the first k eigenvalues are in general different, but the span of the corresponding eigenvectors again admits a base of indicator vectors. We also made a connection between this stepwise property and lumpability properties of the underlying random walk, which results in a class of volume-regular graphs, that may not have constant degree, nor exhibit balanced communities.

Though far from conclusive, we believe our results point to potentially interesting directions for future research. In general, our analysis sheds further light on the connections between temporal evolution of the power method and spectral-related clustering properties of the underlying network. At the same time, we showed that variants of the AVERAGING dynamics (and/or its labeling rule) might be useful in addressing different problems and/or other graph classes, as the examples given in Section 5.1 suggest. On the other hand, identifying k hidden partitions using the algorithm presented in [6] requires relatively strong assumptions on the k main eigenvalues and knowledge of an upper bound to the graph size,⁶ while the analysis becomes considerably more intricate than the perfectly regular and completely balanced case addressed in [6]. Some aspects of our analysis (e.g., the aforementioned presence of a size-dependent time window in which the labeling rule has to be applied) suggest that more sophisticated variants of the AVERAGING dynamics might be needed to express the full power of a spectral method that explicitly computes the k main eigenvectors of a graph-related matrix. While we believe this goal can be achieved, designing and analyzing such an algorithm might prove a challenging task.

References

- 1 Emmanuel Abbe, Afonso S. Bandeira, and Georgina Hall. Exact Recovery in the Stochastic Block Model. *IEEE Trans. Information Theory*, 62(1):471–487, 2016. doi:10.1109/TIT.2015.2490670.
- 2 Emmanuel Abbe and Colin Sandon. Detection in the stochastic block model with multiple clusters: proof of the achievability conjectures, acyclic BP, and the information-computation gap. *CoRR*, abs/1512.09080, 2015. arXiv:1512.09080.
- 3 Dana Angluin, James Aspnes, and David Eisenstat. A simple population protocol for fast robust approximate majority. *Distributed Computing*, 21(2):87–102, 2008. (Preliminary version appeared in DISC 2007). doi:10.1007/s00446-008-0059-z.
- 4 Michael J. Barber and John W. Clark. Detecting network communities by propagating labels under constraints. *Phys. Rev. E*, 80:026129, August 2009. doi:10.1103/PhysRevE.80.026129.
- 5 Luca Becchetti, Andrea E. F. Clementi, Pasin Manurangsi, Emanuele Natale, Francesco Pasquale, Prasad Raghavendra, and Luca Trevisan. Average Whenever You Meet: Opportunistic Protocols for Community Detection. In *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, pages 7:1–7:13, 2018. doi:10.4230/LIPIcs.ESA.2018.7.

⁶ As anecdotal experimental evidence suggests, the presence of a time window to perform labeling is not an artifact of our analysis.

- 6 Luca Becchetti, Andrea E. F. Clementi, Emanuele Natale, Francesco Pasquale, and Luca Trevisan. Find Your Place: Simple Distributed Algorithms for Community Detection. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 940–959, 2017. doi: 10.1137/1.9781611974782.59.
- 7 Ravi B. Boppana. Eigenvalues and Graph Bisection: An Average-Case Analysis (Extended Abstract). In *28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, USA, 27-29 October 1987*, pages 280–285, 1987. doi:10.1109/SFCS.1987.22.
- 8 Charles Bordenave, Marc Lelarge, and Laurent Massoulié. Nonbacktracking spectrum of random graphs: Community detection and nonregular Ramanujan graphs. *Ann. Probab.*, 46(1):1–71, January 2018. doi:10.1214/16-AOP1142.
- 9 Fan RK Chung. Laplacians of graphs and Cheeger’s inequalities. *Combinatorics, Paul Erdos is Eighty*, 2(157-172):13–2, 1996.
- 10 Amin Coja-Oghlan. *Spectral techniques, semidefinite programs, and random graphs*. PhD thesis, Habilitationsschrift, Humboldt Universität zu Berlin, Institut für Informatik, 2005.
- 11 Amin Coja-Oghlan. Graph Partitioning via Adaptive Spectral Techniques. *Comb. Probab. Comput.*, 19(2):227–284, March 2010. doi:10.1017/S0963548309990514.
- 12 Emilio Cruciani, Emanuele Natale, André Nusser, and Giacomo Scornavacca. Phase Transition of the 2-Choices Dynamics on Core-Periphery Networks. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*, pages 777–785, 2018. URL: <http://dl.acm.org/citation.cfm?id=3237499>.
- 13 Emilio Cruciani, Emanuele Natale, and Giacomo Scornavacca. Distributed Community Detection via Metastability of the 2-Choices Dynamics. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019.*, pages 6046–6053, 2019. URL: <https://aaai.org/ojs/index.php/AAAI/article/view/4560>, doi:10.1609/aaai.v33i01.33016046.
- 14 Aurelien Decelle, Florent Krzakala, Cristopher Moore, and Lenka Zdeborová. Asymptotic analysis of the stochastic block model for modular networks and its algorithmic applications. *Phys. Rev. E*, 84:066106, December 2011. doi:10.1103/PhysRevE.84.066106.
- 15 M.E Dyer and A.M Frieze. The solution of some random NP-hard problems in polynomial expected time. *Journal of Algorithms*, 10(4):451–489, 1989. doi:10.1016/0196-6774(89)90001-1.
- 16 P. Erdős. On a lemma of Littlewood and Offord. *Bull. Amer. Math. Soc.*, 51(12):898–902, December 1945. URL: <https://projecteuclid.org:443/euclid.bams/1183507531>.
- 17 Miroslav Fiedler. Laplacian of graphs and algebraic connectivity. *Banach Center Publications*, 25(1):57–70, 1989. URL: <http://eudml.org/doc/267812>.
- 18 Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75–174, 2010. doi:10.1016/j.physrep.2009.11.002.
- 19 Yehuda Hassin and David Peleg. Distributed Probabilistic Polling and Applications to Proportionate Agreement. *Inf. Comput.*, 171(2):248–268, 2001. doi:10.1006/inco.2001.3088.
- 20 Paul W. Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social Networks*, 5(2):109–137, 1983. doi:10.1016/0378-8733(83)90021-7.
- 21 Mark Jerrum and Gregory B. Sorkin. The Metropolis Algorithm for Graph Bisection. *Discrete Applied Mathematics*, 82(1-3):155–175, 1998. doi:10.1016/S0166-218X(97)00133-9.
- 22 John G. Kemeny and J. Laurie Snell. *Finite Markov chains*. D. van Nostrand Company, inc., Princeton, N.J., 1960.
- 23 David Kempe and Frank McSherry. A decentralized algorithm for spectral analysis. *J. Comput. Syst. Sci.*, 74(1):70–83, 2008. (Preliminary version appeared in STOC 2004). doi: 10.1016/j.jcss.2007.04.014.
- 24 Kishore Kothapalli, Sriram V. Pemmaraju, and Vivek Sardeshmukh. On the Analysis of a Label Propagation Algorithm for Community Detection. In *Distributed Computing and Networking, 14th International Conference, ICDCN 2013, Mumbai, India, January 3-6, 2013. Proceedings*, pages 255–269, 2013. doi:10.1007/978-3-642-35668-1_18.

- 25 Florent Krzakala, Cristopher Moore, Elchanan Mossel, Joe Neeman, Allan Sly, Lenka Zdeborová, and Pan Zhang. Spectral redemption in clustering sparse networks. *Proceedings of the National Academy of Sciences*, 110(52):20935–20940, 2013. doi:10.1073/pnas.1312486110.
- 26 Cornelius Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *J. Res. Natl. Bur. Stand. B*, 45:255–282, 1950. doi:10.6028/jres.045.026.
- 27 James R. Lee, Shayan Oveis Gharan, and Luca Trevisan. Multiway Spectral Partitioning and Higher-Order Cheeger Inequalities. *J. ACM*, 61(6):37:1–37:30, 2014. doi:10.1145/2665063.
- 28 Jure Leskovec, Anand Rajaraman, and Jeffrey D. Ullman. *Mining of Massive Datasets, 2nd Ed.* Cambridge University Press, 2014. URL: <http://www.mmds.org/>.
- 29 X. Liu and T. Murata. Advanced modularity-specialized label propagation algorithm for detecting communities in networks. *Physica A: Statistical Mechanics and its Applications*, 389(7):1493–1500, 2010. doi:10.1016/j.physa.2009.12.019.
- 30 David J. C. MacKay. *Information theory, inference, and learning algorithms.* Cambridge University Press, 2003.
- 31 Frederik Mallmann-Trenn, Cameron Musco, and Christopher Musco. Eigenvector Computation and Community Detection in Asynchronous Gossip Models. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 159:1–159:14, 2018. doi:10.4230/LIPIcs.ICALP.2018.159.
- 32 Frank McSherry. Spectral Partitioning of Random Graphs. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 529–537, 2001. doi:10.1109/SFCS.2001.959929.
- 33 S. J. Montgomery-Smith. The Distribution of Rademacher Sums. *Proceedings of the American Mathematical Society*, 109(2):517–522, 1990. URL: <http://www.jstor.org/stable/2048015>.
- 34 Joris M. Mooij and Hilbert J. Kappen. Sufficient Conditions for Convergence of the Sum-Product Algorithm. *IEEE Transactions on Information Theory*, 53(12):4422–4437, December 2007. doi:10.1109/TIT.2007.909166.
- 35 Elchanan Mossel, Joe Neeman, and Allan Sly. Belief propagation, robust reconstruction and optimal recovery of block models. *The Annals of Applied Probability*, 26(4):2211–2256, 2016. (Preliminary version appeared in COLT 2014).
- 36 Elchanan Mossel, Joe Neeman, and Allan Sly. A Proof of the Block Model Threshold Conjecture. *Combinatorica*, 38(3):665–708, 2018. doi:10.1007/s00493-016-3238-8.
- 37 Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On Spectral Clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*, pages 849–856, 2001. URL: <http://papers.nips.cc/paper/2092-on-spectral-clustering-analysis-and-an-algorithm>.
- 38 Richard Peng, He Sun, and Luca Zanetti. Partitioning Well-Clustered Graphs: Spectral Clustering Works! *SIAM J. Comput.*, 46(2):710–743, 2017. (Preliminary version appeared in COLT 2015). doi:10.1137/15M1047209.
- 39 Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Phys. Rev. E*, 76:036106, September 2007. doi:10.1103/PhysRevE.76.036106.
- 40 Irina Shevtsova. On the absolute constants in the Berry-Esseen-type inequalities. *Doklady Mathematics*, 89(3):378–381, May 2014. doi:10.1134/S1064562414030338.
- 41 Jianbo Shi and Jitendra Malik. Normalized Cuts and Image Segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8):888–905, 2000. (Preliminary version appeared in CVPR 1997). doi:10.1109/34.868688.
- 42 J Michael Steele. *The Cauchy-Schwarz master class: an introduction to the art of mathematical inequalities.* Cambridge University Press, 2004.
- 43 He Sun and Luca Zanetti. Distributed Graph Clustering and Sparsification. *CoRR*, abs/1711.01262, 2017. arXiv:1711.01262.
- 44 Jianjun Paul Tian and D. Kannan. Lumpability and Commutativity of Markov Processes. *Stochastic Analysis and Applications*, 24(3):685–702, 2006. doi:10.1080/07362990600632045.

- 45 Ulrike von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007. doi:10.1007/s11222-007-9033-z.
- 46 Yair Weiss. Correctness of Local Probability Propagation in Graphical Models with Loops. *Neural Computation*, 12(1):1–41, 2000. doi:10.1162/089976600300015880.

A Useful inequalities

► **Theorem 14** (Cauchy-Schwarz's inequality [42]). *For all vectors \mathbf{u}, \mathbf{v} of an inner product space it holds that $|\langle \mathbf{u}, \mathbf{v} \rangle|^2 \leq \langle \mathbf{u}, \mathbf{u} \rangle \cdot \langle \mathbf{v}, \mathbf{v} \rangle$, where $\langle \cdot, \cdot \rangle$ is the inner product.*

► **Theorem 15** (Cheeger's inequality [9]). *Let P be the transition matrix of a connected edge-weighted graph $G = (V, E, w)$ and let λ_2 be its second largest eigenvalue. Let $|E(S, V \setminus S)| = \sum_{u \in S, v \in V \setminus S} w(u, v)$ and $h_G = \min_{S: \text{vol}(S) \leq \frac{\text{vol}(V)}{2}} \frac{|E(S, V \setminus S)|}{\text{vol}(S)}$. Then $\frac{1-\lambda_2}{2} \leq h_G \leq \sqrt{2(1-\lambda_2)}$.*

► **Theorem 16** (Berry-Esseen's theorem [40]). *Let X_1, \dots, X_n be independent and identically distributed random variables with mean $\mu = 0$, variance $\sigma^2 > 0$, and third absolute moment $\rho < \infty$. Let $Y_n = \frac{1}{n} \sum_{i=1}^n X_i$; let F_n be the cumulative distribution function of $\frac{Y_n \sqrt{n}}{\sigma}$; let Φ the cumulative distribution function of the standard normal distribution. Then, there exists a positive constant $C < 0.4748$ such that, for all x and for all n , $|F_n(x) - \Phi(x)| \leq \frac{C\rho}{\sigma^3 \sqrt{n}}$.*

► **Theorem 17** (Littlewood-Offord's small ball [16]). *Let x_i be a Rademacher random variable (taking values ± 1 with probability $p = \frac{1}{2}$), let a_i be real constants such that $|a_i| \geq 1$, and let $X = \sum_{i=1}^n a_i x_i$. Then, for any $r \in \mathbb{R}$, it holds that $\mathbf{P}(|X - r| < 1) = \mathcal{O}\left(\frac{1}{\sqrt{n}}\right)$.*

► **Theorem 18** (Rademacher concentration bound [33]). *Let x_i be a Rademacher random variable (taking values ± 1 with probability $p = \frac{1}{2}$), let a_i be real constants, and let $X = \sum_{i=1}^n a_i x_i$. Then, it holds that $\mathbf{P}(|X| > t \|\mathbf{a}\|_2) \leq 2e^{-\frac{t^2}{2}}$, where $\|\mathbf{a}\|_2$ is the Euclidean norm of the vector $\mathbf{a} = (a_1, \dots, a_n)$.*

B Length of the projection of the state vector

In this section we show that every component of \mathbf{y} , i.e., the projection of the contracted initial state vector $D^{\frac{1}{2}} \mathbf{x}$ on the contracted vectors $D^{\frac{1}{2}} \chi_i$ s, is not too small, w.h.p. (Lemma 20). This result is used in Appendices C and D.

▷ **Claim 19.** Let $\alpha(u, v) = \sum_{i=1}^{k-1} \frac{\chi_i(u) \chi_i(v)}{\hat{m}_i}$. For every pair of nodes $u, v \in V$ it holds that $\min_{u, v \in V} |\alpha(u, v)| \geq \frac{k}{\Delta n}$.

Proof. Let $u \in V_l$ and $v \in V_h$, for some $l, h \in [k]$. We divide the proof in two cases. First, we assume that $l = h$, then we handle the case $l \neq h$. Without loss of generality, we assume $m_1 \leq \dots \leq m_k$ and consequently $m = \hat{m}_0 \geq \hat{m}_1 \geq \dots \geq \hat{m}_{k-1} = m_k$.

■ Let us suppose $l = h$. Then

$$\min_{u, v \in V} |\alpha(u, v)| = \min_{u, v \in V} \left(\sum_{i < \min\{h, l\}} \frac{m_i}{\hat{m}_i \hat{m}_{i-1}} + \frac{\hat{m}_l}{m_l \hat{m}_{l-1}} \right) \geq \frac{\hat{m}_1}{m_1 \hat{m}_0} \geq \frac{k}{m} \geq \frac{k}{\Delta n}.$$

■ Let us suppose $l \neq h$. In this case, $\alpha(u, v) = \sum_{i < \min\{h, l\}} \frac{m_i}{\hat{m}_i \hat{m}_{i-1}} - 1 < 0$. In fact

$$\begin{aligned} \sum_{i < \min\{h, l\}} \frac{m_i}{\hat{m}_i \hat{m}_{i-1}} &= \sum_{i < \min\{h, l\}} \frac{m_i}{\left(\sum_{j=i+1}^k m_j\right) \left(\sum_{j=i}^k m_j\right)} \\ &\stackrel{(a)}{\leq} \sum_{i < \min\{h, l\}} \frac{m_i}{(k-i)(k-i+1)m_i^2} = \sum_{i < \min\{h, l\}} \frac{1}{(k-i)(k-i+1)m_i} \\ &\leq \sum_{i < \min\{h, l\}} \frac{1}{(k-i)(k-i+1)} \leq \sum_{j=1}^k \frac{1}{j(j+1)} < 1, \end{aligned}$$

20:20 Step-By-Step Community Detection in Volume-Regular Graphs

where in (a) we use the assumption on the ordering of the volumes of the communities, i.e., $m_i \leq m_j$ for every $i \leq j$. Since $\alpha(u, v) < 0$, we have that $|\alpha(u, v)| = 1 - \sum_{i < \min\{h, l\}} \frac{m_i}{\hat{m}_i \hat{m}_{i-1}}$ and thus

$$\min_{u, v \in V} |\alpha(u, v)| = 1 - \max_{u, v \in V} \left(\sum_{i < \min\{h, l\}} \frac{m_i}{\hat{m}_i \hat{m}_{i-1}} \right) \geq 1 - \frac{(k-2)m_k}{m_k^2} = 1 - \frac{k-2}{m_k}.$$

Note that $m_k \geq \frac{n}{k}$ and, given that $k \leq \sqrt{n}$, we get $m_k \geq k$. Thus $1 - \frac{k-2}{m_k} \geq \frac{2}{k} \geq \frac{k}{\Delta n}$. \triangleleft

► **Lemma 20** (Length of the projection of the state vector). *For every $u \in V$, it holds that $\mathbf{P}(|\mathbf{y}(u)| \geq \frac{k}{\Delta n}) \geq 1 - \mathcal{O}\left(\frac{1}{\sqrt{n}}\right)$.*

Proof. Let us write $\mathbf{y}(u) = \sum_{i=1}^{k-1} \gamma_i \boldsymbol{\chi}_i(u)$ in terms of \mathbf{x} . Recall that $\gamma_i = \frac{\mathbf{x}^\top D \boldsymbol{\chi}_i}{\|D^{1/2} \boldsymbol{\chi}_i\|^2}$ and that $\boldsymbol{\chi}_i = \sqrt{\frac{\hat{m}_i}{m_i}} \mathbf{1}_{V_i} - \sqrt{\frac{m_i}{\hat{m}_i}} \mathbf{1}_{\hat{V}_i}$. Thus, we get $\|D^{1/2} \boldsymbol{\chi}_i\|^2 = \frac{\hat{m}_i}{m_i} \sum_{v \in V_i} \delta(v) + \frac{m_i}{\hat{m}_i} \sum_{v \in \hat{V}_i} \delta(v) = \hat{m}_i + m_i = \hat{m}_{i-1}$, where $\hat{m}_0 := m = \sum_{v \in V} \delta(v)$. Now, we can rewrite $\mathbf{y}(u)$ as

$$\begin{aligned} \mathbf{y}(u) &= \sum_{i=1}^{k-1} \gamma_i \boldsymbol{\chi}_i(u) = \sum_{i=1}^{k-1} \frac{\mathbf{x}^\top D \boldsymbol{\chi}_i}{\hat{m}_{i-1}} \boldsymbol{\chi}_i(u) = \sum_{i=1}^{k-1} \left(\sum_{v \in V} \frac{\delta(v) \mathbf{x}(v) \boldsymbol{\chi}_i(v)}{\hat{m}_{i-1}} \right) \boldsymbol{\chi}_i(u) \\ &= \sum_{v \in V} \left(\sum_{i=1}^{k-1} \frac{\boldsymbol{\chi}_i(u) \boldsymbol{\chi}_i(v)}{\hat{m}_{i-1}} \right) \delta(v) \mathbf{x}(v) = \sum_{v \in V} \alpha(u, v) \delta(v) \mathbf{x}(v), \end{aligned}$$

where $\alpha(u, v) := \sum_{i=1}^{k-1} \frac{\boldsymbol{\chi}_i(u) \boldsymbol{\chi}_i(v)}{\hat{m}_{i-1}}$. Note that, for every $u \in V_l$ and $v \in V_h$, with $l, h \in [k]$,

$$\boldsymbol{\chi}_i(u) \boldsymbol{\chi}_i(v) = \begin{cases} \frac{m_i}{\hat{m}_i} & \text{if } i < \min(l, h), \\ \frac{\hat{m}_i}{m_i} & \text{if } i = \min(l, h) \text{ and } l = h, \\ -1 & \text{if } i = \min(l, h) \text{ and } l \neq h, \\ 0 & \text{if } i > \min(l, h). \end{cases}$$

$$\text{Thus } \alpha(u, v) = \sum_{i=1}^{k-1} \frac{\boldsymbol{\chi}_i(u) \boldsymbol{\chi}_i(v)}{\hat{m}_{i-1}} = \begin{cases} \sum_{i < \min\{h, l\}} \frac{m_i}{\hat{m}_i \hat{m}_{i-1}} + \frac{\hat{m}_l}{m_l \hat{m}_{l-1}} & \text{if } h = l, \\ \sum_{i < \min\{h, l\}} \frac{m_i}{\hat{m}_i \hat{m}_{i-1}} - 1 & \text{if } h \neq l. \end{cases}$$

We apply Theorem 17 and Claim 19 to prove that the length of the projection of the state vector \mathbf{x} on $\{\boldsymbol{\chi}_i : i \in [k]\}$ is not too small, w.h.p.

$$\begin{aligned} \mathbf{P}(|\mathbf{y}(u)| \leq \frac{k}{\Delta n}) &= \mathbf{P}\left(|\sum_{v \in V} \alpha(u, v) \delta(v) \mathbf{x}(v)| \leq \frac{k}{\Delta n}\right) \\ &= \mathbf{P}\left(\left|\sum_{v \in V} \frac{\alpha(u, v)}{\min_{u, v} |\alpha(u, v)|} \delta(v) \mathbf{x}(v)\right| \leq \frac{k}{\Delta n \min_{u, v} |\alpha(u, v)|}\right) \\ &\stackrel{(a)}{\leq} \mathbf{P}\left(\left|\sum_{v \in V} \frac{\alpha(u, v)}{\min_{u, v} |\alpha(u, v)|} \delta(v) \mathbf{x}(v)\right| \leq 1\right) \stackrel{(b)}{\leq} \mathcal{O}\left(\frac{1}{\sqrt{n}}\right), \end{aligned}$$

where in (a) we use Claim 19 to upper bound with 1 the r.h.s. term in the probability; in (b) we can apply Theorem 17 given that $\min_v \delta(v) = 1$ and that $\left|\frac{\alpha(u, v)}{\min_{u, v} |\alpha(u, v)|}\right| \geq 1$. \blacktriangleleft

C Lower bound on the core contribution

In this section we provide a lower bound on the difference of the *core contribution* of the state vector between two consecutive time steps; we also show that the sign of a node depends only on the sign of \mathbf{y} (Lemma 24). In order to prove that we: (i) provide upper and lower bounds on $\mathbf{c}^{(t)}(u)$ (Claim 21), and (ii) we bound $\mathbf{c}^{(t)}(u) - \mathbf{c}^{(t+1)}(u)$ (Claim 22).

- ▷ **Claim 21.** Let $\mathbf{c}^{(t)} = \sum_{i=2}^k \lambda_i^t \alpha_i \mathbf{v}_i$. For every $u \in V$ it holds that:
- $\mathbf{c}^{(t)}(u) \geq \lambda_k^t \sum_{i=2}^k \alpha_i \mathbf{v}_i(u) + t \lambda_2^{t-1} (\lambda_2 - \lambda_k) \sum_{i: \alpha_i \mathbf{v}_i(u) < 0} \alpha_i \mathbf{v}_i(u)$;
 - $\mathbf{c}^{(t)}(u) \leq \lambda_k^t \sum_{i=2}^k \alpha_i \mathbf{v}_i(u) + t \lambda_2^{t-1} (\lambda_2 - \lambda_k) \sum_{i: \alpha_i \mathbf{v}_i(u) > 0} \alpha_i \mathbf{v}_i(u)$.

Proof. Let us start with the lower bound.

$$\begin{aligned}
\mathbf{c}^{(t)}(u) &= \sum_{i=2}^k \lambda_i^t \alpha_i \mathbf{v}_i(u) = \sum_{i: \alpha_i \mathbf{v}_i(u) > 0} \lambda_i^t \alpha_i \mathbf{v}_i(u) + \sum_{i: \alpha_i \mathbf{v}_i(u) < 0} \lambda_i^t \alpha_i \mathbf{v}_i(u) \\
&\geq \lambda_k \sum_{i: \alpha_i \mathbf{v}_i(u) > 0} \lambda_i^{t-1} \alpha_i \mathbf{v}_i(u) + \lambda_2 \sum_{i: \alpha_i \mathbf{v}_i(u) < 0} \lambda_i^{t-1} \alpha_i \mathbf{v}_i(u) \\
&\stackrel{(a)}{=} \lambda_k \sum_{i=2}^k \lambda_i^{t-1} \alpha_i \mathbf{v}_i(u) + (\lambda_2 - \lambda_k) \sum_{i: \alpha_i \mathbf{v}_i(u) < 0} \lambda_i^{t-1} \alpha_i \mathbf{v}_i(u) \\
&\stackrel{(b)}{=} \lambda_k \left[\lambda_k \sum_{i=2}^k \lambda_i^{t-2} \alpha_i \mathbf{v}_i(u) + (\lambda_2 - \lambda_k) \sum_{i: \alpha_i \mathbf{v}_i(u) < 0} \lambda_i^{t-2} \alpha_i \mathbf{v}_i(u) \right] \\
&\quad + (\lambda_2 - \lambda_k) \lambda_2^{t-1} \sum_{i: \alpha_i \mathbf{v}_i(u) < 0} \alpha_i \mathbf{v}_i(u) \\
&= \lambda_k^2 \sum_{i=2}^k \lambda_i^{t-2} \alpha_i \mathbf{v}_i(u) + \lambda_k (\lambda_2 - \lambda_k) \sum_{i: \alpha_i \mathbf{v}_i(u) < 0} \lambda_i^{t-2} \alpha_i \mathbf{v}_i(u) \\
&\quad + (\lambda_2 - \lambda_k) \lambda_2^{t-1} \sum_{i: \alpha_i \mathbf{v}_i(u) < 0} \alpha_i \mathbf{v}_i(u) \\
&\geq \lambda_k^2 \sum_{i=2}^k \lambda_i^{t-2} \alpha_i \mathbf{v}_i(u) + \lambda_k \lambda_2^{t-2} (\lambda_2 - \lambda_k) \sum_{i: \alpha_i \mathbf{v}_i(u) < 0} \alpha_i \mathbf{v}_i(u) \\
&\quad + (\lambda_2 - \lambda_k) \lambda_2^{t-1} \sum_{i: \alpha_i \mathbf{v}_i(u) < 0} \alpha_i \mathbf{v}_i(u) \\
&= \lambda_k^2 \sum_{i=2}^k \lambda_i^{t-2} \alpha_i \mathbf{v}_i(u) + (\lambda_k \lambda_2^{t-2} + \lambda_2^{t-1}) (\lambda_2 - \lambda_k) \sum_{i: \alpha_i \mathbf{v}_i(u) < 0} \alpha_i \mathbf{v}_i(u) \\
&\geq \lambda_k^2 \sum_{i=2}^k \lambda_i^{t-2} \alpha_i \mathbf{v}_i(u) + 2 \lambda_2^{t-1} (\lambda_2 - \lambda_k) \sum_{i: \alpha_i \mathbf{v}_i(u) < 0} \alpha_i \mathbf{v}_i(u) \\
&\geq \dots \\
&\geq \lambda_k^t \sum_{i=2}^k \alpha_i \mathbf{v}_i(u) + t \lambda_2^{t-1} (\lambda_2 - \lambda_k) \sum_{i: \alpha_i \mathbf{v}_i(u) < 0} \alpha_i \mathbf{v}_i(u),
\end{aligned}$$

where in (a) we add and subtract $\lambda_k \sum_{i: \alpha_i \mathbf{v}_i(u) < 0} \lambda_i^{t-1} \alpha_i \mathbf{v}_i(u)$; in (b) we iterate the same reasoning on the first term only.

The upper bound follows with analogous calculations. \triangleleft

By using Claim 21 is possible to give upper and lower bounds on the difference between the *core contribution* in two consecutive rounds.

- ▷ **Claim 22.** Let $\mathbf{c}^{(t)} = \sum_{i=2}^k \lambda_i^t \alpha_i \mathbf{v}_i$ and let $\lambda_2 > \lambda_k > \frac{1}{2}$. For every $u \in V$, it holds that
- $\mathbf{c}^{(t)}(u) - \mathbf{c}^{(t+1)}(u) \geq \lambda_k^t (1 - \lambda_2) \sum_{i=2}^k \alpha_i \mathbf{v}_i(u) + (t+1) \lambda_2^t (\lambda_2 - \lambda_k) \sum_{i: \alpha_i \mathbf{v}_i(u) < 0} \alpha_i \mathbf{v}_i(u)$;
 - $\mathbf{c}^{(t)}(u) - \mathbf{c}^{(t+1)}(u) \leq \lambda_k^t (1 - \lambda_2) \sum_{i=2}^k \alpha_i \mathbf{v}_i(u) + (t+1) \lambda_2^t (\lambda_2 - \lambda_k) \sum_{i: \alpha_i \mathbf{v}_i(u) > 0} \alpha_i \mathbf{v}_i(u)$.

The proof of Lemma 24 requires one extra claim about the coefficients β_i , i.e., the ones such that $\alpha_i \mathbf{v}_i = \beta_i D^{\frac{1}{2}} \mathbf{w}_i$. This last bound is shown in Claim 23.

- ▷ **Claim 23.** Let $\mathbf{x} \in \{-1, 1\}^n$ be a Rademacher random vector. Let $D \in \mathbb{R}^{n \times n}$ be a positive diagonal matrix with maximum element $\Delta = \max_i D_{ii}$ and let $\mathbf{w} \in \mathbb{R}^n$ be a vector such that $\|\mathbf{w}\|_2 = 1$. Let $\beta = \langle \mathbf{x}, D^{\frac{1}{2}} \mathbf{w} \rangle$. It holds that $|\beta| \leq \sqrt{\Delta \log n}$, with high probability.

Proof. Note that β is a weighted sum of Rademacher random variables with i -th coefficient equal to $(D^{\frac{1}{2}} \mathbf{w})(i)$ and that $\|D^{\frac{1}{2}} \mathbf{w}\|_2 = \sqrt{\sum_{i=1}^n \delta(i) \mathbf{w}(i)^2} \leq \sqrt{\Delta}$, since by hypothesis $\|\mathbf{w}\|_2 = 1$ and thus $\|D^{\frac{1}{2}} \mathbf{w}\|_2^2$ is a convex combination of the diagonal elements of D . Let $t = \sqrt{\log n}$; by applying Theorem 18 we get $\mathbf{P}\left(|\beta| > \sqrt{\Delta \log n}\right) \leq \mathbf{P}\left(|\beta_i| > t \|D^{\frac{1}{2}} \mathbf{w}\|_2\right) \leq 2e^{-\frac{\log n}{2}} = \mathcal{O}\left(\frac{1}{n}\right)$. Thus $|\beta| \leq \sqrt{\Delta \log n}$, with high probability. \triangleleft

We are now ready to state and prove Lemma 24.

20:22 Step-By-Step Community Detection in Volume-Regular Graphs

► **Lemma 24** (Lower bound on the core contribution). *Let $\mathbf{c}^{(t)} = \sum_{i=2}^k \lambda_i^t \alpha_i \mathbf{v}_i$. Let $\lambda_k > \frac{1}{2}$ and $\frac{1-\lambda_2}{\lambda_2-\lambda_k} \geq \Delta^{\frac{2}{3}} n^{1+c}$, for some positive constant c . For every $u \in V$ and for every time $t < T_2$, such that $T_2 = \Omega(n^{c/3})$, the two following conditions hold, w.h.p.:*

- $|\mathbf{c}^{(t)}(u) - \mathbf{c}^{(t+1)}(u)| \geq \frac{1}{2} \lambda_k^t (1 - \lambda_2) |\mathbf{y}(u)|;$
- $\text{sgn}(\mathbf{c}^{(t)}(u) - \mathbf{c}^{(t+1)}(u)) = \text{sgn}(\mathbf{y}(u)).$

Proof. We show the lower bound in the time window. To do that, first we suppose that $\mathbf{c}^{(t)}(u) - \mathbf{c}^{(t+1)}(u) > 0$ and show that the claim holds; then we show that the claim also holds when $\mathbf{c}^{(t)}(u) - \mathbf{c}^{(t+1)}(u) < 0$.

Let us suppose $\mathbf{c}^{(t)}(u) - \mathbf{c}^{(t+1)}(u) > 0$. If $\mathbf{y}(u) < 0$ the thesis follows directly; then let us suppose $\mathbf{y}(u) \geq 0$. From Claim 22 we have that $\mathbf{c}^{(t)}(u) - \mathbf{c}^{(t+1)}(u) \geq \lambda_k^t (1 - \lambda_2) \sum_{i=2}^k \alpha_i \mathbf{v}_i(u) + (t+1) \lambda_2^t (\lambda_2 - \lambda_k) \sum_{i: \alpha_i \mathbf{v}_i(u) < 0} \alpha_i \mathbf{v}_i(u)$. In order to prove the lemma in this first case, we need to show that

$$\frac{1}{2} \lambda_k^t (1 - \lambda_2) \sum_{i=2}^k \alpha_i \mathbf{v}_i(u) > -(t+1) \lambda_2^t (\lambda_2 - \lambda_k) \sum_{i: \alpha_i \mathbf{v}_i(u) < 0} \alpha_i \mathbf{v}_i(u). \quad (7)$$

We lower bound the left hand side and upper bound the right hand side. For the lower bound we apply Lemma 4 to get that $\sum_{i=2}^k \alpha_i \mathbf{v}_i(u) = \mathbf{y}(u)$ and Lemma 20 to get $\mathbf{y}(u) \geq \frac{k}{\Delta n}$, with high probability. For the upper bound, instead, we rely on Claim 23 and on the fact that $\alpha_i \mathbf{v}_i = \beta_i D^{\frac{1}{2}} \mathbf{w}_i$, for every $i \in [n]$. Indeed

$$-\sum_{i: \alpha_i \mathbf{v}_i(u) < 0} \alpha_i \mathbf{v}_i(u) = -\sum_{i: \beta_i \mathbf{w}_i(u) < 0} \frac{\beta_i}{\sqrt{\delta(u)}} \mathbf{w}_i(u) \stackrel{(a)}{\leq} k \sqrt{\Delta \log n},$$

where in (a) we can apply Claim 23 since $\|\mathbf{w}_i\|_2 = 1$ for every $i \in [k]$ and $\beta_i = \langle D^{\frac{1}{2}} \mathbf{x}, \mathbf{w}_i \rangle$. By combining lower and upper bounds, we get

$$\begin{aligned} \frac{1}{2} \lambda_k^t (1 - \lambda_2) \sum_{i=2}^k \alpha_i \mathbf{v}_i(u) &> -(t+1) \lambda_2^t (\lambda_2 - \lambda_k) \sum_{i: \alpha_i \mathbf{v}_i(u) < 0} \alpha_i \mathbf{v}_i(u) \\ \frac{1}{2} \lambda_k^t (1 - \lambda_2) \frac{k}{\Delta n} &> (t+1) \lambda_2^t (\lambda_2 - \lambda_k) k \sqrt{\Delta \log n} \\ \left(\frac{\lambda_2}{\lambda_k}\right)^t (t+1) &< \frac{1}{2} \frac{1-\lambda_2}{\lambda_2-\lambda_k} \frac{1}{\Delta^{\frac{2}{3}} n \sqrt{\log n}}. \end{aligned} \quad (8)$$

By hypothesis we have that $\frac{1-\lambda_2}{\lambda_2-\lambda_k} \geq \Delta^{\frac{2}{3}} n^{1+c}$ and that $\lambda_k > \frac{1}{2}$. Thus, we can derive an upper bound for $\frac{\lambda_2}{\lambda_k}$, namely

$$\frac{\lambda_2}{\lambda_k} = 1 + \frac{\lambda_2 - \lambda_k}{\lambda_k} \leq 1 + \frac{1 - \lambda_2}{\lambda_k \Delta^{\frac{2}{3}} n^{1+c}} \leq 1 + \frac{1}{\Delta^{\frac{2}{3}} n^{1+c}} \leq 1 + \frac{1}{n^{\frac{c}{3}}}. \quad (9)$$

Moreover, by the hypothesis on $\frac{1-\lambda_2}{\lambda_2-\lambda_k}$, we know that

$$\frac{1}{2} \frac{1-\lambda_2}{\lambda_2-\lambda_k} \frac{1}{\Delta^{\frac{2}{3}} n \sqrt{\log n}} \geq \frac{1}{2} n^{\frac{c}{2}}. \quad (10)$$

We apply Equations (9) and (10) to Equation (8) to find a time T_2 such that for every $t \leq T_2$ the lemma holds, and get $\left(1 + \frac{1}{n^{\frac{c}{3}}}\right)^t (t+1) < \frac{1}{2} n^{\frac{c}{2}}$. Let $T_2 = n^{\frac{c}{3}}$. Note that $\left(1 + \frac{1}{n^{\frac{c}{3}}}\right)^t \leq e$ for every time $t \leq T_2$; thus, for every time $t < T_2$, it also holds that $e(t+1) < \frac{1}{2} n^{\frac{c}{2}}$. We conclude that, in this first case, there exists a time $T_2 = \Omega(n^{\frac{c}{3}})$ such that, for every $t < T_2$,

$$\mathbf{c}^{(t)}(u) - \mathbf{c}^{(t+1)}(u) \geq \frac{1}{2} \lambda_k^t (1 - \lambda_2) \sum_{i=1}^{k-1} \gamma_i \chi_i(u). \quad (11)$$

Let us now suppose $\mathbf{c}^{(t)}(u) - \mathbf{c}^{(t+1)}(u) < 0$. As before, if $\mathbf{y}(u) > 0$ the thesis directly follows; then let us suppose $\mathbf{y}(u) \leq 0$. From Claim 22 we have that $\mathbf{c}^{(t)}(u) - \mathbf{c}^{(t+1)}(u) \leq \lambda_k^t(1 - \lambda_2) \sum_{i=2}^k \alpha_i \mathbf{v}_i(u) + (t+1)\lambda_2^t(\lambda_2 - \lambda_k) \sum_{i:\alpha_i \mathbf{v}_i(u) > 0} \alpha_i \mathbf{v}_i(u)$. Similarly to the previous case, in order to prove the lemma we need to show that

$$\frac{1}{2}\lambda_k^t(1 - \lambda_2) \sum_{i=2}^k \alpha_i \mathbf{v}_i(u) \leq -(t+1)\lambda_2^t(\lambda_2 - \lambda_k) \sum_{i:\alpha_i \mathbf{v}_i(u) > 0} \alpha_i \mathbf{v}_i(u). \quad (12)$$

Again, we upper bound the left hand side using Lemma 4 and Lemma 20 and getting $\sum_{i=2}^k \alpha_i \mathbf{v}_i(u) = \sum_{i=1}^{k-1} \gamma_i \chi_i(u) \leq -\frac{k}{\Delta n}$, with high probability. As for the right hand side we use Claim 23 and get that $-\sum_{i:\alpha_i \mathbf{v}_i(u) > 0} \alpha_i \mathbf{v}_i(u) \geq -k\sqrt{\Delta \log n}$. By combining the two bounds we get $-\frac{1}{2}\lambda_k^t(1 - \lambda_2)\frac{k}{\Delta n} < -(t+1)\lambda_2^t(\lambda_2 - \lambda_k)k\sqrt{\Delta \log n}$, which is exactly the same condition of the previous case. Thus, for every time $t < T_2 = \Omega(n^{\frac{3}{2}})$, we have that

$$\mathbf{c}^{(t)}(u) - \mathbf{c}^{(t+1)}(u) \leq \frac{1}{2}\lambda_k^t(1 - \lambda_2) \sum_{i=1}^{k-1} \gamma_i \chi_i(u). \quad (13)$$

By combining Equation (11) and Equation (13), we conclude that $|\mathbf{c}^{(t)}(u) - \mathbf{c}^{(t+1)}(u)| \geq \frac{1}{2}\lambda_k^t(1 - \lambda_2) |\mathbf{y}(u)|$.

Now we show that $\text{sgn}(\mathbf{c}^{(t)}(u) - \mathbf{c}^{(t+1)}(u)) = \text{sgn}(\mathbf{y}(u))$. In particular, Equations (7) and (12) imply that $-(t+1)\lambda_2^t(\lambda_2 - \lambda_k) \sum_{i:\alpha_i \mathbf{v}_i(u) < 0} \alpha_i \mathbf{v}_i(u) \leq \frac{1}{2}\lambda_k^t(1 - \lambda_2) |\mathbf{y}(u)|$ and that $(t+1)\lambda_2^t(\lambda_2 - \lambda_k) \sum_{i:\alpha_i \mathbf{v}_i(u) > 0} \alpha_i \mathbf{v}_i(u) \leq \frac{1}{2}\lambda_k^t(1 - \lambda_2) |\mathbf{y}(u)|$. Thus, upper and lower bounds for $\mathbf{c}^{(t)}(u) - \mathbf{c}^{(t+1)}(u)$ in Claim 22, during for every $t < T_2$, have the same sign of \mathbf{y} and consequently $\text{sgn}(\mathbf{c}^{(t)}(u) - \mathbf{c}^{(t+1)}(u)) = \text{sgn}(\mathbf{y}(u))$. ◀

D Upper bound on the error contribution

In this section we upper bound the *error contribution*, i.e., the part of the state vector in the eigenspace of eigenvalues $\lambda_{k+1}, \dots, \lambda_n$ (Lemma 25).

► **Lemma 25** (Upper bound on the error contribution). *Let $\mathbf{e}^{(t)} := \sum_{i=k+1}^n \lambda_i^t \alpha_i \mathbf{v}_i$. For every $u \in V$, it holds that $|\mathbf{e}^{(t)}(u)| \leq \lambda_{k+1}^t \sqrt{\Delta n}$.*

Proof. To bound all components of vector $\mathbf{e}^{(t)}$ we use its ℓ^∞ norm, defined for any vector \mathbf{x} as $\|\mathbf{x}\|_\infty := \sup_i |\mathbf{x}(i)|$. In particular

$$\begin{aligned} \|\mathbf{e}^{(t)}\|_\infty^2 &\leq \|\mathbf{e}^{(t)}\|^2 = \left\| \sum_{i=k+1}^n \lambda_i^t \alpha_i \mathbf{v}_i \right\|^2 = \left\| \sum_{i=k+1}^n \lambda_i^t \beta_i D^{-\frac{1}{2}} \mathbf{w}_i \right\|^2 \\ &\stackrel{(a)}{\leq} \left\| D^{-\frac{1}{2}} \right\|^2 \left\| \sum_{i=k+1}^n \lambda_i^t \beta_i \mathbf{w}_i \right\|^2 \stackrel{(b)}{=} \left\| D^{-\frac{1}{2}} \right\|^2 \sum_{i=k+1}^n \lambda_i^{2t} \beta_i^2 \\ &\leq \left\| D^{-\frac{1}{2}} \right\|^2 \lambda_{k+1}^{2t} \sum_{i=k+1}^n \beta_i^2 \leq \left\| D^{-\frac{1}{2}} \right\|^2 \lambda_{k+1}^{2t} \sum_{i=1}^n \beta_i^2 \\ &= \left\| D^{-\frac{1}{2}} \right\|^2 \lambda_{k+1}^{2t} \left\| D^{\frac{1}{2}} \mathbf{x} \right\|^2 \leq \left\| D^{-\frac{1}{2}} \right\|^2 \lambda_{k+1}^{2t} \left\| D^{\frac{1}{2}} \right\|^2 \|\mathbf{x}\|^2 \\ &\stackrel{(c)}{=} \frac{\max_u \delta(u)}{\min_u \delta(u)} \lambda_{k+1}^{2t} \|\mathbf{x}\|^2 \leq \lambda_{k+1}^{2t} \Delta n, \end{aligned}$$

where in (a) we use Cauchy-Schwarz inequality (Theorem 14) and we apply the definition of spectral norm of an operator, i.e., $\|A\| := \sup_{\mathbf{x}: \|\mathbf{x}\|=1} \|A\mathbf{x}\|$; in (b) we use that the \mathbf{w}_i s are orthonormal; in (c) we use that the spectral norm of a diagonal matrix is equal to its maximum value. Thus, for every $u \in V$ it holds that $|\mathbf{e}^{(t)}(u)| \leq \sqrt{\|\mathbf{e}^{(t)}\|_\infty^2} \leq \lambda_{k+1}^t \sqrt{\Delta n}$. ◀

Blocking Dominating Sets for H -Free Graphs via Edge Contractions

Esther Galby

Department of Informatics, University of Fribourg, Fribourg, Switzerland
esther.galby@unifr.ch

Paloma T. Lima

Department of Informatics, University of Bergen, Bergen, Norway
paloma.lima@uib.no

Bernard Ries

Department of Informatics, University of Fribourg, Fribourg, Switzerland
bernard.ries@unifr.ch

Abstract

In this paper, we consider the following problem: given a connected graph G , can we reduce the domination number of G by one by using only one edge contraction? We show that the problem is NP-hard when restricted to $\{P_6, P_4 + P_2\}$ -free graphs and that it is coNP-hard when restricted to subcubic claw-free graphs and $2P_3$ -free graphs. As a consequence, we are able to establish a complexity dichotomy for the problem on H -free graphs when H is connected.

2012 ACM Subject Classification Mathematics of computing \rightarrow Graph theory

Keywords and phrases domination number, blocker problem, H -free graphs

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.21

1 Introduction

A *blocker problem* asks whether given a graph G , a graph parameter π , a set \mathcal{O} of one or more graph operations and an integer $k \geq 1$, G can be transformed into a graph G' by using at most k operations from \mathcal{O} such that $\pi(G') \leq \pi(G) - d$ for some *threshold* $d \geq 0$. Such a designation follows from the fact that the set of vertices or edges involved can be viewed as "blocking" the parameter π . Identifying such sets may provide information on the structure of the input graph; for instance, if $\pi = \alpha$, $k = d = 1$ and $\mathcal{O} = \{\text{vertex deletion}\}$, the problem is equivalent to testing whether the input graph contains a vertex that is in every maximum independent set (see [18]). Blocker problems have received much attention in the recent literature (see for instance [1, 2, 3, 4, 5, 7, 8, 9, 11, 12, 13, 15, 16, 17, 18, 19]) and have been related to other well-known graph problems such as HADWIGER NUMBER, CLUB CONTRACTION and several graph transversal problems (see for instance [7, 17]). The graph parameters mainly considered in the literature so far include the chromatic number, the independence number, the clique number, the matching number and the vertex cover number while the set \mathcal{O} is always a singleton consisting of a vertex deletion, edge contraction, edge deletion or edge addition. In this paper, we focus on the domination number γ , let \mathcal{O} consist of an edge contraction and set the threshold d to one.

Formally, let $G = (V, E)$ be a graph. The *contraction* of an edge $uv \in E$ removes vertices u and v from G and replaces them by a new vertex that is made adjacent to precisely those vertices which were adjacent to u or v in G (without introducing self-loops nor multiple edges). We say that a graph G can be *k -contracted* into a graph G' , if G can be transformed into G' by a sequence of at most k edge contractions, for an integer $k \geq 1$. The problem we consider is then the following (note that contracting an edge cannot increase the domination number).



© Esther Galby, Paloma T. Lima, and Bernard Ries;
licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 21; pp. 21:1–21:14

Leibniz International Proceedings in Informatics



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

k -EDGE CONTRACTION(γ)

Instance: A connected graph $G = (V, E)$.

Question: Can G be k -contracted into a graph G' such that $\gamma(G') \leq \gamma(G) - 1$?

Reducing the domination number using edge contractions was first considered in [10]. The authors proved that for a connected graph G such that $\gamma(G) \geq 2$, we have $ct_\gamma(G) \leq 3$, where $ct_\gamma(G)$ denotes the minimum number of edge contractions required to transform G into a graph G' such that $\gamma(G') \leq \gamma(G) - 1$ (note that if $\gamma(G) = 1$ then G is a NO-instance for k -EDGE CONTRACTION(γ) independently of the value of k). Thus, if G is a connected graph with $\gamma(G) \geq 2$, then G is always a YES-instance for k -EDGE CONTRACTION(γ) when $k \geq 3$. It was later shown in [9] that k -EDGE CONTRACTION(γ) is coNP-hard for $k \leq 2$ and so, restrictions on the input graph to some special graph classes were considered. In particular, the authors in [9] proved that for $k = 1, 2$, the problem is polynomial-time solvable for P_5 -free graphs while for $k = 1$, it remains NP-hard when restricted to P_9 -free graphs and $\{C_3, \dots, C_\ell\}$ -free graphs, for any $\ell \geq 3$.

In this paper, we continue the systematic study of the computational complexity of 1-EDGE CONTRACTION(γ) initiated in [9]. Ultimately, the aim is to obtain a complete classification for 1-EDGE CONTRACTION(γ) restricted to H -free graphs, for any (not necessarily connected) graph H , as it has been done for other blocker problems (see for instance [8, 18, 19]). As a step towards this end, we prove the following three theorems.

► **Theorem 1.** 1-EDGE CONTRACTION(γ) is NP-hard when restricted to $\{P_6, P_4 + P_2\}$ -free graphs.

► **Theorem 2.** 1-EDGE CONTRACTION(γ) is coNP-hard when restricted to subcubic claw-free graphs.

► **Theorem 3.** 1-EDGE CONTRACTION(γ) is coNP-hard when restricted to $2P_3$ -free graphs.

Note that Theorems 1 and 2 lead to a complexity dichotomy for H -free graphs when H is connected. Indeed, since 1-EDGE CONTRACTION(γ) is NP-hard when restricted to $\{C_3, \dots, C_\ell\}$ -free graphs, for any $\ell \geq 3$, it follows that 1-EDGE CONTRACTION(γ) is NP-hard for H -free graphs when H contains a cycle. If H is a tree with a vertex of degree at least three, we conclude by Theorem 2 that 1-EDGE CONTRACTION(γ) is coNP-hard for H -free graphs; and Theorem 1 shows that if H is a path of length at least 6, then 1-EDGE CONTRACTION(γ) is NP-hard for H -free graphs. Finally, since in [9] 1-EDGE CONTRACTION(γ) is shown to be polynomial-time solvable on $\{P_5 + pK_1\}$ -free graphs for any $p \geq 0$, it follows that 1-EDGE CONTRACTION(γ) is polynomial-time solvable on H -free graphs if $H \subseteq_i P_5$. We therefore obtain the following result.

► **Corollary 4.** Let H be a connected graph. If $H \subseteq_i P_5$ then 1-EDGE CONTRACTION(γ) is polynomial-time solvable on H -free graphs, otherwise it is NP-hard or coNP-hard.

If the graph H is not required to be connected, we know the following. As previously mentioned, 1-EDGE CONTRACTION(γ) is NP-hard (resp. coNP-hard) on H -free graphs when H contains a cycle (resp. an induced claw). Thus, there remains to consider the case where H is a linear forest, that is, a disjoint union of paths. Theorems 1 and 3 show that if H contains either a P_6 , a $P_4 + P_2$ or a $2P_3$ as an induced subgraph, then 1-EDGE CONTRACTION(γ) is NP-hard or coNP-hard on H -free graphs. Since it is known that 1-EDGE CONTRACTION(γ) is polynomial-time solvable on H -free graphs if $H \subseteq_i P_5 + pK_1$, there remains to determine the complexity status of the problem restricted to H -free graphs when $H = P_3 + qP_2 + pK_1$, for $q \geq 1$ and $p \geq 0$.

2 Preliminaries

Throughout the paper, we only consider finite, undirected and connected graphs that have no self-loops or multiple edges. We refer the reader to [6] for any terminology and notation not defined here.

For $n \geq 1$, the path and cycle on n vertices are denoted by P_n and C_n respectively. The *claw* is the complete bipartite graph with one partition of size one and the other of size three.

Let G be a graph, with vertex set $V(G)$ and edge set $E(G)$, and let $u \in V(G)$. We denote by $N_G(u)$, or simply $N(u)$ if it is clear from the context, the set of vertices that are adjacent to u i.e., the *neighbors* of u , and let $N[u] = N(u) \cup \{u\}$. The *degree* of a vertex u , denoted by $d_G(u)$ or simply $d(u)$ if it is clear from the context, is the size of its neighborhood i.e., $d(u) = |N(u)|$. The maximum degree in G is denoted by $\Delta(G)$ and G is *subcubic* if $\Delta(G) \leq 3$.

For a family $\{H_1, \dots, H_p\}$ of graphs, G is said to be $\{H_1, \dots, H_p\}$ -free if G has no induced subgraph isomorphic to a graph in $\{H_1, \dots, H_p\}$; if $p = 1$, we may write H_1 -free instead of $\{H_1\}$ -free. For a subset $V' \subseteq V(G)$, we let $G[V']$ denote the subgraph of G induced by V' , which has vertex set V' and edge set $\{uv \in E(G) \mid u, v \in V'\}$.

A subset $S \subseteq V(G)$ is called an *independent set* or is said to be *independent*, if no two vertices in S are adjacent. A subset $D \subseteq V(G)$ is called a *dominating set*, if every vertex in $V(G) \setminus D$ is adjacent to at least one vertex in D ; the *domination number* $\gamma(G)$ is the number of vertices in a minimum dominating set. For any $v \in D$ and $u \in N[v]$, v is said to *dominate* u (in particular, v dominates itself). We say that D *contains an edge* (or more) if the graph $G[D]$ contains an edge (or more). A dominating set D of G is *efficient* if for every vertex $v \in V$, $|N[v] \cap D| = 1$ that is, v is dominated by exactly one vertex.

In the following, we consider those graphs for which one edge contraction suffices to decrease their domination number by one. A characterization of this class is given in [10].

► **Theorem 5** ([10]). *For a connected graph G , $ct_\gamma(G) = 1$ if and only if there exists a minimum dominating set in G that is not independent.*

In order to prove Theorems 2 and 3, we introduce the following two problems.

ALL EFFICIENT MD

Instance: A connected graph $G = (V, E)$.

Question: Is every minimum dominating set of G efficient?

ALL INDEPENDENT MD

Instance: A connected graph $G = (V, E)$.

Question: Is every minimum dominating set of G independent?

The following is then a straightforward consequence of Theorem 5.

▷ **Fact 1.** Given a graph G , G is a YES-instance for 1-EDGE CONTRACTION(γ) if and only if G is a NO-instance for ALL INDEPENDENT MD.

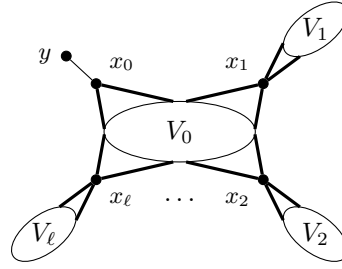
3 The proof of Theorem 1

In this section, we show that 1-EDGE CONTRACTION(γ) is NP-hard when restricted to $\{P_6, P_4 + P_2\}$ -free graphs.

21:4 Blocking Dominating Sets for H -Free Graphs via Edge Contractions

To this end, we give a reduction from DOMINATING SET. Given an instance (G, ℓ) for DOMINATING SET, we construct an instance G' for 1-EDGE CONTRACTION(γ) as follows. We denote by $\{v_1, \dots, v_n\}$ the vertex set of G . The vertex set of the graph G' is given by $V(G') = V_0 \cup \dots \cup V_\ell \cup \{x_0, \dots, x_\ell, y\}$, where each V_i is a copy of the vertex set of G . We denote the vertices of V_i by $v_1^i, v_2^i, \dots, v_n^i$. The adjacencies in G' are then defined as follows:

- $V_0 \cup \{x_0\}$ is a clique;
 - $yx_0 \in E(G')$;
- and for any $1 \leq i \leq \ell$,
- V_i is an independent set;
 - x_i is adjacent to all the vertices in $V_0 \cup V_i$;
 - v_j^i is adjacent to $\{v_a^0 \mid v_a \in N_G[v_j]\}$ for any $1 \leq j \leq n$.



■ **Figure 1** The graph G' (thick lines indicate that the vertex x_i is adjacent to every vertex in V_0 and V_i , for $i = 0, \dots, \ell$).

▷ **Claim 1.** $\gamma(G') = \min\{\gamma(G) + 1, \ell + 1\}$.

Proof. It is clear that $\{x_0, x_1, \dots, x_\ell\}$ is a dominating set of G' ; thus, $\gamma(G') \leq \ell + 1$. If $\gamma(G) \leq \ell$ and $\{v_{i_1}, \dots, v_{i_k}\}$ is a minimum dominating set of G , it is easily seen that $\{v_{i_1}^0, \dots, v_{i_k}^0, x_0\}$ is a dominating set of G' . Thus, $\gamma(G') \leq \gamma(G) + 1$ and so, $\gamma(G') \leq \min\{\gamma(G) + 1, \ell + 1\}$. Now, suppose to the contrary that $\gamma(G') < \min\{\gamma(G) + 1, \ell + 1\}$ and consider a minimum dominating set D' of G' . We first make the following simple observation.

▷ **Observation 1.** For any dominating set D of G' , $D \cap \{y, x_0\} \neq \emptyset$.

Now, since $\gamma(G') < \ell + 1$, there exists $1 \leq i \leq \ell$ such that $x_i \notin D'$ (otherwise, $\{x_1, \dots, x_\ell\} \subset D'$ and combined with Observation 1, D' would be of size at least $\ell + 1$). But then, $D'' = D' \cap V_0$ must dominate every vertex in V_i , and so $|D''| \geq \gamma(G)$. Since $|D''| \leq |D'| - 1$ (recall that $D' \cap \{y, x_0\} \neq \emptyset$), we then have $\gamma(G) \leq |D'| - 1$, a contradiction. Thus, $\gamma(G') = \min\{\gamma(G) + 1, \ell + 1\}$. \triangleleft

We now show that (G, ℓ) is a YES-instance for DOMINATING SET if and only if G' is a YES-instance for 1-EDGE CONTRACTION(γ).

Assume first that $\gamma(G) \leq \ell$. Then $\gamma(G') = \gamma(G) + 1$ by the previous claim, and if $\{v_{i_1}, \dots, v_{i_k}\}$ is a minimum dominating set of G , then $\{v_{i_1}^0, \dots, v_{i_k}^0, x_0\}$ is a minimum dominating set of G' which is not independent. Hence, by Theorem 5, G' is a YES-instance for 1-EDGE CONTRACTION(γ).

Conversely, assume that G' is a YES-instance for 1-EDGE CONTRACTION(γ) i.e., there exists a minimum dominating set D' of G' which is not independent (see Theorem 5). Then, Observation 1 implies that there exists $1 \leq i \leq \ell$ such that $x_i \notin D'$; indeed, if it weren't the case, by Claim 1 we would then have $\gamma(G') = \ell + 1$ and thus, D' would consist of x_1, \dots, x_ℓ

and either y or x_0 . In both cases, D' would be independent, a contradiction. It follows that $D'' = D' \cap V_0$ must dominate every vertex in V_i and thus, $|D''| \geq \gamma(G)$. But $|D''| \leq |D'| - 1$ (recall that $D' \cap \{y, x_0\} \neq \emptyset$) and so by Claim 1, $\gamma(G) \leq |D'| - 1 \leq (\ell + 1) - 1$ that is, (G, ℓ) is a YES-instance for DOMINATING SET.

We next show that G' is a P_6 -free graph. Let P be an induced path of G' . First observe that since V_0 is a clique, $|V(P) \cap V_0| \leq 2$. If $|V(P) \cap V_0| = 0$, since each V_i is independent and the same holds for $\{x_0, \dots, x_\ell\}$, we have that $|V(P)| \leq 3$. We now consider the following two cases.

Case 1. $|V(P) \cap V_0| = 2$. Let $u, v \in V_0$ be the vertices of $V(P) \cap V_0$. Since P is an induced path, u and v appear consecutively in P , that is, $uv \in E(P)$. Furthermore, $V(P) \cap \{x_0, \dots, x_\ell\} = \emptyset$ since u and v are adjacent to all the vertices of $\{x_0, \dots, x_\ell\}$. If u has another neighbor $w \in V_i$ in P , for some $i > 0$, then since $N(w) \subset V_0 \cup \{x_i\}$, w can have no neighbor in P other than u , that is, w is an endpoint of the path. Symmetrically, the same holds for a neighbor of v in P different from u . Hence, we conclude that $|V(P)| \leq 4$.

Case 2. $|V(P) \cap V_0| = 1$. Let $u \in V_0$ be the vertex of $V(P) \cap V_0$. If $V(P) \cap \{x_0, \dots, x_\ell\} = \emptyset$, then it is easy to see that $|V(P)| \leq 3$, since any neighbor of u in the path must belong to $\cup_{1 \leq i \leq \ell} V_i$ and, by the same argument as in Case 1, such a neighbor would have to be an endpoint of the path. If $V(P) \cap \{x_0, \dots, x_\ell\} \neq \emptyset$, let x_i be a vertex that is in P . Since $ux_i \in E(G')$, we necessarily have that $ux_i \in E(P)$. Suppose that x_i has another neighbor w in P . Then $w \in V_i$ since $N(x_i) = V_0 \cup V_i$. By the argument used above, w must then be an endpoint of the path; and since u can have at most two neighbors in $\{x_0, \dots, x_\ell\}$, we conclude that $|V(P)| \leq 5$.

Finally, to see that G' is also a $\{P_4 + P_2\}$ -free graph, it suffices to note that any induced P_4 of G' contains at least one vertex of the clique V_0 . This concludes the proof of Theorem 1.

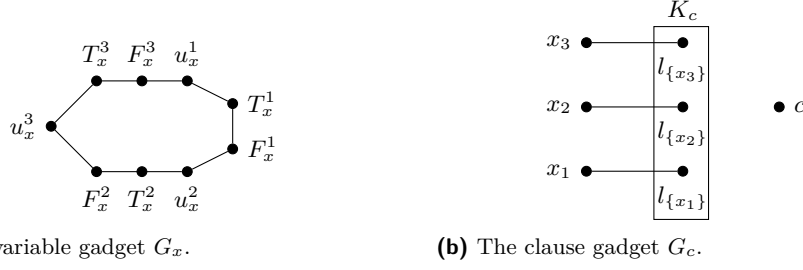
4 The proof of Theorem 2

In this section, we show that 1-EDGE CONTRACTION(γ) is coNP-hard when restricted to subcubic claw-free graphs. To this end, we first prove the following.

► **Lemma 6.** ALL EFFICIENT MD is NP-hard when restricted to subcubic graphs.

Proof. We reduce from POSITIVE EXACTLY 3-BOUNDED 1-IN-3 3-SAT, where each variable appears in exactly three clauses and only positively, each clause contains three positive literals, and we want a truth assignment such that each clause contains exactly one true literal. This problem is shown to be NP-complete in [14]. Given an instance Φ of this problem, with variable set X and clause set C , we construct an equivalent instance of ALL EFFICIENT MD as follows. For any variable $x \in X$, we introduce a copy of G_9 , which we denote by G_x , with three distinguished *true vertices* T_x^1, T_x^2 and T_x^3 , and three distinguished *false vertices* F_x^1, F_x^2 and F_x^3 (see Fig. 2a). For any clause $c \in C$ containing variables x_1, x_2 and x_3 , we introduce the gadget G_c depicted in Fig. 2b which has one distinguished *clause vertex* c and three distinguished *variable vertices* x_1, x_2 and x_3 (note that G_c is not connected). For every $j \in \{1, 2, 3\}$, we then add an edge between x_j and $F_{x_j}^i$ and between c and $T_{x_j}^i$ for some $i \in \{1, 2, 3\}$ so that $F_{x_j}^i$ (resp. $T_{x_j}^i$) is adjacent to exactly one variable vertex (resp. clause vertex). We denote by G_Φ the resulting graph. Note that $\Delta(G_\Phi) = 3$.

▷ **Observation 1.** For any dominating set D of G_Φ , $|D \cap V(G_x)| \geq 3$ for any $x \in X$ and $|D \cap V(G_c)| \geq 1$ for any $c \in C$. In particular, $\gamma(G_\Phi) \geq 3|X| + |C|$.



■ **Figure 2** Construction of the graph G_Φ (the rectangle indicates that the corresponding set of vertices induces a clique).

Indeed, for any $x \in X$, since u_x^1, u_x^2 and u_x^3 must be dominated and their neighborhoods are pairwise disjoint and contained in G_x , it follows that $|D \cap V(G_x)| \geq 3$. For any $c \in C$, since the vertices of K_c must be dominated and their neighborhoods are contained in G_c , $|D \cap V(G_c)| \geq 1$. \square

▷ **Observation 2.** For any $x \in X$, if D is a minimum dominating set of G_x then either $D = \{u_x^1, u_x^2, u_x^3\}$, $D = \{T_x^1, T_x^2, T_x^3\}$ or $D = \{F_x^1, F_x^2, F_x^3\}$.

▷ **Claim 1.** Φ is satisfiable if and only if $\gamma(G_\Phi) = 3|X| + |C|$.

Proof. Assume that Φ is satisfiable and consider a truth assignment satisfying Φ . We construct a dominating set D of G_Φ as follows. For any variable $x \in X$, if x is true, add T_x^1, T_x^2 and T_x^3 to D ; otherwise, add F_x^1, F_x^2 and F_x^3 to D . For any clause $c \in C$ containing variables x_1, x_2 and x_3 , exactly one variable is true, say x_1 without loss of generality; we then add $l_{\{x_1\}}$ to D . Clearly, D is dominating and we conclude by Observation 1 that $\gamma(G_\Phi) = 3|X| + |C|$.

Conversely, assume that $\gamma(G_\Phi) = 3|X| + |C|$ and consider a minimum dominating set D of G_Φ . Then by Observation 1, $|D \cap V(G_x)| = 3$ for any $x \in X$ and $|D \cap V(G_c)| = 1$ for any $c \in C$. Now, for a clause $c \in C$ containing variables x_1, x_2 and x_3 , if $D \cap \{c, x_1, x_2, x_3\} \neq \emptyset$ then $D \cap V(K_c) = \emptyset$ and so, at least two vertices from K_c are not dominated; thus, $D \cap \{c, x_1, x_2, x_3\} = \emptyset$. It follows that for any $x \in X$, $D \cap V(G_x)$ is a minimum dominating set of G_x which by Observation 2 implies either $\{T_x^1, T_x^2, T_x^3\} \subset D$ or $D \cap \{T_x^1, T_x^2, T_x^3\} = \emptyset$; and we conclude similarly that either $\{F_x^1, F_x^2, F_x^3\} \subset D$ or $D \cap \{F_x^1, F_x^2, F_x^3\} = \emptyset$. Now given a clause $c \in C$ containing variables x_1, x_2 and x_3 , since $D \cap \{c, x_1, x_2, x_3\} = \emptyset$, at least one true vertex adjacent to the clause vertex c must belong to D , say $T_{x_1}^i$ for some $i \in \{1, 2, 3\}$ without loss of generality. It then follows that $\{T_{x_1}^1, T_{x_1}^2, T_{x_1}^3\} \subset D$ and $D \cap \{F_{x_1}^1, F_{x_1}^2, F_{x_1}^3\} = \emptyset$ which implies that $l_{\{x_1\}} \in D$ (either x_1 or a vertex from K_c would otherwise not be dominated). But then, since x_j for $j \neq 1$, must be dominated, it follows that $\{F_{x_j}^1, F_{x_j}^2, F_{x_j}^3\} \subset D$. We thus construct a truth assignment satisfying Φ as follows: for any variable $x \in X$, if $\{T_x^1, T_x^2, T_x^3\} \subset D$, set x to true, otherwise set x to false. \square

▷ **Claim 2.** $\gamma(G_\Phi) = 3|X| + |C|$ if and only if every minimum dominating set of G_Φ is efficient.

Proof. Assume that $\gamma(G_\Phi) = 3|X| + |C|$ and consider a minimum dominating set D of G_Φ . Then by Observation 1, $|D \cap V(G_x)| = 3$ for any $x \in X$ and $|D \cap V(G_c)| = 1$ for any $c \in C$. As shown previously, it follows that for any clause $c \in C$ containing variables x_1, x_2 and x_3 , $D \cap \{c, x_1, x_2, x_3\} = \emptyset$; and for any $x \in X$, either $\{T_x^1, T_x^2, T_x^3\} \subset D$ or $D \cap \{T_x^1, T_x^2, T_x^3\} = \emptyset$ (we conclude similarly with $\{F_x^1, F_x^2, F_x^3\}$ and $\{u_x^1, u_x^2, u_x^3\}$). Thus, for any $x \in X$, every

vertex in G_x is dominated by exactly one vertex. Now given a clause $c \in C$ containing variables x_1, x_2 and x_3 , since the clause vertex c does not belong to D , there exists at least one true vertex adjacent to c which belongs to D . Suppose to the contrary that c has strictly more than one neighbor in D , say $T_{x_1}^i$ and $T_{x_2}^j$ without loss of generality. Then, $\{T_{x_k}^1, T_{x_k}^2, T_{x_k}^3\} \subset D$ for $k = 1, 2$ which implies that $D \cap \{F_{x_1}^1, F_{x_1}^2, F_{x_1}^3, F_{x_2}^1, F_{x_2}^2, F_{x_2}^3\} = \emptyset$ as $|D \cap V(G_{x_k})| = 3$ for $k = 1, 2$. It follows that the variable vertices x_1 and x_2 must be dominated by some vertices in G_c ; but $|D \cap V(G_c)| = 1$ and $N[x_1] \cap N[x_2] = \emptyset$ and so, either x_1 or x_2 is not dominated. Thus, c has exactly one neighbor in D , say $T_{x_1}^i$ without loss of generality. Then, necessarily $D \cap V(G_c) = \{u_{\{x_1\}}\}$ for otherwise either x_1 or some vertex in K_c would not be dominated. But then, it is clear that every vertex in G_c is dominated by exactly one vertex; thus, D is efficient.

Conversely, assume that every minimum dominating set of G_Φ is efficient and consider a minimum dominating set D of G_Φ . If for some $x \in X$, $|D \cap V(G_x)| \geq 4$, then clearly at least one vertex in G_x is dominated by two vertices in $D \cap V(G_x)$. Thus, $|D \cap V(G_x)| \leq 3$ for any $x \in X$ and we conclude by Observation 1 that in fact, equality holds. The next observation immediately follows from the fact that D is efficient.

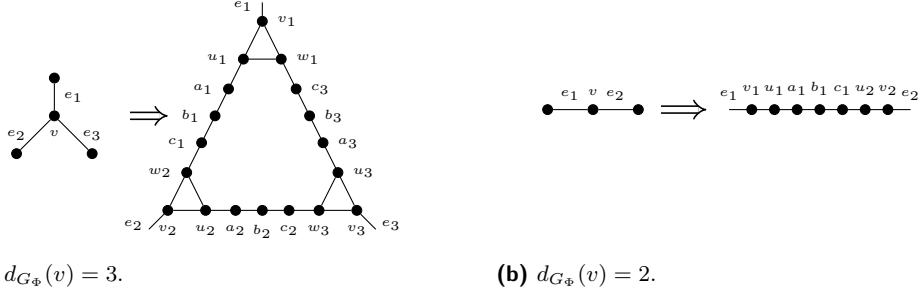
▷ **Observation 3.** For any $x \in X$, if $|D \cap V(G_x)| = 3$ then either $\{u_x^1, u_x^2, u_x^3\} \subset D$, $\{T_x^1, T_x^2, T_x^3\} \subset D$ or $\{F_x^1, F_x^2, F_x^3\} \subset D$.

Now, consider a clause $c \in C$ containing variables x_1, x_2 and x_3 and suppose without loss of generality that $T_{x_1}^1$ is adjacent to c (note that then the variable vertex x_1 is adjacent to $F_{x_1}^1$). If the clause vertex c belongs to D then, since D is efficient, $T_{x_1}^1 \notin D$ and $u_{x_1}^1, F_{x_1}^1 \notin D$ ($T_{x_1}^1$ would otherwise be dominated by at least two vertices) which contradicts Observation 3. Thus, no clause vertex belongs to D . Similarly, suppose that there exists $i \in \{1, 2, 3\}$ such that $x_i \in D$, say $x_1 \in D$ without loss of generality. Then, since D is efficient, $F_{x_1}^1 \notin D$ and $T_{x_1}^1, u_{x_1}^2 \notin D$ ($F_{x_1}^1$ would otherwise be dominated by at least two vertices) which again contradicts Observation 3. Thus, no variable vertex belongs to D . Finally, since D is efficient, $|D \cap V(K_c)| \leq 1$ and so, $|D \cap V(G_c)| = 1$ by Observation 1. ◀

Now by combining Claims 1 and 2, we obtain that Φ is satisfiable if and only if every minimum dominating set of G_Φ is efficient, that is, G_Φ is a YES-instance for ALL EFFICIENT MD. ◀

▶ **Theorem 7.** ALL INDEPENDENT MD is NP-hard when restricted to subcubic claw-free graphs.

Proof. We give a reduction from POSITIVE EXACTLY 3-BOUNDED 1-IN-3 3-SAT, where each variable appears in exactly three clauses and only positively, each clause contains three positive literals, and we want a truth assignment such that each clause contains exactly one true literal. This problem is shown to be NP-complete in [14]. Given an instance Φ of this problem, with variable set X and clause set C , we construct an equivalent instance of ALL INDEPENDENT MD as follows. Consider the graph $G_\Phi = (V, E)$ constructed in the proof of Lemma 6 and let $V_i = \{v \in V : d_{G_\Phi}(v) = i\}$ for $i = 2, 3$ (note that no vertex in G_Φ has degree one). Then, for any $v \in V_3$, we replace the vertex v by the gadget G_v depicted in Fig. 3a; and for any $v \in V_2$, we replace the vertex v by the gadget G_v depicted in Fig. 3b. We denote by G'_Φ the resulting graph. Note that G'_Φ is claw-free and $\Delta(G'_\Phi) = 3$ (also note that no vertex in G'_Φ has degree one). It is shown in the proof of Lemma 6 that Φ is satisfiable if and only if G_Φ is a YES-instance for ALL EFFICIENT MD; we here show that G_Φ is a YES-instance for ALL EFFICIENT MD if and only if G'_Φ is a YES-instance for ALL INDEPENDENT MD. To this end, we first prove the following.



■ **Figure 3** The gadget G_v .

▷ **Claim 3.** $\gamma(G'_\Phi) = \gamma(G_\Phi) + 5|V_3| + 2|V_2|$.

Proof. Let D be a minimum dominating set of G_Φ . We construct a dominating set D' of G'_Φ as follows. For any $v \in D$, if $v \in V_3$, add v_1, v_2, v_3, b_1, b_2 and b_3 to D' ; otherwise, add v_1, v_2 and b_1 to D' . For any $v \in V \setminus D$, let $u \in D$ be a neighbor of v , say $e_1 = uv$ without loss of generality. Then, if $v \in V_3$, add a_1, c_3, w_2, u_3 and b_2 to D' ; otherwise, add a_1 and u_2 to D' . Clearly, D' is dominating and $|D'| = \gamma(G_\Phi) + 5|V_3| + 2|V_2| \geq \gamma(G'_\Phi)$.

▷ **Observation 4.** For any dominating set D' of G'_Φ , the following holds.

- (i) For any $v \in V_2$, $|D' \cap V(G_v)| \geq 2$. Moreover, if equality holds then $D' \cap \{v_1, v_2\} = \emptyset$ and there exists $j \in \{1, 2\}$ such that $u_j \notin D'$.
- (ii) For any $v \in V_3$, $|D' \cap V(G_v)| \geq 5$. Moreover, if equality holds then $D' \cap \{v_1, v_2, v_3\} = \emptyset$ and there exists $j \in \{1, 2, 3\}$ such that $D' \cap \{u_j, v_j, w_j\} = \emptyset$.

(i) Clearly, $D' \cap \{v_1, u_1, a_1\} \neq \emptyset$ and $D' \cap \{c_1, u_2, v_2\} \neq \emptyset$ as u_1 and u_2 must be dominated. Thus, $|D' \cap V(G_v)| \geq 2$. Now, suppose that $D' \cap \{v_1, v_2\} \neq \emptyset$ say $v_1 \in D'$ without loss of generality. Then $D' \cap \{u_1, a_1, b_1\} \neq \emptyset$ as a_1 must be dominated which implies that $|D' \cap V(G_v)| \geq 3$ (recall that $D' \cap \{c_1, u_2, v_2\} \neq \emptyset$). Similarly, if both u_1 and u_2 belong to D' , then $|D' \cap V(G_v)| \geq 3$ as $D' \cap \{a_1, b_1, c_1\} \neq \emptyset$ (b_1 would otherwise not be dominated).

(ii) Clearly, for any $i \in \{1, 2, 3\}$, $D' \cap \{a_i, b_i, c_i\} \neq \emptyset$ as b_i must be dominated. Now, if there exists $j \in \{1, 2, 3\}$ such that $D' \cap \{u_j, v_j, w_j\} = \emptyset$, say $j = 1$ without loss of generality, then $a_1, c_3 \in D'$ (one of u_1 and w_1 would otherwise not be dominated). But then, $D' \cap \{b_1, c_1, w_2\} \neq \emptyset$ as c_1 must be dominated, and $D' \cap \{a_3, b_3, u_3\} \neq \emptyset$ as a_3 must be dominated; and so, $|D' \cap V(G_v)| \geq 5$ (recall that $D' \cap \{a_2, b_2, c_2\} \neq \emptyset$). Otherwise, for any $j \in \{1, 2, 3\}$, $D' \cap \{u_j, v_j, w_j\} \neq \emptyset$ which implies that $|D' \cap V(G_v)| \geq 6$.

Now suppose that $D' \cap \{v_1, v_2, v_3\} \neq \emptyset$, say $v_1 \in D'$ without loss of generality. If there exists $j \neq 1$ such that $D' \cap \{u_j, v_j, w_j\} = \emptyset$, say $j = 2$ without loss of generality, then $c_1, a_2 \in D'$ (one of u_2 and w_2 would otherwise not be dominated). But then, $D' \cap \{a_1, b_1, u_1\} \neq \emptyset$ as a_1 should be dominated, and $D' \cap \{b_2, c_2, w_3\} \neq \emptyset$ as c_2 must be dominated. Since $D' \cap \{a_3, b_3, c_3\} \neq \emptyset$, it then follows that $|D' \cap V(G_v)| \geq 6$. Otherwise, $D' \cap \{u_j, v_j, w_j\} \neq \emptyset$ for any $j \in \{1, 2, 3\}$ and so, $|D' \cap V(G_v)| \geq 6$ (recall that $D' \cap \{a_i, b_i, c_i\} \neq \emptyset$ for any $i \in \{1, 2, 3\}$). \lrcorner

▷ **Observation 5.** If D' is a minimum dominating set of G'_Φ , then $|D' \cap V(G_v)| \leq 3$ for any $v \in V_2$ and $|D' \cap V(G_v)| \leq 6$ for any $v \in V_3$.

Indeed, if $v \in V_2$ then $\{v_1, b_1, v_2\}$ is a dominating set of $V(G_v)$; and if $v \in V_3$, then $\{v_1, v_2, v_3, b_1, b_2, b_3\}$ is a dominating set of $V(G_v)$. \lrcorner

Now, consider a minimum dominating set D' of G'_Φ and let $D_3 = \{v \in V_3 : |D' \cap V(G_v)| = 6\}$ and $D_2 = \{v \in V_2 : |D' \cap V(G_v)| = 3\}$. We claim that $D = D_3 \cup D_2$ is a dominating set of G_Φ . Indeed, consider a vertex $v \in V \setminus D$. We distinguish two cases depending on whether $v \in V_2$ or $v \in V_3$.

Case 1. $v \in V_2$. Then $|D' \cap V(G_v)| = 2$ by construction, which by Observation 4(i) implies that there exists $j \in \{1, 2\}$ such that $D' \cap \{v_j, u_j\} = \emptyset$, say $j = 1$ without loss of generality. Since v_1 must be dominated, v_1 must then have a neighbor x_i belonging to D' , for some vertex x adjacent to v in G_Φ . But then, it follows from Observation 4 that $|D' \cap V(G_x)| > 2$ if $x \in V_2$, and $|D' \cap V(G_x)| > 5$ if $x \in V_3$ (indeed, $x_i \in D'$); thus, $x \in D$.

Case 2. $v \in V_3$. Then $|D' \cap V(G_v)| = 5$ by construction, which by Observation 4(ii) implies that there exists $j \in \{1, 2, 3\}$ such that $D' \cap \{u_j, v_j, w_j\} = \emptyset$, say $j = 1$ without loss of generality. Since v_1 must be dominated, v_1 must then have a neighbor x_i belonging to D' , for some vertex x adjacent to v in G_Φ . But then, it follows from Observation 4 that $|D' \cap V(G_x)| > 2$ if $x \in V_2$, and $|D' \cap V(G_x)| > 5$ if $x \in V_3$ (indeed, $x_i \in D'$); thus, $x \in D$.

Hence, D is a dominating set of G_Φ . Moreover, it follows from Observations 4 and 5 that $|D'| = 6|D_3| + 5|V_3 \setminus D_3| + 3|D_2| + 2|V_2 \setminus D_2| = |D| + 5|V_3| + 2|V_2|$. Thus, $\gamma(G'_\Phi) = |D'| \geq \gamma(G_\Phi) + 5|V_3| + 2|V_2|$ and so, $\gamma(G'_\Phi) = \gamma(G_\Phi) + 5|V_3| + 2|V_2|$. Finally note that this implies that the constructed dominated set D is in fact minimum. \triangleleft

We next show that G_Φ is a YES-instance for ALL EFFICIENT MD if and only if G'_Φ is a YES-instance for ALL INDEPENDENT MD. Since Φ is satisfiable if and only if G_Φ is a YES-instance for ALL EFFICIENT MD, as shown in the proof of Lemma 6, this would conclude the proof.

Assume first that G_Φ is a YES-instance for ALL EFFICIENT MD and suppose to the contrary that G'_Φ is a NO-instance for ALL INDEPENDENT MD that is, G'_Φ has a minimum dominating set D' which is not independent. Denote by D the minimum dominating set of G_Φ constructed from D' according to the proof of Claim 3. Let us show that D is not efficient. Consider two adjacent vertices $a, b \in D'$. If a and b belong to gadgets G_x and G_v respectively, for two adjacent vertices x and v in G_Φ , that is, a is of the form x_i and b is of the form v_j , then by Observation 4 $x, v \in D$ and so, D is not efficient. Thus, it must be that a and b both belong the same gadget G_v , for some $v \in V_2 \cup V_3$. We distinguish cases depending on whether $v \in V_2$ or $v \in V_3$.

Case 1. $v \in V_2$. Suppose that $|D' \cap V(G_v)| = 2$. Then by Observation 4(i), $D' \cap \{v_1, v_2\} = \emptyset$ and there exists $j \in \{1, 2\}$ such that $u_j \notin D'$, say $u_1 \notin D'$ without loss of generality. Then, necessarily $a_1 \in D'$ (u_1 would otherwise not be dominated) and so, $b_1 \in D'$ as $D' \cap V(G_v)$ contains an edge and $|D' \cap V(G_v)| = 2$ by assumption; but then, u_2 is not dominated. Thus, $|D' \cap V(G_v)| \geq 3$ and we conclude by Observation 5 that in fact, equality holds. Note that consequently, $v \in D$. We claim that then, $|D' \cap \{v_1, v_2\}| \leq 1$. Indeed, if both v_1 and v_2 belong to D' , then $b_1 \in D'$ (since $|D' \cap V(G_v)| = 3$, D' would otherwise not be dominating) which contradicts that fact that $D' \cap V(G_v)$ contains an edge. Thus, $|D' \cap \{v_1, v_2\}| \leq 1$ and we may assume without loss of generality that $v_2 \notin D'$. Let $x_i \neq u_2$ be the other neighbor of v_2 in G'_Φ , where x is a neighbor of v in G_Φ . Suppose first that $x \in V_2$. Then, $|D' \cap V(G_x)| = 2$ for otherwise x would belong to D and so, D would contain the edge vx . It then follows from Observation 4(i) that there exists $j \in \{1, 2\}$ such that $D' \cap \{x_j, y_j\} = \emptyset$, where y_j is the neighbor of x_j in $V(G_x)$.

We claim that $j \neq i$; indeed, if $j = i$, since $v_2, x_i, y_i \notin D'$, x_i would not be dominated. But then, x_j must have a neighbor $t_k \neq y_j$, for some vertex t adjacent to x in G_Φ , which belongs to D' ; it then follows from Observation 4 and the construction of D that $t \in D$ and so, x has two neighbors in D , namely v and t , a contradiction.

Second, suppose that $x \in V_3$. Then, $|D' \cap V(G_x)| = 5$ for otherwise x would belong to D and so, D would contain the edge vx . It then follows from Observation 4(ii) that there exists $j \in \{1, 2, 3\}$ such that $D' \cap \{x_j, y_j, z_j\} = \emptyset$, where y_j and z_j are the two neighbors of x_j in $V(G_x)$. We claim that $j \neq i$; indeed, if $j = i$, since $v_2, x_i, y_i, z_i \notin D'$, x_i would not be dominated. But then, x_j must have a neighbor $t_k \neq y_j, z_j$, for some vertex t adjacent to x in G_Φ , which belongs to D' ; it then follows from Observation 4 and the construction of D that $t \in D$ and so, x has two neighbors in D , namely v and t , a contradiction.

Case 2. $v \in V_3$. Suppose that $|D' \cap V(G_v)| = 5$. Then, by Observation 4(ii), $D' \cap \{v_1, v_2, v_3\} = \emptyset$ and there exists $j \in \{1, 2, 3\}$ such that $D' \cap \{u_j, v_j, w_j\} = \emptyset$, say $j = 1$ without loss of generality. Then, $a_1, c_3 \in D'$ (one of u_1 and w_1 would otherwise not be dominated), $D' \cap \{c_1, w_2, u_2\} \neq \emptyset$ (w_2 would otherwise not be dominated), $D' \cap \{a_3, u_3, w_3\} \neq \emptyset$ (u_3 would otherwise not be dominated) and $D' \cap \{a_2, b_2, c_2\} \neq \emptyset$ (b_2 would otherwise not be dominated); in particular, $b_1, b_3 \notin D'$ as $|D' \cap V(G_v)| = 5$ by assumption. Since $D' \cap V(G_v)$ contains an edge, it follows that either $u_2, a_2 \in D'$ or $c_2, w_3 \in D'$; but then, either c_1 or a_3 is not dominated, a contradiction. Thus, $|D' \cap V(G_v)| \geq 6$ and we conclude by Observation 5 that in fact, equality holds. Note that consequently, $v \in D$. It follows that $\{v_1, v_2, v_3\} \not\subset D'$ for otherwise $D' \cap V(G_v) = \{v_1, v_2, v_3, b_1, b_2, b_3\}$ and so, $D' \cap V(G_v)$ contains no edge. Thus, we may assume without loss of generality that $v_1 \notin D'$. Denoting by $x_i \neq u_1, w_1$ the third neighbor of v_1 , where x is a neighbor of v in G_Φ , we then proceed as in the previous case to conclude that x has two neighbors in D .

Thus, D is not efficient, which contradicts the fact that G_Φ is a YES-instance for ALL EFFICIENT MD. Hence, every minimum dominating set of G'_Φ is independent i.e., G'_Φ is a YES-instance for ALL INDEPENDENT MD.

Conversely, assume that G'_Φ is a YES-instance for ALL INDEPENDENT MD and suppose to the contrary that G_Φ is a NO-instance for ALL EFFICIENT MD that is, G_Φ has a minimum dominating set D which is not efficient. Let us show that D either contains an edge or can be transformed into a minimum dominating set of G_Φ containing an edge. Since any minimum dominating set of G'_Φ constructed according to the proof of Claim 3 from a minimum dominating set of G_Φ containing an edge, also contains an edge, this would lead to a contradiction and thus conclude the proof.

Suppose that D contains no edge. Since D is not efficient, there must then exist a vertex $v \in V \setminus D$ such that v has two neighbors in D . We distinguish cases depending on which type of vertex v is.

Case 1. v is a variable vertex. Suppose that $v = x_1$ in some clause gadget G_c , where $c \in C$ contains variables x_1, x_2 and x_3 , and assume without loss of generality that x_1 is adjacent to $F_{x_1}^1$. By assumption, $F_{x_1}^1, l_{\{x_1\}} \in D$ which implies that $D \cap \{l_{\{x_2\}}, l_{\{x_3\}}, T_{x_1}^1, u_{x_1}^2\} = \emptyset$ (D would otherwise contain an edge). We may then assume that $F_{x_2}^i$ and $F_{x_3}^j$, where $F_{x_2}^i x_2, F_{x_3}^j x_3 \in E(G_\Phi)$, belong to D ; indeed, since x_2 (resp. x_3) must be dominated, $D \cap \{F_{x_2}^i, x_2\} \neq \emptyset$ (resp. $D \cap \{F_{x_3}^j, x_3\} \neq \emptyset$) and since $l_{\{x_1\}} \in D$, $(D \setminus \{x_2\}) \cup \{F_{x_2}^i\}$ (resp. $(D \setminus \{x_3\}) \cup \{F_{x_3}^j\}$) remains dominating. We may then assume that $T_{x_2}^i, T_{x_3}^j \notin D$ for otherwise D would contain an edge. It follows that $c \in D$ (c would otherwise not be dominated); but then, it suffices to consider $(D \setminus \{c\}) \cup \{T_{x_1}^1\}$ to obtain a minimum dominating set of G_Φ containing an edge.

Case 2. $v = u_x^i$ for some variable $x \in X$ and $i \in \{1, 2, 3\}$. Assume without loss of generality that $i = 1$. Then $T_x^1, F_x^3 \in D$ by assumption, which implies that $F_x^1, T_x^3 \notin D$ (D would otherwise contain an edge). But then, $|D \cap \{u_x^2, F_x^2, T_x^2, u_x^3\}| \geq 2$ as u_x^2 and u_x^3 must be dominated; and so, $(D \setminus \{u_x^3, F_x^2, T_x^2, u_x^2\}) \cup \{F_x^2, T_x^2\}$ is a dominating set of G_Φ of size at most that of D which contains an edge.

Case 3. v is a clause vertex. Suppose that $v = c$ for some clause $c \in C$ containing variables x_1, x_2 and x_3 , and assume without loss of generality that c is adjacent to $T_{x_i}^1$ for any $i \in \{1, 2, 3\}$. By assumption c has two neighbors in D , say $T_{x_1}^1$ and $T_{x_2}^1$ without loss of generality. Since D contains no edge, it follows that $F_{x_1}^1, F_{x_2}^1 \notin D$; but then, $|D \cap \{x_1, x_2, l_{\{x_1\}}, l_{\{x_2\}}\}| \geq 2$ (one of x_1 and x_2 would otherwise not be dominated) and so, $(D \setminus \{x_1, x_2, l_{\{x_1\}}, l_{\{x_2\}}\}) \cup \{l_{\{x_1\}}, l_{\{x_2\}}\}$ is a dominating set of G_Φ of size at most that of D which contains an edge.

Case 4. $v \in V(K_c)$ for some clause $c \in C$. Denote by x_1, x_2 and x_3 the variables contained in c and assume without loss of generality that $v = l_{\{x_1\}}$. Since $l_{\{x_1\}}$ has two neighbors in D and D contains no edge, necessarily $x_1 \in D$. Now assume without loss of generality that x_1 is adjacent to $F_{x_1}^1$ (note that by construction, c is then adjacent to $T_{x_1}^1$). Then, $F_{x_1}^1 \notin D$ (D would otherwise contain an edge) and $T_{x_1}^1, u_{x_1}^2 \notin D$ for otherwise $(D \setminus \{x_1\}) \cup \{F_{x_1}^1\}$ would be a minimum dominating set of G_Φ containing an edge (recall that by assumption, $D \cap V(K_c) \neq \emptyset$). It follows that $T_{x_1}^2 \in D$ ($u_{x_1}^2$ would otherwise not be dominated) and so, $F_{x_1}^2 \notin D$ as D contains no edge. It follows that $|D \cap \{u_{x_1}^1, F_{x_1}^3, T_{x_1}^3, u_{x_1}^3\}| \geq 2$ as $u_{x_1}^1$ and $u_{x_1}^3$ must be dominated. Now if c belongs to D , then $(D \setminus \{u_{x_1}^1, F_{x_1}^3, T_{x_1}^3, u_{x_1}^3\}) \cup \{F_{x_1}^3, T_{x_1}^3\}$ is a dominating set of G_Φ of size at most that of D which contains an edge. Thus, we may assume that $c \notin D$ which implies that $u_{x_1}^1 \in D$ ($T_{x_1}^1$ would otherwise not be dominated) and that there exists $j \in \{2, 3\}$ such that $T_{x_j}^i \in D$ with $cT_{x_j}^i \in E(G_\Phi)$ (c would otherwise not be dominated). Now, since $u_{x_1}^3$ must be dominated and $F_{x_1}^2 \notin D$, it follows that $D \cap \{u_{x_1}^3, T_{x_1}^3\} \neq \emptyset$ and we may assume that in fact $T_{x_1}^3 \in D$ (recall that $T_{x_1}^2 \in D$ and so, $F_{x_1}^2$ is dominated). But then, by considering the minimum dominating set $(D \setminus \{u_{x_1}^1\}) \cup \{T_{x_1}^1\}$, we fall back into Case 3 as c is then dominated by both $T_{x_1}^1$ and $T_{x_j}^i$.

Case 5. v is a true vertex. Assume without loss of generality that $v = T_x^1$ for some variable $x \in X$. Suppose first that $u_x^1 \in D$. Then since D contains no edge, $F_x^3 \notin D$; furthermore, denoting by $t \neq u_x^1, T_x^3$ the variable vertex adjacent to F_x^3 , we also have $t \notin D$ for otherwise $(D \setminus \{u_x^1\}) \cup \{F_x^3\}$ would be a minimum dominating set containing an edge (recall that T_x^1 has two neighbors in D by assumption). But then, since t must be dominated, it follows that the second neighbor of t must belong to D ; and so, by considering the minimum dominating set $(D \setminus \{u_x^1\}) \cup \{F_x^3\}$, we fall back into Case 1 as the variable vertex t is then dominated by two vertices. Thus, we may assume that $u_x^1 \notin D$ which implies that $F_x^1, c \in D$, where c is the clause vertex adjacent to T_x^1 . Now, denote by $x_1 = x, x_2$ and x_3 the variables contained in c (note that by construction, x_1 is then adjacent to $F_{x_1}^1$). Then, $x_1 \notin D$ (D would otherwise contain the edge $F_{x_1}^1 x_1$) and we may assume that $l_{\{x_1\}} \notin D$ (we otherwise fall back into Case 1 as x_1 would then have two neighbors in D). It follows that $D \cap V(K_c) \neq \emptyset$ ($l_{\{x_1\}}$ would otherwise not be dominated) and since D contains no edge, in fact $|D \cap V(K_c)| = 1$, say $l_{\{x_2\}} \in D$ without loss of generality. Then, $x_2 \notin D$ as D contains no edge and we may assume that $F_{x_2}^j \notin D$, where $F_{x_2}^j$ is the false vertex adjacent to x_2 , for otherwise we fall back into Case 1. In the following, we assume without loss of generality that $j = 1$, that is, x_2 is adjacent to $F_{x_2}^1$ (note that by construction, c is then adjacent to $T_{x_2}^1$). Now, since the clause vertex c belongs to D by assumption, it follows that $T_{x_2}^1 \notin D$ (D would otherwise contain the edge $cT_{x_2}^1$); and as shown previously, we may assume that $u_{x_2}^1 \notin D$ (indeed, $T_{x_2}^1$ would otherwise have two neighbors in D , namely c and $u_{x_2}^1$, but this case has already been dealt with). Then,

21:12 Blocking Dominating Sets for H -Free Graphs via Edge Contractions

since $u_{x_2}^1$ and $F_{x_2}^1$ must be dominated, necessarily $F_{x_2}^3$ and $u_{x_2}^2$ belong to D (recall that $D \cap \{x_2, F_{x_2}^1, T_{x_2}^1, u_{x_2}^1\} = \emptyset$) which implies that $T_{x_2}^3, T_{x_2}^2 \notin D$ (D would otherwise contain an edge). Now since $u_{x_2}^3$ must be dominated, $D \cap \{u_{x_2}^3, F_{x_2}^2\} \neq \emptyset$ and we may assume without loss of generality that in fact, $F_{x_2}^2 \in D$. But then, by considering the minimum dominating set $(D \setminus \{u_{x_2}^2\}) \cup \{F_{x_2}^1\}$, we fall back into Case 1 as x_2 is then dominated by two vertices.

Case 6. v is a false vertex. Assume without loss of generality that $v = F_{x_1}^1$ for some variable $x_1 \in X$ and let $c \in C$ be the clause whose corresponding clause vertex is adjacent to $T_{x_1}^1$. Denote by x_2 and x_3 the two other variables contained in c . Suppose first that $x_1 \in D$. Then, we may assume that $D \cap V(K_c) = \emptyset$ for otherwise either D contains an edge (if $l_{\{x_1\}} \in D$) or we fall back into Case 4 ($l_{\{x_1\}}$ would indeed have two neighbors in D). Since every vertex of K_c must be dominated, it then follows that $x_2, x_3 \in D$; but then, by considering the minimum dominating set $(D \setminus \{x_1\}) \cup \{l_{\{x_1\}}\}$ (recall that $F_{x_1}^1$ has two neighbors in D by assumption), we fall back into Case 4 as $l_{\{x_2\}}$ is then dominated by two vertices. Thus, we may assume that $x_1 \notin D$ which implies that $T_{x_1}^1, u_{x_1}^2 \in D$ and $T_{x_1}^2, u_{x_1}^1 \notin D$ as D contains no edge. Now, denote by c' the clause vertex adjacent to $T_{x_1}^2$. Then, we may assume that $c' \notin D$ for otherwise we fall back into Case 5 ($T_{x_1}^2$ would indeed have two neighbors in D); but then, there must exist a true vertex, different from $T_{x_1}^2$, adjacent to c' and belonging to D (c' would otherwise not be dominated) and by considering the minimum dominating set $(D \setminus \{u_{x_1}^2\}) \cup \{T_{x_1}^2\}$, we then fall back into Case 3 (c' would indeed be dominated by two vertices).

Consequently, G_Φ has a minimum dominating set which is not independent which implies that G'_Φ also has a minimum dominating set which is not independent, a contradiction which concludes the proof. \blacktriangleleft

Theorem 2 now easily follows from Fact 1 and Theorem 7.

5 The proof of Theorem 3

In this section, we show that 1-EDGE CONTRACTION(γ) is coNP-hard when restricted to $2P_3$ -free graphs. To this end, we prove the following.

► **Theorem 8.** ALL INDEPENDENT MD is NP-hard when restricted to $2P_3$ -free graphs.

Proof. We reduce from 3-SAT: given an instance Φ of this problem, with variable set X and clause set C , we construct an equivalent instance of ALL INDEPENDENT MD as follows. For any variable $x \in X$, we introduce a copy of C_3 , which we denote by G_x , with one distinguished *positive literal vertex* x and one distinguished *negative literal vertex* \bar{x} ; in the following, we denote by u_x the third vertex in G_x . For any clause $c \in C$, we introduce a *clause vertex* c ; we then add an edge between c and the (positive or negative) literal vertices whose corresponding literal occurs in c . Finally, we add an edge between any two clause vertices so that the set of clause vertices induces a clique denoted by K in the following. We denote by G_Φ the resulting graph.

▷ **Observation 1.** For any dominating set D of G_Φ and any variable $x \in X$, $|D \cap V(G_x)| \geq 1$. In particular, $\gamma(G_\Phi) \geq |X|$.

▷ **Claim 1.** Φ is satisfiable if and only if $\gamma(G_\Phi) = |X|$.

Proof. Assume that Φ is satisfiable and consider a truth assignment satisfying Φ . We construct a dominating set D of G_Φ as follows. For any variable $x \in X$, if x is true, add the positive literal vertex x to D ; otherwise, add the negative variable vertex \bar{x} to D . Clearly, D is dominating and we conclude by Observation 1 that $\gamma(G_\Phi) = |X|$.

Conversely, assume that $\gamma(G_\Phi) = |X|$ and consider a minimum dominating set D of G_Φ . Then by Observation 1, $|D \cap V(G_x)| = 1$ for any $x \in X$. It follows that $D \cap K = \emptyset$ and so, every clause vertex must be adjacent to some (positive or negative) literal vertex belonging to D . We thus construct a truth assignment satisfying Φ as follows: for any variable $x \in X$, if the positive literal vertex x belongs to D , set x to true; otherwise, set x to false. \triangleleft

\triangleright Claim 2. $\gamma(G_\Phi) = |X|$ if and only if every minimum dominating set of G_Φ is independent.

Proof. Assume that $\gamma(G_\Phi) = |X|$ and consider a minimum dominating set D of G_Φ . Then by Observation 1, $|D \cap V(G_x)| = 1$ for any $x \in X$. It follows that $D \cap K = \emptyset$ and since $N[V(G_x)] \cap N[V(G_{x'})] \subset K$ for any two $x, x' \in X$, D is independent.

Conversely, consider a minimum dominating set D of G_Φ . Since D is independent, $|D \cap V(G_x)| \leq 1$ for any $x \in X$ and we conclude by Observation 1 that in fact, equality holds. Now suppose that there exists $c \in C$, containing variables x_1, x_2 and x_3 , such that the corresponding clause vertex c belongs to D (note that since D is independent, $|D \cap K| \leq 1$). Assume without loss of generality that x_1 occurs positively in c , that is, c is adjacent to the positive literal vertex x_1 . Then, $x_1 \notin D$ since D is independent and so, either $u_{x_1} \in D$ or $\bar{x}_1 \in D$. In the first case, we immediately obtain that $(D \setminus \{u_{x_1}\}) \cup \{x_1\}$ is a minimum dominating set of G_Φ containing an edge, a contradiction. In the second case, since $c \in D$, any vertex dominated by \bar{x}_1 is also dominated by c ; thus, $(D \setminus \{\bar{x}_1\}) \cup \{x_1\}$ is a minimum dominating set of G_Φ containing an edge, a contradiction. Consequently, $D \cap K = \emptyset$ and so, $\gamma(G_\Phi) = |D| = |X|$. \triangleleft

Now by combining Claims 1 and 2, we obtain that Φ is satisfiable if and only if every minimum dominating set of G_Φ is independent, that is, G_Φ is a YES-instance for ALL INDEPENDENT MD. There remains to show that G_Φ is $2P_3$ -free. To see this, it suffices to observe that any induced P_3 of G_Φ contains at least one vertex in the clique K . This concludes the proof. \blacktriangleleft

Theorem 3 now easily follows from Fact 1 and Theorem 8.

6 Conclusion


In this work, we establish a complexity dichotomy for 1-EDGE CONTRACTION(γ) on H -free graphs when H is a connected graph. If we do not require H to be connected, there only remains to settle the complexity status of 1-EDGE CONTRACTION(γ) restricted to H -free graphs when $H = P_3 + qP_2 + pK_1$, with $q \geq 1$ and $p \geq 0$.

References

- 1 Cristina Bazgan, Sonia Toubaline, and Zsolt Tuza. The most vital nodes with respect to independent set and vertex cover. *Discrete Applied Mathematics*, 159:1933–1946, October 2011. doi:10.1016/j.dam.2011.06.023.
- 2 Cristina Bazgan, Sonia Toubaline, and Daniel Vanderpooten. Critical edges for the assignment problem: Complexity and exact resolution. *Operations Research Letters*, 41:685–689, November 2013. doi:10.1016/j.orl.2013.10.001.

- 3 Cédric Bentz, Costa Marie-Christine, Dominique de Werra, Christophe Picouleau, and Bernard Ries. Blockers and Transversals in some subclasses of bipartite graphs: when caterpillars are dancing on a grid. *Discrete Mathematics*, 310:132–146, January 2010. doi:10.1016/j.disc.2009.08.009.
- 4 Cédric Bentz, Costa Marie-Christine, Dominique de Werra, Christophe Picouleau, and Bernard Ries. *Weighted Transversals and Blockers for Some Optimization Problems in Graphs*, pages 203–222. Progress in Combinatorial Optimization. ISTE-WILEY, 2012.
- 5 Marie-Christine Costa, Dominique de Werra, and Christophe Picouleau. Minimum d -blockers and d -transversals in graphs. *Journal of Combinatorial Optimization*, 22(4):857–872, 2011. doi:10.1007/s10878-010-9334-6.
- 6 Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, Heidelberg; New York, fourth edition, 2010.
- 7 Öznur Yaşar Diner, Daniël Paulusma, Christophe Picouleau, and Bernard Ries. Contraction Blockers for Graphs with Forbidden Induced Paths. In *Algorithms and Complexity*, pages 194–207. Springer International Publishing, 2015.
- 8 Öznur Yaşar Diner, Daniël Paulusma, Christophe Picouleau, and Bernard Ries. Contraction and deletion blockers for perfect graphs and H -free graphs. *Theoretical Computer Science*, 746:49–72, 2018. doi:10.1016/j.tcs.2018.06.023.
- 9 Esther Galby, Paloma T. Lima, and Bernard Ries. Reducing the domination number of graphs via edge contractions. In *Mathematical Foundations of Computer Science (MFCS) 2019 (to appear)*, 2019.
- 10 Jia Huang and Jun-Ming Xu. Domination and Total Domination Contraction Numbers of Graphs. *Ars Combinatoria*, 94, January 2010.
- 11 Chaya Keller and Micha A Perles. Blockers for simple Hamiltonian paths in convex geometric graphs of even order. *Discrete & Computational Geometry*, 60(1):1–8, 2018.
- 12 Chaya Keller, Micha A Perles, Eduardo Rivera-Campo, and Virginia Urrutia-Galicia. Blockers for noncrossing spanning trees in complete geometric graphs. In *Thirty Essays on Geometric Graph Theory*, pages 383–397. Springer, 2013.
- 13 Foad Mahdavi Pajouh, Vladimir Boginski, and Eduardo Pasiliao. Minimum Vertex Blocker Clique Problem. *Networks*, 64:48–64, August 2014. doi:10.1002/net.21556.
- 14 C. Moore and J. M. Robson. Hard Tiling Problems with Simple Tiles. *Discrete Computational Geometry*, 26(4):573–590, 2001. doi:10.1007/s00454-001-0047-6.
- 15 Farzaneh Nasirian, Foad Mahdavi Pajouh, and Josephine Namayanja. Exact algorithms for the minimum cost vertex blocker clique problem. *Computers & Operations Research*, 103:296–309, 2019.
- 16 Foad Mahdavi Pajouh, Jose L. Walteros, Vladimir Boginski, and Eduardo L. Pasiliao. Minimum edge blocker dominating set problem. *European Journal of Operational Research*, 247(1):16–26, 2015.
- 17 Daniël Paulusma, Christophe Picouleau, and Bernard Ries. Reducing the Clique and Chromatic Number via Edge Contractions and Vertex Deletions. In *ISCO 2016*, volume 9849 of *LNCS*, pages 38–49, 2016. doi:10.1007/978-3-319-45587-7_4.
- 18 Daniël Paulusma, Christophe Picouleau, and Bernard Ries. Blocking Independent Sets for H -Free Graphs via Edge Contractions and Vertex Deletions. In *TAMC 2017*, volume 10185 of *LNCS*, pages 470–483, 2017. doi:10.1007/978-3-319-55911-7_34.
- 19 Daniël Paulusma, Christophe Picouleau, and Bernard Ries. Critical vertices and edges in H -free graphs. *Discrete Applied Mathematics*, 257:361–367, 2019. doi:10.1016/j.dam.2018.08.016.

Internal Dictionary Matching

Panagiotis Charalampopoulos 

Department of Informatics, King's College London, London, UK
Institute of Informatics, University of Warsaw, Warsaw, Poland
<https://nms.kcl.ac.uk/panagiotis.charalampopoulos/>
panagiotis.charalampopoulos@kcl.ac.uk

Tomasz Kociumaka 

Institute of Informatics, University of Warsaw, Warsaw, Poland
Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel
<https://www.mimuw.edu.pl/~kociumaka/>
kociumaka@mimuw.edu.pl

Manal Mohamed 

Department of Informatics, King's College London, London, UK
manal.mohamed@kcl.ac.uk

Jakub Radoszewski 

Institute of Informatics, University of Warsaw, Warsaw, Poland
Samsung R&D Institute, Warsaw, Poland
<https://www.mimuw.edu.pl/~jrad>
jrad@mimuw.edu.pl

Wojciech Rytter 

Institute of Informatics, University of Warsaw, Warsaw, Poland
<https://www.mimuw.edu.pl/~rytter>
rytter@mimuw.edu.pl

Tomasz Waleń 

Institute of Informatics, University of Warsaw, Warsaw, Poland
<https://www.mimuw.edu.pl/~walen>
walen@mimuw.edu.pl

Abstract

We introduce data structures answering queries concerning the occurrences of patterns from a given dictionary \mathcal{D} in *fragments* of a given string T of length n . The dictionary is *internal* in the sense that each pattern in \mathcal{D} is given as a fragment of T . This way, \mathcal{D} takes space proportional to the number of patterns $d = |\mathcal{D}|$ rather than their total length, which could be $\Theta(n \cdot d)$.

In particular, we consider the following types of queries: reporting and counting *all* occurrences of patterns from \mathcal{D} in a fragment $T[i..j]$ (operations $\text{REPORT}(i, j)$ and $\text{COUNT}(i, j)$ below, as well as operation $\text{EXISTS}(i, j)$ that returns true iff $\text{COUNT}(i, j) > 0$) and reporting *distinct* patterns from \mathcal{D} that occur in $T[i..j]$ (operation $\text{REPORTDISTINCT}(i, j)$). We show how to construct, in $\mathcal{O}((n + d) \log^{\mathcal{O}(1)} n)$ time, a data structure that answers each of these queries in time $\mathcal{O}(\log^{\mathcal{O}(1)} n + |\text{output}|)$ – see the table below for specific time and space complexities.

Query	Preprocessing time	Space	Query time
$\text{EXISTS}(i, j)$	$\mathcal{O}(n + d)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$
$\text{REPORT}(i, j)$	$\mathcal{O}(n + d)$	$\mathcal{O}(n + d)$	$\mathcal{O}(1 + \text{output})$
$\text{REPORTDISTINCT}(i, j)$	$\mathcal{O}(n \log n + d)$	$\mathcal{O}(n + d)$	$\mathcal{O}(\log n + \text{output})$
$\text{COUNT}(i, j)$	$\mathcal{O}(\frac{n \log n}{\log \log n} + d \log^{3/2} n)$	$\mathcal{O}(n + d \log n)$	$\mathcal{O}(\frac{\log^2 n}{\log \log n})$

The case of counting patterns is much more involved and needs a combination of a locally consistent parsing with orthogonal range searching. Reporting distinct patterns, on the other hand, uses the structure of maximal repetitions in strings. Finally, we provide tight – up to subpolynomial factors – upper and lower bounds for the case of a dynamic dictionary.



© Panagiotis Charalampopoulos, Tomasz Kociumaka, Manal Mohamed, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń;
licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 22; pp. 22:1–22:17



Leibniz International Proceedings in Informatics

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

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases string algorithms, dictionary matching, internal pattern matching

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.22

Related Version A full version of the paper is available at <https://arxiv.org/abs/1909.11577>.

Funding *Panagiotis Charalampopoulos*: Supported in part by the ERC grant TOTAL agreement no. 677651.

Tomasz Kociumaka: Supported by ISF grants no. 824/17 and 1278/16 and by an ERC grant MPM under the EU's Horizon 2020 Research and Innovation Programme (grant no. 683064).

Jakub Radoszewski: Supported by the Polish National Science Center, grant number 2018/31/D/ST6/03991.

Tomasz Walen: Supported by the Polish National Science Center, grant number 2018/31/D/ST6/03991.

Acknowledgements Panagiotis Charalampopoulos and Manal Mohamed thank Solon Pissis for preliminary discussions.

1 Introduction

In the problem of dictionary matching, which has been studied for more than forty years, we are given a dictionary \mathcal{D} , consisting of d patterns, and the goal is to preprocess \mathcal{D} so that presented with a text T we are able to efficiently compute the occurrences of the patterns from \mathcal{D} in T . The Aho–Corasick automaton preprocesses the dictionary in linear time with respect to its total length and then processes T in time $\mathcal{O}(|T| + |\text{output}|)$ [1]. Compressed indexes for dictionary matching [9], as well as indexes for approximate dictionary matching [10] have been studied. Dynamic dictionary matching in its more general version consists in the problem where a dynamic dictionary is maintained, text strings are presented as input and for each such text all the occurrences of patterns from the dictionary in the text have to be reported; see [2, 3].

Internal queries in texts have received much attention in recent years. Among them, the *Internal Pattern Matching* (IPM) problem consists in preprocessing a text T of length n so that we can efficiently compute the occurrences of a substring of T in another substring of T . A nearly-linear sized data structure that allows for sublogarithmic-time IPM queries was presented in [22], while a linear sized data structure allowing for constant-time IPM queries in the case that the ratio between the lengths of the two substrings is constant was presented in [25]. Other types of internal queries include computing the longest common prefix of two substrings of T , computing the periods of a substring of T , etc. We refer the interested reader to [23], which contains an overview of the literature.

We introduce the problem of *Internal Dictionary Matching* (IDM) that consists in answering the following types of queries for an internal dictionary \mathcal{D} consisting of substrings of text T : given (i, j) , report/count all occurrences of patterns from \mathcal{D} in $T[i..j]$ and report the distinct patterns from \mathcal{D} that occur in $T[i..j]$.

Some interesting internal dictionaries \mathcal{D} are the ones comprising of palindromic, square, or non-primitive substrings of T . In each of these three cases, the total length of patterns might be quadratic, but the internal dictionary is of linear size and can be constructed in $\mathcal{O}(n)$ time [16, 12, 6]. Our data structure provides a general framework for solving problems related to the internal structure of the string. The case of palindromes has already been studied in [30], where authors proposed a data structure of size $\mathcal{O}(n \log n)$ that returns the number of all distinct palindromes in $T[i..j]$ in $\mathcal{O}(\log n)$ time.

By building upon our solutions for static dictionaries, we provide algorithms for the case of a dynamic dictionary, where patterns can be added to or removed from \mathcal{D} . We show how to process updates in $\tilde{O}(n^\alpha)$ time and answer queries $\text{EXISTS}(i, j)$, $\text{REPORT}(i, j)$ and $\text{REPORTDISTINCT}(i, j)$ in $\tilde{O}(n^{1-\alpha} + |\text{output}|)$ time for any $0 < \alpha < 1$, matching – up to subpolynomial factors – our conditional lower bound.

Our techniques and a roadmap. First, in Section 3, we present straightforward solutions for queries $\text{EXISTS}(i, j)$ and $\text{REPORT}(i, j)$. In Section 4 we describe an involved solution for $\text{REPORTDISTINCT}(i, j)$ queries, that heavily relies on the periodic structure of the input text and on tools that we borrow from computational geometry. In Section 5 we rely on locally consistent parsing and further computational geometry tools to obtain an efficient solution for $\text{COUNT}(i, j)$ queries. Finally, in Section 6 we extend our solutions for the case of a dynamic dictionary and provide a matching conditional lower bound.

2 Preliminaries

We begin with basic definitions and notation generally following [11]. Let $T = T[1]T[2] \cdots T[n]$ be a *string* of length $|T| = n$ over a linearly sortable alphabet Σ . The elements of Σ are called *letters*. By ε we denote an *empty string*. For two positions i and j on T , we denote by $T[i..j] = T[i] \cdots T[j]$ the *fragment* (sometimes called substring) of T that starts at position i and ends at position j (it equals ε if $j < i$). It is called *proper* if $i > 1$ or $j < n$. A fragment of T is represented in $\mathcal{O}(1)$ space by specifying the indices i and j . A *prefix* of T is a fragment that starts at position 1 ($T[1..j]$, notation: $T^{(j)}$) and a *suffix* is a fragment that ends at position n ($T[i..n]$, notation: $T_{(i)}$). We denote the *reverse string* of T by T^R , i.e. $T^R = T[n]T[n-1] \cdots T[1]$.

Let U be a string of length m with $0 < m \leq n$. We say that there exists an *occurrence* of U in T , or, more simply, that U *occurs in* T , when U is a fragment of T . We thus say that U occurs at the *starting position* i in T when $U = T[i..i+m-1]$.

If a string U is both a proper prefix and a proper suffix of a string T of length n , then U is called a *border* of T . A positive integer p is called a *period* of T if $T[i] = T[i+p]$ for all $i = 1, \dots, n-p$. A string T has a period p if and only if it has a border of length $n-p$. We refer to the smallest period as *the period* of the string, and denote it as $\text{per}(T)$, and, analogously, to the longest border as *the border* of the string. A string is called *periodic* if its period is no more than half of its length and *aperiodic* otherwise.

The elements of the dictionary \mathcal{D} are called *patterns*. Henceforth we assume that $\varepsilon \notin \mathcal{D}$, i.e. the length of each $P \in \mathcal{D}$ is at least 1. If ε was in \mathcal{D} , we could trivially treat it individually. We further assume that each pattern of \mathcal{D} is given by the starting and ending positions of its occurrence in T . Thus, the size of the dictionary $d = |\mathcal{D}|$ refers to the number of strings in \mathcal{D} and not their total length.

The *suffix tree* $\mathcal{T}(T)$ of a non-empty string T of length n is a compact trie representing all suffixes of T . The *branching* nodes of the trie as well as the *terminal* nodes, that correspond to suffixes of T , become *explicit* nodes of the suffix tree, while the other nodes are *implicit*. Each edge of the suffix tree can be viewed as an upward maximal path of implicit nodes starting with an explicit node. Moreover, each node belongs to a unique path of that kind. Thus, each node of the trie can be represented in the suffix tree by the edge it belongs to and an index within the corresponding path. We let $\mathcal{L}(v)$ denote the *path-label* of a node v , i.e., the concatenation of the edge labels along the path from the root to v . We say that v is path-labelled $\mathcal{L}(v)$. Additionally, $\delta(v) = |\mathcal{L}(v)|$ is used to denote the *string-depth* of node v .

A terminal node v such that $\mathcal{L}(v) = T_{(i)}$ for some $1 \leq i \leq n$ is also labelled with index i . Each fragment of T is uniquely represented by either an explicit or an implicit node of $\mathcal{T}(T)$, called its *locus*. Once $\mathcal{T}(T)$ is constructed, it can be traversed in a depth-first manner to compute the string-depth $\delta(v)$ for each explicit node v . The suffix tree of a string of length n , over an integer ordered alphabet, can be computed in time and space $\mathcal{O}(n)$ [13]. In the case of integer alphabets, in order to access the child of an explicit node by the first letter of its edge label in $\mathcal{O}(1)$ time, perfect hashing [14] can be used. Throughout the paper, when referring to the suffix tree $\mathcal{T}(T)$ of T , we mean the suffix tree of $T\$$, where $\$ \notin \Sigma$ is a sentinel letter that is lexicographically smaller than all the letters in Σ . This ensures that all terminal nodes are leaves.

We say that a tree is a *weighted tree* if it is a rooted tree with an integer weight on each node v , denoted by $\omega(v)$, such that the weight of the root is zero and $\omega(u) < \omega(v)$ if u is the parent of v . We say that a node v is a *weighted ancestor at depth ℓ* of a node u if v is the highest ancestor of u with weight of at least ℓ . After $\mathcal{O}(n)$ -time preprocessing, weighted ancestor queries for nodes of a weighted tree \mathcal{T} of size n can be answered in $\mathcal{O}(\log \log n)$ time per query [4]. If ω has a property that the difference of weights of a child and its parent is always equal to 1, then the queries can be answered in $\mathcal{O}(1)$ time after $\mathcal{O}(n)$ -time preprocessing [7]; in this special case the values ω are called *levels* and the queries are called *level ancestor queries*. The suffix tree $\mathcal{T}(T)$ is a weighted tree with $\omega = \delta$. Hence, the locus of a fragment $T[i..j]$ in $\mathcal{T}(T)$ is the weighted ancestor of the terminal node with path-label $T_{(i)}$ at string-depth $j - i + 1$.

3 Exists(i, j) and Report(i, j) queries

We first present a convenient modification to the suffix tree with respect to a dictionary \mathcal{D} ; see Fig. 2.

► **Definition 2.** A \mathcal{D} -modified suffix tree of string T is a tree with terminal nodes corresponding to non-empty suffixes of $T\$$ and branching nodes corresponding to $\{\varepsilon\} \cup \mathcal{D}$. A node corresponding to string U is an ancestor of a node corresponding to string V if and only if U is a prefix of V . Each node stores its level as well as its string-depth (i.e., the length of its corresponding string).

► **Lemma 3.** A \mathcal{D} -modified suffix tree of T has size $\mathcal{O}(n + d)$ and can be constructed in $\mathcal{O}(n + d)$ time.

Proof. The \mathcal{D} -modified suffix tree is obtained from the suffix tree $\mathcal{T}(T)$ in two steps.

In the first step, we mark all nodes of $\mathcal{T}(T)$ with path-label equal to a pattern $P \in \mathcal{D}$: if any of them are implicit, we first make them explicit; see Fig. 3(a). We can find the loci of the patterns in $\mathcal{T}(T)$ in $\mathcal{O}(n + d)$ time by answering the weighted ancestor queries as a batch [24], employing a data structure for a special case of Union-Find [15]. (If many implicit nodes along an edge are to become explicit, we can avoid the local sorting based on depth if we sort globally in time $\mathcal{O}(n + d)$ using bucket sort and then add the new explicit nodes in decreasing order with respect to depth.)

In the second step, we recursively contract any edge (u, v) , where u is the parent of v if:

1. both u and v are unmarked, or
2. u is marked and v is an unmarked internal node.

The resulting tree is the \mathcal{D} -modified suffix tree and has $\mathcal{O}(n)$ terminal nodes and $\mathcal{O}(d)$ internal nodes; see Fig. 3(b). ◀

Proof. (a) Let us define an array $B[a] = \min\{b : T[a..b] \in \mathcal{D}\}$. If there is no pattern from \mathcal{D} starting in T at position a , then $B[a] = \infty$. It can be readily verified that the answer to query $\text{EXISTS}(i, j)$ is yes if and only if the minimum element in the subarray $B[i..j]$ is at most j . Thus, in order to answer $\text{EXISTS}(i, j)$ queries, it suffices to construct the array B and a data structure that answers range minimum queries (RMQ) on B . Using the \mathcal{D} -modified suffix tree of T , whose construction time is the bottleneck, array B can be populated in $\mathcal{O}(n)$ time as follows. For each terminal node with path-label $T_{(a)}$ and level greater than 1, we set $B[a]$ to the string-depth of its ancestor at level 1 using a level ancestor query. If the terminal node is at level 1, then $B[a] = \infty$. A data structure answering range minimum queries in $\mathcal{O}(1)$ time can be built in time $\mathcal{O}(n)$ [17, 8].

(b) We first identify all positions $a \in [i..j]$ that are starting positions of occurrences of some pattern $P \in \mathcal{D}$ in $T[i..j]$ using RMQs over array B , which has been defined in the proof of part (a), as follows. The first RMQ, is over the range $[i..j]$ and identifies a position a (if any such position exists). The range is then split into two parts, namely $[i, a-1]$ and $[a+1, j]$. We recursively, use RMQs to identify the remaining positions in each part. Once we have found all the positions where at least one pattern from \mathcal{D} occurs, we report all the patterns occurring at each of these positions and being contained in $T[i..j]$. The complexities follow from Lemmas 3 and 4. \blacktriangleleft

4 ReportDistinct(i, j) queries

Below, we present an algorithm that reports patterns from \mathcal{D} occurring in $T[i..j]$, allowing for $\mathcal{O}(1)$ copies of each pattern on the output. We can then sort these patterns, remove duplicates, and report distinct ones using an additional global array of counters, one for each pattern.

Let us first partition \mathcal{D} into $\mathcal{D}_0, \dots, \mathcal{D}_{\lfloor \log n \rfloor}$ such that $\mathcal{D}_k = \{P \in \mathcal{D} : \lfloor \log |P| \rfloor = k\}$. We call \mathcal{D}_k a k -dictionary. We now show how to process a single k -dictionary \mathcal{D}_k ; the query procedure may clearly assume $k \leq \log |T[i..j]|$.

We precompute an array $L_k[1..n]$ such that $T[a..L_k[a]]$ is the longest pattern in \mathcal{D}_k is a prefix of $T_{(a)}$. We can do this in $\mathcal{O}(n)$ time by inspecting the parents of terminal nodes in the \mathcal{D}_k -modified suffix tree. Next, we assign to all the patterns of \mathcal{D}_k equal to some $T[a..L_k[a]]$ integer identifiers id (or colors) in $[1..n]$, and construct an array $I_k[a] = \text{id}(P)$, where $P = T[a..L_k[a]]$. We then rely on the following theorem.

► **Theorem 6** (Colored Range Reporting [28]). *Given an array $A[1..N]$ of elements from $[1..U]$, we can construct a data structure of size $\mathcal{O}(N)$ in $\mathcal{O}(N+U)$ time, so that upon query $[i..j]$ all distinct elements in $A[i..j]$ can be reported in $\mathcal{O}(1 + |\text{output}|)$ time.*

We first perform a colored range reporting query on the range $[i..j - 2^{k+1}]$ of array I_k and obtain a set of distinct patterns \mathcal{C}_k , employing Theorem 6. We observe the following.

► **Observation 7.** *Any pattern of a k -dictionary \mathcal{D}_k occurring in T at position $p \in [i..j - 2^{k+1}]$ is a prefix of a pattern $P \in \mathcal{C}_k$.*

Based on this observation, we will report the remaining patterns using the \mathcal{D}_k -modified suffix tree, following parent pointers and temporarily marking the loci of reported patterns to avoid double-reporting. We thus now only have to compute the patterns from \mathcal{D}_k that occur in $T[t..j]$, where $t = \max\{i, j - 2^{k+1} + 1\}$.

We further partition \mathcal{D}_k for $k > 1$ to a *periodic k -dictionary* and an *aperiodic k -dictionary*:

$$\mathcal{D}_k^p = \{P \in \mathcal{D}_k : \text{per}(P) \leq 2^k/3\} \quad \text{and} \quad \mathcal{D}_k^a = \{P \in \mathcal{D}_k : \text{per}(P) > 2^k/3\}.$$

Note that we can partition \mathcal{D}_k in $\mathcal{O}(|\mathcal{D}_k|)$ time using the so-called 2-PERIOD QUERIES of [25, 5, 23]. Such a query decides whether a given fragment of the text is periodic and, if so, it also returns its period. It can be answered in $\mathcal{O}(1)$ time after an $\mathcal{O}(n)$ -time preprocessing of the text.

4.1 Processing an aperiodic k -dictionary

We make use of the following sparsity property.

► **Fact 8** (Sparsity of occurrences). *The occurrences of a pattern P of an aperiodic k -dictionary \mathcal{D}_k^a in T start over $\frac{1}{6}|P|$ positions apart.*

Proof. If two occurrences of P started $d \leq \frac{2^k}{3}$ positions apart, then d would be a period of P , contradicting $P \in \mathcal{D}_k^a$. Then, since $2^k \leq |P| < 2^{k+1}$, we have that $2^k/3 \geq \frac{1}{6}|P|$. ◀

► **Lemma 9.** *REPORTDISTINCT(t, j) queries for the aperiodic k -dictionary \mathcal{D}_k^a and $j - t \leq 2^{k+1}$ can be answered in $\mathcal{O}(1 + |\text{output}|)$ time with a data structure of size $\mathcal{O}(n + |\mathcal{D}_k^a|)$, that can be constructed in $\mathcal{O}(n + |\mathcal{D}_k^a|)$ time.*

Proof. Since the fragment $T[t..j]$ is of length at most 2^{k+1} , it may only contain a constant number of occurrences of each pattern in \mathcal{D}_k^a by Fact 8. We can thus simply use a REPORT(t, j) query for dictionary \mathcal{D}_k^a and then remove duplicates. The complexities follow from Theorem 5(b). ◀

4.2 Processing a periodic k -dictionary

Our solution for periodic patterns relies on the well-studied theory of maximal repetitions (*runs*) in strings. A run is a periodic fragment $R = T[a..b]$ which can be extended neither to the left nor to the right without increasing the period $p = \text{per}(R)$, that is, $T[a-1] \neq T[a+p-1]$ and $T[b-p+1] \neq T[b+1]$ provided that the respective letters exist. The number of runs in a string of length n is $\mathcal{O}(n)$ and all the runs can be computed in $\mathcal{O}(n)$ time [26, 5].

► **Observation 10.** *Let P be a periodic pattern. If P occurs in $T[t..j]$, then P is a fragment of a unique run R such that $\text{per}(R) = \text{per}(P)$. We say that this run R extends P .*

Let \mathcal{R} be the set of all runs in T . Following [23], we construct for all $k \in [0.. \lfloor \log n \rfloor]$ the sets of runs $\mathcal{R}_k = \{R \in \mathcal{R} : \text{per}(R) \leq \frac{2^k}{3}, |R| \geq 2^k\}$ in $\mathcal{O}(n)$ time overall. Note that these sets are not disjoint; however, $|\mathcal{R}_k| = \mathcal{O}(\frac{n}{2^k})$ (cf. Lemma 11 below) and thus their total size is $\mathcal{O}(n)$. If U is a fragment of T , by $\mathcal{R}_k(U) \subseteq \mathcal{R}_k$ we denote the set of all runs $R \in \mathcal{R}_k$ such that $|R \cap U| \geq 2^k$, that is, runs whose overlap with the fragment U is at least 2^k .

► **Lemma 11** (see [23, Lemma 4.4.7]). $|\mathcal{R}_k(U)| = \mathcal{O}(\frac{1}{2^k}|U|)$.

Strategy. Given a fragment $U = T[t..j]$, we will first identify all runs $\mathcal{R}_k(U)$ of \mathcal{R}_k that have a sufficient overlap with U . There is a constant number of them by Lemma 11. For an occurrence of a pattern $P \in \mathcal{D}_k^p$ in U , the unique run R extending this occurrence of P must be in $\mathcal{R}_k(U)$. We will preprocess the runs in order to be able to compute a unique (the leftmost) occurrence *induced* by run R for each such pattern P .

► **Lemma 12.** *Let U be a fragment of T of length at most 2^{k+1} . Then $\mathcal{R}_k(U)$ can be retrieved in $\mathcal{O}(1)$ time after an $\mathcal{O}(n)$ -time preprocessing.*

Proof. PERIODIC EXTENSION QUERIES [23, Section 5.1], given a fragment V of the text T as input, return the run R extending V . They can be answered in $\mathcal{O}(1)$ time after $\mathcal{O}(n)$ -time preprocessing.

Let us cover U using $\mathcal{O}(\frac{1}{2^k}|U|)$ fragments of length $\frac{2^{k+1}}{3}$ with overlaps of at least $\frac{2^k}{3}$ and ask a PERIODIC EXTENSION QUERY for each fragment V in the cover. For each run $R \in \mathcal{R}_k(U)$ with sufficient overlap, $R \cap U$ must contain a fragment V in the cover and its periodic extension must be R since $|V| \geq 2 \cdot \text{per}(R)$. ◀

Preprocessing. We construct an array $\ell_k[1..n]$ such that $T[i.. \ell_k[i]]$ is the shortest pattern $P \in \mathcal{D}_k^p$ that occurs at position i . Note that $\ell_k[i]$ can be retrieved in $\mathcal{O}(1)$ time using a level ancestor query in the \mathcal{D}_p^k -modified suffix tree (asking for a level-1 ancestor of the leaf corresponding to $T_{(i)}$, as in the proof of Theorem 5(a)). We then preprocess the array ℓ_k for RMQ queries.

Processing a run at query. Let us begin with a consequence of the fact that the shortest period is primitive.

► **Observation 13.** *If a pattern P occurs in a text Q and satisfies $|P| \geq \text{per}(Q)$, then P has exactly one occurrence in the first $\text{per}(Q)$ positions of Q .*

We use RMQs repeatedly, as in the proof of Theorem 5(b), for the subarray of ℓ_k corresponding to the first $\text{per}(R)$ positions of $R \cap U$. This way, due to Observation 13, we compute exactly the positions where a pattern $P \in \mathcal{D}_k^p$ has its leftmost occurrence in $R \cap U$. The number of positions identified for a single run $R \in \mathcal{R}_k(U)$ is therefore upper bounded by the number of distinct patterns occurring within $R \cap U$. We then report all distinct patterns occurring within $R \cap U$ by processing each such starting position using Lemma 4. There is no double-reporting while processing a single run, by Observation 13 and hence the time required to process this run is $\mathcal{O}(1 + |\text{output}|) - |\text{output}|$ here refers to the number of distinct patterns from \mathcal{D}_k^p occurring within U . Since $|\mathcal{R}_k(U)| = \mathcal{O}(1)$, we report each pattern a constant number of times and the overall time required is $\mathcal{O}(1 + |\text{output}|)$.

The space occupied by our data structure can be reduced to $\mathcal{O}(n + d)$; details can be found in the full version of the paper.

► **Theorem 14.** *REPORTDISTINCT(i, j) queries can be answered in $\mathcal{O}(\log n + |\text{output}|)$ time with a data structure of size $\mathcal{O}(n + d)$ that can be constructed in $\mathcal{O}(n \log n + d)$ time.*

5 Count(i, j) queries

We first solve an auxiliary problem and show how it can be employed to give an unsatisfactory solution to COUNT(i, j). We then refine our approach using recompression and obtain the following.

► **Theorem 15.** *COUNT(i, j) queries can be answered in $\mathcal{O}(\log^2 n / \log \log n)$ time with a data structure of size $\mathcal{O}(n + d \log n)$ that can be constructed in $\mathcal{O}(n \log n / \log \log n + d \log^{3/2} n)$ time.*

5.1 An auxiliary problem

By inter-position $i + 1/2$ we refer to a location between positions i and $i + 1$ in T . We also refer to inter-positions $1/2$ and $n + 1/2$. We consider the following auxiliary problem, in which we are given a set of inter-positions (*breakpoints*) B of P and upon query we are to compute all fragments of $T[i..j]$ that align a specific inter-position (*anchor*) β of the text with some inter-position in B .

BREAKPOINTS-ANCHOR IPM

Input: A length- n text T , its length- m substring P , and a set B of inter-positions (breakpoints) of P .

Query: $\text{COUNT}_\beta(i, j)$: the number of fragments $T[r..r+m-1]$ of $T[i..j]$ that match P such that $\beta - r + 1 \in B$ (β is an anchor).

In the 2D orthogonal range counting problem, one is to preprocess an $n \times n$ grid with $\mathcal{O}(n)$ marked points so that upon query $[x_1, y_1] \times [x_2, y_2]$, the number of points in this rectangle can be computed efficiently. In the (dual) 2D range stabbing counting problem, one is to preprocess the grid with $\mathcal{O}(n)$ rectangles so that upon query (x, y) the number of (stabbed) rectangles that contain (x, y) can be retrieved efficiently. The counting version of range stabbing queries in 2D reduces to two-sided range counting queries in 2D as follows (cf. [29]). For each rectangle $[x_1, y_1] \times [x_2, y_2]$ in grid G , we add points (x_1, y_1) and $(x_2 + 1, y_2 + 1)$ with weight 1 and points $(x_1, y_2 + 1)$ and $(x_2, y_1 + 1)$ with weight -1 in a grid G' . Then the number of rectangles stabbed by point (a, b) in G is equal to the sum of weights of points in $(-\infty, a] \times (-\infty, b]$ in G' . We will use the following result in our solution to BREAKPOINTS-ANCHOR IPM (Lemma 18).

► **Theorem 16** ([27]). *Range counting queries for n points in 2D (rank space) can be answered in time $\mathcal{O}(\log n / \log \log n)$ with a data structure of size $\mathcal{O}(n)$ that can be constructed in time $\mathcal{O}(n\sqrt{\log n})$.*

Data structure. Let $W_1 = \{P[[b]..m] : b \in B\}$ and consider the set W_2 obtained by adding $U\$$ and $U\#$ for each element U of W_1 to an initially empty set, where $\$$ is a letter smaller (resp. $\#$ is larger) than all the letters in Σ . Let W be the compact trie for the set of strings W_2 . For each internal node v of W that does not have an outgoing edge with label $\$$, we add such a (leftmost) edge with a leaf attached to its endpoint. W can be constructed in $\mathcal{O}(|B|)$ time after an $\mathcal{O}(n)$ -time preprocessing of T , allowing for constant-time longest common prefix queries; cf. [11]. We also build the W_1 -modified suffix tree of T and preprocess it for weighted ancestor queries. We keep two-sided pointers between nodes of W and of the W_1 -modified suffix tree of T that have the same path-label. Similarly, let W^R be the compact trie for set Z_2 consisting of elements $U\$$ and $U\#$ for each $U \in Z_1 = \{(P[1..[b]])^R : b \in B\}$. We preprocess W^R analogously. Each of the tries has at most $k = \mathcal{O}(|B|)$ leaves.

Let us now consider a 2D grid of size $k \times k$, whose x -coordinates (resp. y -coordinates) correspond to the leaves of W (resp. W^R). For each $b \in B$ we do the following. Let x_1 and x_2 be the leaves with path-label $P[[b]..m]\$$ and $P[[b]..m]\#$ in W , respectively. Similarly, let y_1 and y_2 be the leaves with path-label $(P[1..[b]])^R\$$ and $(P[1..[b]])^R\#$ in W^R , respectively. We add the rectangle $R_b = [x_1, y_1] \times [x_2, y_2]$ in the grid. An illustration is provided in Fig. 4. We then preprocess the grid for the counting version of 2D range stabbing queries, employing Theorem 16.

Query. Let the longest prefix of $T[[\beta]..j]$ that is a prefix of an element of W_1 be U and its locus in W be u . This can be computed in $\mathcal{O}(\log \log n)$ time using a weighted ancestor query in the W_1 -modified suffix tree of T and following the pointer to W . If u is an explicit

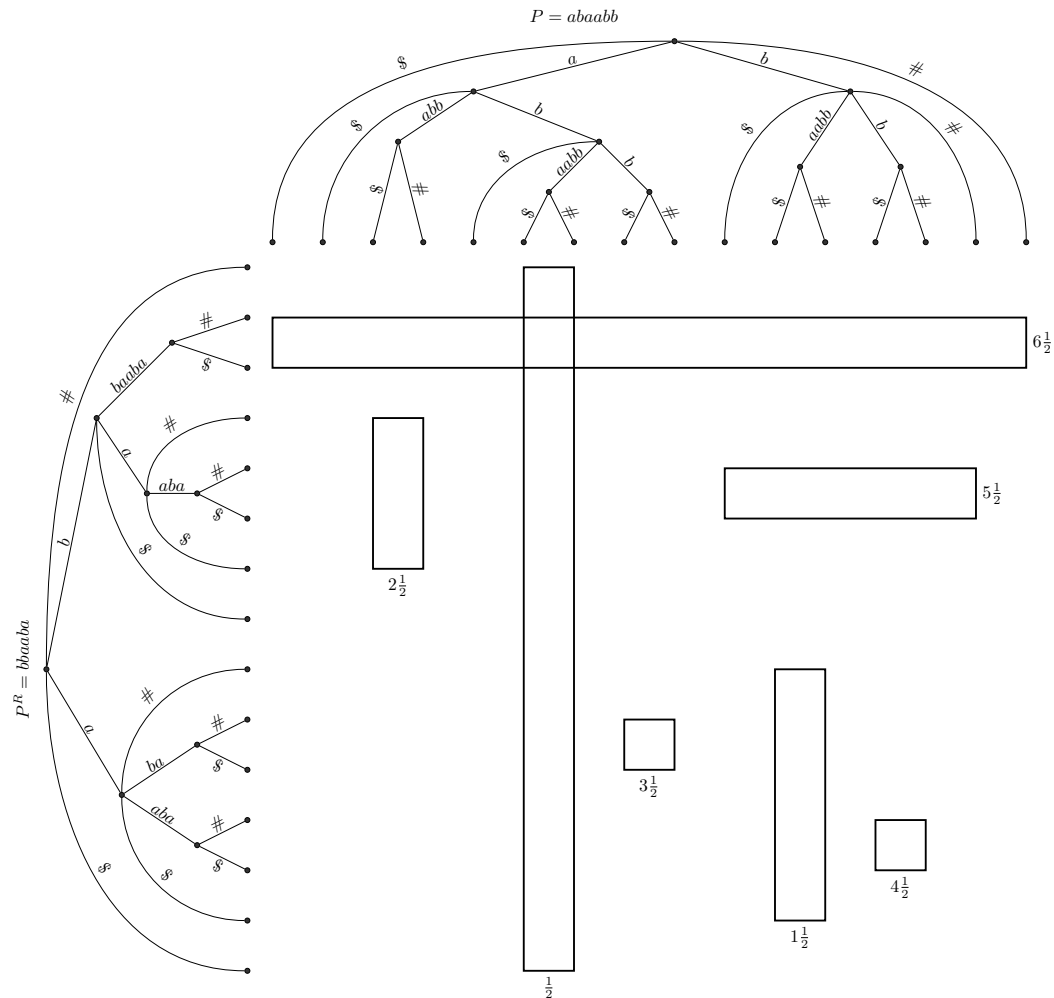
node, we follow the edge with label \$, while if it is implicit along edge (p, q) , we follow the edge with label \$ from p . In either case, we reach a leaf u' . We do the symmetric procedure with $(T[i..[\beta]])^R$ in W^R and obtain a leaf v' .

► **Observation 17.** *The number of fragments $T[r..t] = P$ with $r, t \in [i..j]$ and $\beta - r + 1 \in B$ is equal to the number of rectangles stabbed by the point of the grid defined by u' and v' .*

The observation holds because this point is inside rectangle R_b for $b \in B$ if and only if $P[[b]..m]$ is a prefix of $T[[\beta]..j]$ and $P[1..[b]]$ is a suffix of $T[i..[\beta]]$. This concludes the proof of the following result.

► **Lemma 18.** *BREAKPOINTS-ANCHOR IPM queries can be answered in $\mathcal{O}(\log n / \log \log n)$ time with a data structure of size $\mathcal{O}(n + |B|)$ that can be constructed in time $\mathcal{O}(n + |B|\sqrt{\log |B|})$.*

For the analogously defined problem BREAKPOINTS-ANCHOR IDM, we obtain the following lemma by building trie W for the union of the sets W_2 defined in the above proof for each pattern (similarly for W^R) and adding all rectangles to a single grid.



■ **Figure 4** Example of the construction of rectangles in the proof of Lemma 18 for $P = abaabb$ and breakpoints $i + 1/2$ for $i = 0, 1, 2, 3, 4, 5, 6$. Each rectangle is annotated with its breakpoint.

► **Lemma 19.** BREAKPOINTS-ANCHOR IDM queries can be answered in $\mathcal{O}(\log n / \log \log n)$ time with a data structure of size $\mathcal{O}(n + \sum_{P \in \mathcal{D}} |B_P|)$. The data structure can be constructed in time $\mathcal{O}(n + \sqrt{\log n} \sum_{P \in \mathcal{D}} |B_P|)$.

A warm-up solution for Count(i, j). Lemma 19 can be applied somewhat naively to answer COUNT(i, j) queries as follows. Let us set $B_P = \{p + 1/2 : p \in [1 \dots |P| - 1]\}$ for each pattern $P \in \mathcal{D}$ and construct the data structure of Lemma 19. We build a balanced binary tree BT on top of the text and for each node v in BT define $\text{val}(v)$ to be the fragment consisting of the characters corresponding to the leaves in the subtree of v . Note that if v is a leaf, then $|\text{val}(v)| = 1$; otherwise, $\text{val}(v) = \text{val}(u_\ell)\text{val}(u_r)$, where u_ℓ and u_r are the children of v . For each node v in BT, we precompute and store the count for $\text{val}(v)$. If v is a leaf, this count can be determined easily. Otherwise, each occurrence is contained in $\text{val}(u_\ell)$, is contained in $\text{val}(u_r)$, or spans both $\text{val}(u_\ell)$ and $\text{val}(u_r)$. Hence, we sum the answers for the children u_ℓ and u_r of v and add the result of a BREAKPOINTS-ANCHOR IDM query in $\text{val}(v)$ with the anchor between $\text{val}(u_\ell)$ and $\text{val}(u_r)$.

To answer a query concerning $T[i \dots j]$, we recursively count the occurrences in the intersection of $\text{val}(v)$ with $T[i \dots j]$, starting from the root r of BT as it satisfies $\text{val}(r) = T[1 \dots n]$. If the intersection is empty, the result is 0, and if $\text{val}(v)$ is contained in $T[i \dots j]$, we can use the precomputed count. Otherwise, we recurse on the children u_ℓ and u_r of v and sum the resulting counts. It remains to add the number of occurrences spanning across both $\text{val}(u_\ell)$ and $\text{val}(u_r)$. This value is non-zero only if $T[i \dots j]$ spans both these fragments, and it can be determined from a BREAKPOINTS-ANCHOR IDM query in the intersection of $\text{val}(v)$ and $T[i \dots j]$ with the anchor between $\text{val}(u_\ell)$ and $\text{val}(u_r)$.

The query-time is $\mathcal{O}(\log^2 n / \log \log n)$ since non-trivial recursive calls are made only for nodes on the paths from the root r to the leaves representing $T[i]$ and $T[j]$. Nevertheless, the space required for this “solution” can be $\Omega(nd)$, which is unacceptable. Below, we refine this technique using a locally consistent parsing; our goal is to decrease the size of each set B_P from $\Theta(|P|)$ to $\mathcal{O}(\log n)$.

5.2 Recompression

A *run-length straight line program* (RSLP) is a context-free grammar which generates exactly one string and contains two kinds of non-terminals: *concatenations* with production of the form $A \rightarrow BC$ (for symbols B, C) and *powers* with production of the form $A \rightarrow B^k$ (for a symbol B and an integer $k \geq 2$). Every symbol A generates a unique string denoted $\mathbf{g}(A)$.

Each symbol A is also associated with its *parse tree* $\text{PT}(A)$ consisting of a root labeled with A to which zero or more subtrees are attached: if A is a terminal, there are no subtrees; if $A \rightarrow BC$ is a concatenation symbol, then $\text{PT}(B)$ and $\text{PT}(C)$ are attached; if $A \rightarrow B^k$ is a power symbol, then k copies of $\text{PT}(B)$ are attached. Note that if we traverse the leaves of $\text{PT}(A)$ from left to right, spelling out the corresponding non-terminals, then we obtain $\mathbf{g}(A)$. The parse tree PT of the whole RSLP generating T is defined as $\text{PT}(S)$ for the starting symbol S . We define the *value* $\text{val}(v)$ of a node v in PT to be the fragment $T[a \dots b]$ corresponding to the leaves $T[a], \dots, T[b]$ in the subtree of v . Note that $\text{val}(v)$ is an occurrence of $\mathbf{g}(A)$, where A is the label of v . A sequence of nodes in PT is a *chain* if their values are consecutive fragments in T .

The *recompression* technique by Jež [20, 21] consists in the construction of a particular RSLP generating the input text T . The underlying parse tree PT is of depth $\mathcal{O}(\log n)$ and it can be constructed in $\mathcal{O}(n)$ time. As observed by I [19], this parse tree PT is *locally consistent* in a certain sense. To formalize this property, he introduced the *popped sequence* of every fragment $T[a \dots b]$, which is a sequence of symbols labelling a certain chain of nodes whose values constitute $T[a \dots b]$.

► **Theorem 20** ([19]). *If two fragments are equal, then their popped sequences are equal. Moreover, each popped sequence consists of $\mathcal{O}(\log n)$ runs (maximal powers of a single symbol) and can be constructed in $\mathcal{O}(\log n)$ time. The nodes corresponding to symbols in a run share a single parent. Furthermore, the popped sequence consists of a single symbol only for fragments of length 1.*

Let $F_1^{p_1} \dots F_t^{p_t}$ be the run-length encoding of the popped sequence of a substring S of T . We define

$$L(S) = \{|\mathbf{g}(F_1)|, |\mathbf{g}(F_1^{p_1})|, |\mathbf{g}(F_1^{p_1} F_2^{p_2})|, \dots, |\mathbf{g}(F_1^{p_1} \dots F_{t-1}^{p_{t-1}})|, |\mathbf{g}(F_1^{p_1} \dots F_{t-1}^{p_{t-1}} F_t^{p_t})|\}.$$

By Theorem 20, the set $L(S)$ can be constructed in $\mathcal{O}(\log n)$ time given the occurrence $T[a..b] = S$.

► **Lemma 21.** *Let v be a non-leaf node of PT and let $T[a..b]$ be an occurrence of S contained in $\text{val}(v)$, but not contained in $\text{val}(u)$ for any child u of v . If $T[a..c]$ is the longest prefix of $T[a..b]$ contained in $\text{val}(u)$ for a child u of v , then $|T[a..c]| \in L(S)$. Symmetrically, if $T[c'+1..b]$ is the longest suffix of $T[a..b]$ contained in $\text{val}(u)$ for a child u of v , then $|T[a..c']| \in L(S)$.*

Proof. Consider a sequence v_1, \dots, v_p of nodes in the chain corresponding to the popped sequence of $S = T[a..b]$. Each of these nodes is a descendant of a child of v . Note that $T[a..c] = \text{val}(v_1) \dots \text{val}(v_q)$, where v_1, \dots, v_q is the longest prefix consisting of descendants of the same child. If the labels of v_q and v_{q+1} are distinct, then they belong to distinct runs and $|T[a..c]| \in L(S)$. Otherwise, v_q and v_{q+1} share the same parent: v . Thus, $q = 1$ and $|T[a..c]| = |\text{val}(v_1)| \in L(S)$. The proof of the second claim is symmetric. ◀

Data Structure. We use recompression to build an RSLP generating T and the underlying parse tree PT. We also construct the component of Lemma 19 with $B_P = \{i + \frac{1}{2} : i \in L(P)\}$ for each pattern $P \in \mathcal{D}$. Moreover, for every symbol A we store the number of occurrences of patterns from \mathcal{D} in $\mathbf{g}(A)$. Additionally, if $A \rightarrow B^k$ is a power, we also store the number of occurrences in $\mathbf{g}(B^i)$ for $i \in [1..k]$. The space consumption is $\mathcal{O}(n + d \log n)$ since $|B_P| = \mathcal{O}(\log n)$ for each $P \in \mathcal{D}$.

Efficient preprocessing. The RSLP and the parse tree are built in $\mathcal{O}(n)$ time, and the sets B_P are determined in $\mathcal{O}(d \log n)$ time using Theorem 20. The data structure of Lemma 19 is then constructed in $\mathcal{O}(n + d \log^{3/2} n)$ time. Next, we process the RSLP in a bottom-up fashion. If A is a terminal, its count is easily determined. If $A \rightarrow BC$ is a concatenation, we sum the counts for B and C and the number of occurrences spanning both $\mathbf{g}(B)$ and $\mathbf{g}(C)$. To determine the latter value, we fix an arbitrary node v with label A and denote its children u_ℓ, u_r . By Lemma 21, any occurrence of P intersecting both $\text{val}(u_\ell)$ and $\text{val}(u_r)$ has a breakpoint aligned to the inter-position between the two fragments. Hence, the third summand is the result of a BREAKPOINTS-ANCHOR IDM query in $\text{val}(v)$ with the anchor between $\text{val}(u_\ell)$ and $\text{val}(u_r)$. Finally, if $A \rightarrow B^k$, then to determine the count in $\mathbf{g}(B^i)$, we add the count for B , the count in $\mathbf{g}(B^{i-1})$, and the number of occurrences in B^i spanning both the prefix B and the suffix B^{i-1} . To find the latter value, we fix an arbitrary node v with label A , denote its children u_1, \dots, u_k , and make a BREAKPOINTS-ANCHOR IDM query in $\text{val}(u_1) \dots \text{val}(u_i)$ with the anchor between $\text{val}(u_1)$ and $\text{val}(u_2)$. The correctness of this step follows from Lemma 21. The running time of the last phase is $\mathcal{O}(n \log n / \log \log n)$, so the overall construction time is $\mathcal{O}(n \log n / \log \log n + d \log^{3/2} n)$.

Query. Upon a query $\text{COUNT}(i, j)$, we proceed essentially as in the warm-up solution: we recursively count the occurrences contained in the intersection of $T[i..j]$ with $\text{val}(v)$ for nodes v in PT , starting from the root of PT . If the two fragments are disjoint, the result is 0, and if $\text{val}(v)$ is contained in $T[i..j]$, it is the count precomputed for the label of v . Otherwise, the label of v is a non-terminal. If it is a concatenation symbol, we recurse on both children u_ℓ, u_r of v and sum the obtained counts. If $T[i..j]$ spans both $\text{val}(u_\ell)$ and $\text{val}(u_r)$, we also add the result of a $\text{BREAKPOINTS-ANCHOR IDM}$ query in the intersection of $T[i..j]$ with $\text{val}(v)$ and the anchor between $\text{val}(u_\ell)$ and $\text{val}(u_r)$. If the label is a power symbol $A \rightarrow B^k$, we determine which of the children u_1, \dots, u_k of v are spanned by $T[i..j]$. We denote these children by u_ℓ, \dots, u_r and recurse on u_ℓ and on u_r . If $r > \ell$, we also make a $\text{BREAKPOINTS-ANCHOR IDM}$ query in the intersection of $T[i..j]$ with $\text{val}(u_\ell) \cdots \text{val}(u_r)$ and anchor between $\text{val}(u_\ell)$ and $\text{val}(u_{\ell+1})$. If $r > \ell + 1$, we further add the precomputed value for $\mathbf{g}(B^{r-\ell-1})$ to account for the occurrences contained in $\text{val}(u_{\ell+1}) \cdots \text{val}(u_{r-1})$ and make a $\text{BREAKPOINTS-ANCHOR IDM}$ query in the intersection of $T[i..j]$ with $\text{val}(u_{\ell+1}) \cdots \text{val}(u_r)$ and anchor between u_{r-1} and u_r . By Lemma 21, the answer is the sum of the up to five values computed. The overall query time is $\mathcal{O}(\log^2 n / \log \log n)$, since we make $\mathcal{O}(\log n)$ non-trivial recursive calls and each of them is processed in $\mathcal{O}(\log n / \log \log n)$ time.

6 Dynamic dictionaries

In the Online Boolean Matrix-Vector Multiplication (OMv) problem, we are given as input an $n \times n$ boolean matrix M . Then, we are given in an online fashion a sequence of n vectors r_1, \dots, r_n , each of size n . For each such vector r_i , we are required to output Mr_i before receiving r_{i+1} .

► **Conjecture 22** (OMv Conjecture [18]). *For any constant $\epsilon > 0$, there is no $\mathcal{O}(n^{3-\epsilon})$ -time algorithm that solves OMv correctly with probability at least $2/3$.*

We now present a restricted, but sufficient for our purposes, version of [18, Theorem 2.2].

► **Theorem 23** ([18]). *Conjecture 22 implies that there is no algorithm, for a fixed $\gamma > 0$, that given as input an $r_1 \times r_2$ matrix M , with $r_1 = \lfloor r_2^\gamma \rfloor$, preprocesses M in time polynomial in $r_1 + r_2$ and, then, presented with a vector v of size r_2 , computes Mv in time $\mathcal{O}(r_2^{1+\gamma-\epsilon})$ for $\epsilon > 0$, and has error probability of at most $1/3$.*

We proceed to obtain a conditional lower bound for IDM in the case of a dynamic dictionary. This lower bound clearly carries over to the other problems we considered.

► **Theorem 24.** *The OMv conjecture implies that there is no algorithm that preprocesses T and \mathcal{D} in time polynomial in n , performs insertions to \mathcal{D} in time $\mathcal{O}(n^\alpha)$, answers $\text{EXISTS}(i, j)$ queries in time $\mathcal{O}(n^\beta)$, in an online manner, such that $\alpha + \beta = 1 - \epsilon$ for $\epsilon > 0$, and has error probability of at most $1/3$.*

Proof. Let us suppose that there is such an algorithm and set $\gamma = (\alpha + \epsilon/2)/(\beta + \epsilon/2)$. Given an $r_1 \times r_2$ matrix M , satisfying $r_1 = \lfloor r_2^\gamma \rfloor$, we construct a text T of length $n = r_1 r_2$ as follows. Let T' be a text created by concatenating the rows of M in increasing order. Then obtain T by assigning to each non-zero element of T' the column index of the matrix entry it originates from. Formally, for $i \in [1..r_1 r_2]$, let $a[i] = \lfloor (i-1)/r_2 \rfloor$ and $b[i] = i - a[i]r_2$ and set $T[i] = b[i] \cdot M[a[i] + 1, b[i]]$.

We compute Mv as follows. We add the indices of its at most r_2 non-zero entries in an initially empty dictionary. We then perform queries $\text{EXISTS}(1 + tr_2, (t+1)r_2)$ for $t = 0, \dots, r_1 - 1$. The answer to query $\text{EXISTS}(1 + tr_2, (t+1)r_2)$ is equal to the product of the

t th row of M with v . We thus obtain Mv . In total we perform $\mathcal{O}(r_2)$ insertions to \mathcal{D} and $\mathcal{O}(r_1)$ EXISTS queries. Thus, the total time required is $\mathcal{O}(r_2 n^\alpha + r_1 n^\beta) = \mathcal{O}(n^{\beta+\epsilon/2} n^\alpha + n^{\alpha+\epsilon/2} n^\beta) = \mathcal{O}(n^{1-\epsilon/2}) = \mathcal{O}(r_2^{1+\gamma-\epsilon'})$ for $\epsilon' > 0$. Conjecture 22 would be disproved due to Theorem 23. ◀

► **Example 25.** For the matrix

$$M = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

we construct the text $T = 103000340204$. For the vector $v = [1 \ 1 \ 0 \ 0]^T$, the dictionary is $\mathcal{D} = \{1, 2\}$. The answers to EXISTS(1, 4), EXISTS(5, 8), EXISTS(9, 12) are Yes, No, Yes, respectively, which corresponds to $Mv = [1 \ 0 \ 1]^T$.

A proof of the following theorem, in which we provide algorithms that essentially match this lower bound, can be found in the full version of the paper.

► **Theorem 26.** EXISTS(i, j), REPORT(i, j), REPORTDISTINCT(i, j), and COUNT(i, j) queries for a dynamic dictionary can be answered in $\tilde{\mathcal{O}}(m + |\text{output}|)$ time per query and $\tilde{\mathcal{O}}(n/m)$ time per update for any parameter $m \in [1..n]$ using $\tilde{\mathcal{O}}(n + d)$ space.

7 Final Remarks

The question of whether queries of the type COUNTDISTINCT(i, j), which ask for the number c of patterns from \mathcal{D} that occur in $T[i..j]$, can be answered in time $o(\min\{c, |j-i|\})$ or even $\tilde{\mathcal{O}}(1)$ with a data structure of size $\tilde{\mathcal{O}}(n + d)$ is left open for further investigation. It turns out that our techniques can be used to efficiently answer such queries $\mathcal{O}(\log n)$ -approximately; details are provided in the full version of the paper.

References

- 1 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.
- 2 Amihood Amir, Martin Farach, Zvi Galil, Raffaele Giancarlo, and Kunsoo Park. Dynamic Dictionary Matching. *Journal of Computer and System Sciences*, 49(2):208–222, 1994. doi:10.1016/S0022-0000(05)80047-9.
- 3 Amihood Amir, Martin Farach, Ramana M. Idury, Johannes A. La Poutré, and Alejandro A. Schäffer. Improved Dynamic Dictionary Matching. *Information and Computation*, 119(2):258–282, 1995. doi:10.1006/inco.1995.1090.
- 4 Amihood Amir, Gad M. Landau, Moshe Lewenstein, and Dina Sokol. Dynamic text and static pattern matching. *ACM Transactions on Algorithms*, 3(2):19, 2007. doi:10.1145/1240233.1240242.
- 5 Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, and Kazuya Tsuruta. The “Runs” Theorem. *SIAM Journal on Computing*, 46(5):1501–1514, 2017. doi:10.1137/15M1011032.
- 6 Hideo Bannai, Shunsuke Inenaga, and Dominik Köppl. Computing All Distinct Squares in Linear Time for Integer Alphabets. In Juha Kärkkäinen, Jakub Radoszewski, and Wojciech Rytter, editors, *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017*, volume 78 of *LIPIcs*, pages 22:1–22:18. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.CPM.2017.22.
- 7 Michael A. Bender and Martin Farach-Colton. The Level Ancestor Problem simplified. *Theoretical Computer Science*, 321(1):5–12, 2004. doi:10.1016/j.tcs.2003.05.002.

- 8 Michael A. Bender, Martin 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. doi:10.1016/j.jalgor.2005.08.001.
- 9 Ho-Leung Chan, Wing-Kai Hon, Tak Wah Lam, and Kunihiko Sadakane. Dynamic dictionary matching and compressed suffix trees. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005*, pages 13–22. SIAM, 2005. URL: <http://dl.acm.org/citation.cfm?id=1070432.1070436>.
- 10 Richard Cole, Lee-Ad Gottlieb, and Moshe Lewenstein. Dictionary matching and indexing with errors and don't cares. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, STOC 2004*, pages 91–100. ACM, 2004. doi:10.1145/1007352.1007374.
- 11 Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on Strings*. Cambridge University Press, 2007. doi:10.1017/cbo9780511546853.
- 12 Maxime Crochemore, Costas S. Iliopoulos, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Extracting powers and periods in a word from its runs structure. *Theoretical Computer Science*, 521:29–41, 2014. doi:10.1016/j.tcs.2013.11.018.
- 13 Martin Farach-Colton, Paolo Ferragina, and S. Muthukrishnan. On the Sorting-complexity of Suffix Tree Construction. *Journal of the ACM*, 47(6):987–1011, November 2000. doi:10.1145/355541.355547.
- 14 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.
- 15 Harold N. Gabow and Robert Endre Tarjan. A Linear-Time Algorithm for a Special Case of Disjoint Set Union. *Journal of Computer and System Sciences*, 30(2):209–221, 1985. doi:10.1016/0022-0000(85)90014-5.
- 16 Richard Groult, Élise Prieur, and Gwénaél Richomme. Counting distinct palindromes in a word in linear time. *Information Processing Letters*, 110(20):908–912, 2010. doi:10.1016/j.ipl.2010.07.018.
- 17 Dov Harel and Robert Endre Tarjan. Fast Algorithms for Finding Nearest Common Ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984. doi:10.1137/0213024.
- 18 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and Strengthening Hardness for Dynamic Problems via the Online Matrix-Vector Multiplication Conjecture. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015*, pages 21–30. ACM, 2015. doi:10.1145/2746539.2746609.
- 19 Tomohiro I. Longest Common Extensions with Recompression. In Juha Kärkkäinen, Jakub Radoszewski, and Wojciech Rytter, editors, *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017*, volume 78 of *LIPICs*, pages 18:1–18:15. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.CPM.2017.18.
- 20 Artur Jeż. Faster Fully Compressed Pattern Matching by Recompression. *ACM Transactions on Algorithms*, 11(3):20:1–20:43, 2015. doi:10.1145/2631920.
- 21 Artur Jeż. Recompression: A Simple and Powerful Technique for Word Equations. *Journal of the ACM*, 63(1):4:1–4:51, 2016. doi:10.1145/2743014.
- 22 Orgad Keller, Tsvi Kopelowitz, Shir Landau Feibish, and Moshe Lewenstein. Generalized substring compression. *Theoretical Computer Science*, 525:42–54, 2014. doi:10.1016/j.tcs.2013.10.010.
- 23 Tomasz Kociumaka. *Efficient Data Structures for Internal Queries in Texts*. PhD thesis, University of Warsaw, 2018. URL: <https://mimuw.edu.pl/~kociumaka/files/phd.pdf>.
- 24 Tomasz Kociumaka, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. A Linear Time Algorithm for Seeds Computation, 2019. arXiv:1107.2422.
- 25 Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Internal Pattern Matching Queries in a Text and Applications. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015*, pages 532–551. SIAM, 2015. doi:10.1137/1.9781611973730.36.

- 26 Roman M. Kolpakov and Gregory Kucherov. Finding Maximal Repetitions in a Word in Linear Time. In *40th Annual Symposium on Foundations of Computer Science, FOCS 1999*, pages 596–604. IEEE Computer Society, 1999. doi:10.1109/SFFCS.1999.814634.
- 27 J. Ian Munro, Yakov Nekrich, and Jeffrey Scott Vitter. Fast construction of wavelet trees. *Theoretical Computer Science*, 638:91–97, 2016. doi:10.1016/j.tcs.2015.11.011.
- 28 S. Muthukrishnan. Efficient algorithms for document retrieval problems. In David Eppstein, editor, *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2002*, pages 657–666. SIAM, 2002. URL: <http://dl.acm.org/citation.cfm?id=545381.545469>.
- 29 Mihai Pătraşcu. Unifying the Landscape of Cell-Probe Lower Bounds. *SIAM Journal on Computing*, 40(3):827–847, 2011. doi:10.1137/09075336X.
- 30 Mikhail Rubinchik and Arseny M. Shur. Counting Palindromes in Substrings. In Gabriele Fici, Marinella Sciortino, and Rossano Venturini, editors, *String Processing and Information Retrieval - 24th International Symposium, SPIRE 2017*, volume 10508 of *Lecture Notes in Computer Science*, pages 290–303. Springer, 2017. doi:10.1007/978-3-319-67428-5_25.

Approximating the Geometric Edit Distance

Kyle Fox

The University of Texas at Dallas, USA
kyle.fox@utdallas.edu

Xinyi Li

The University of Texas at Dallas, USA
Xinyi.Li2@utdallas.edu

Abstract

Edit distance is a measurement of similarity between two sequences such as strings, point sequences, or polygonal curves. Many matching problems from a variety of areas, such as signal analysis, bioinformatics, etc., need to be solved in a geometric space. Therefore, the geometric edit distance (GED) has been studied. In this paper, we describe the first strictly sublinear approximate near-linear time algorithm for computing the GED of two point sequences in constant dimensional Euclidean space. Specifically, we present a randomized $O(n \log^2 n)$ time $O(\sqrt{n})$ -approximation algorithm. Then, we generalize our result to give a randomized α -approximation algorithm for any $\alpha \in [1, \sqrt{n}]$, running in time $\tilde{O}(n^2/\alpha^2)$. Both algorithms are Monte Carlo and return approximately optimal solutions with high probability.

2012 ACM Subject Classification Theory of computation \rightarrow Approximation algorithms analysis; Theory of computation \rightarrow Computational geometry

Keywords and phrases Geometric edit distance, Approximation, Randomized algorithms

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.23

Related Version A full version of the paper is available at <https://arxiv.org/abs/1910.00773>.

Acknowledgements The authors would like to thank Anne Driemel and Benjamin Raichel for helpful discussions.

1 Introduction

Ordered sequences are frequently studied objects in the context of similarity measurements, because sequence alignment plays a vital role in trajectory comparison and pattern recognition. As a consequence, several metrics have been developed to measure the similarity of two sequences, e.g., Fréchet distance, dynamic time warping, and their variations. Geometric edit distance, a natural extension of the string metric to geometric space, is the focus of this paper. This concept is formally introduced by Agarwal et al. [2]; however, a similar idea (extending string edit distance to a geometric space) has been applied in other ways during the past decade. Examples include an l^p -type edit distance for biological sequence comparison [19], ERP (Edit distance with Real Penalty) [10], EDR (Edit Distance on Real sequence) [11], TWED (Time Warping Edit Distance) [16] and a matching framework from Swaminathan et al. [18] motivated by computing the similarity of time series and trajectories. See also a survey by Wang et al. [22].

Problem statement

Geometric Edit Distance (GED) is the minimum cost of any matching between two geometric point sequences that respects order along the sequences. The cost includes a constant penalty for each unmatched point.



© Kyle Fox and Xinyi Li;

licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 23; pp. 23:1–23:16

Leibniz International Proceedings in Informatics



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

23:2 Approximating the Geometric Edit Distance

Formally, let $P = \langle p_1, \dots, p_m \rangle$ and $Q = \langle q_1, \dots, q_n \rangle$ be two point sequences in \mathbb{R}^d for some constant d . A monotone *matching* \mathcal{M} is a set of index pairs $\{(i_1, j_1), \dots, (i_k, j_k)\}$ such that for any two elements (i, j) and (i', j') in \mathcal{M} , $i < i'$ if $j < j'$.

We call every unmatched point a *gap point*. Let $\Gamma(\mathcal{M})$ be the set of all gap points. The *cost* of \mathcal{M} is defined as

$$\delta(\mathcal{M}) = \sum_{(i,j) \in \mathcal{M}} \text{dist}(p_i, q_j) + \rho(\Gamma(\mathcal{M})) \quad (1)$$

where $\text{dist}(p, q)$ is the distance between points p and q (i.e. the Euclidean norm), and $\rho(\Gamma(\mathcal{M}))$ is a function of all gap points, which is known as a *gap penalty function*. The use of gap points and the gap penalty function allow us to recognize good matchings even in the presence of outlier points. The distance is sensitive to scaling, so, we can only match points pairs that are sufficiently close together based on the current position. For geometric edit distance, we use a linear gap function. That is to say, $\rho(\Gamma(\mathcal{M})) = |\Gamma(\mathcal{M})| \cdot \ell$, where ℓ is a constant parameter called the *gap penalty*.

► **Definition 1.** We denote the GED between two sequences P, Q as:

$$\text{GED}(P, Q) = \min_{\mathcal{M}} \delta(\mathcal{M}) = \min_{\mathcal{M}} \left(\sum_{(i,j) \in \mathcal{M}} \text{dist}(p_i, q_j) + |\Gamma(\mathcal{M})| \cdot \ell \right)$$

where the minimum is taken over all monotone matchings. Without loss of generality, we assume $\ell = 1$ throughout the paper.

Prior work

To simplify the presentation of prior work, we assume $n \geq m$. It is trivial to compute $\text{GED}(P, Q)$ in $O(mn)$ time by simply changing the cost of substitution in the original string edit distance (Levenstein distance) dynamic programming algorithm [21]. Assuming k is the GED, we can achieve an $O(nk)$ time algorithm by restricting our attention to the middle k diagonals of the dynamic programming table (see also Ukkonen [20]). There is a slightly subquadratic $O(n^2/\log n)$ time algorithm [17] for the string version, but it appears unlikely we can apply it to the geometric case. Accordingly, Gold and Sharir [12] proposed a different algorithm which can compute GED as well as the closely related dynamic time warping (DTW) distance in $O(n^2 \log \log n / \log \log n)$ time in polyhedral metric spaces. Recent papers have shown conditional lower bounds for several sequence distance measures even with some restrictions. In particular, there is no $O(n^{2-\delta})$ time algorithm for any constant $\delta > 0$ for Fréchet distance [5], DTW over a constant size alphabet [1] or restricted to one-dimensional curves [6], and string edit distance on the binary alphabet [4, 6].¹ The latter of the above results implies the same lower bound for GED, even assuming the sequences consist entirely of 0, 1-points in \mathbb{R} .

Due to these limitations and difficulties, many researchers have turned to approximation algorithms for these distances. Much work has been done to explore approximate algorithms for Fréchet distance, DTW, and string edit distance [2, 3, 7–9, 14]. In particular, Bringmann

¹ The (discrete) Fréchet and DTW distances are defined similarly to GED; however, they use one-to-many correspondences instead of one-to-one matchings, and they disallow the use of gap points. As in GED, DTW aims to minimize the sum of distances between corresponding points, while discrete Fréchet distance aims to minimize the maximum distance over corresponding points.

and Mulzer [7] describe an α -approximation algorithm for the discrete Fréchet distance that runs in time $O(n \log n + n^2/\alpha)$ for any $\alpha \in [1, n]$. Chan and Rahmati [9] improved this running time to $O(n \log n + n^2/\alpha^2)$. Very recently, Kuszmaul [14] provided $O(\alpha)$ -approximation algorithms with $O((n^2/\alpha) \text{polylog } n)$ running times for edit distance over arbitrary metric spaces and DTW over well separated tree metrics. Another $O(n^2/\alpha)$ time algorithm with an $O(\alpha)$ approximation factor for *string* edit distance is to run Ukkonen's [20] $O(nk)$ time algorithm letting k be n/α , and unmatch all characters if this algorithm cannot return the optimal matching. Similarly, we can obtain a different $O(\alpha)$ -approximation algorithm for GED running in $O(n^2/\alpha)$ time by making use of the $O(nk)$ time exact algorithm mentioned above. There are many other approximation algorithms specialized for the string version of edit distance. In particular, an $O(\sqrt{n})$ -approximation algorithm can be acquired easily from an $O(n + k^2)$ time exact algorithm [15]. The current best results include papers with $(\log n)^{O(1/\epsilon)}$ [3] and constant approximation ratios [8] with different running time tradeoffs.

For GED, a simple linear time $O(n)$ -approximation algorithm was observed by Agarwal et al. [2]. In the same paper, they also offered a subquadratic time (near-linear time in some scenarios) approximation scheme on several well-behaved families of sequences. Using the properties of these families, they reduced the search space to find the optimal admissible path in the dynamic programming graph [2].

Our results

Inspired by the above applications and prior work, we commit to finding a faster approach to approximating GED between general point sequences while also returning the approximate best matching. Here, we give the first near-linear time algorithm to compute GED with a strictly sublinear approximation factor. We then generalize our result to achieve a tradeoff between the running time and approximation factor. Both of these algorithms are Monte Carlo algorithms, returning an approximately best matching with high probability². To simplify our exposition, we assume the points are located in the plane (i.e., $d = 2$), and we assume the input sequences are the same length (i.e., $m = n$). We can easily extend our results to the unbalanced case, and our analysis implies that outside the plane, the running times and approximation ratios increase only by a factor polynomial in d .

► **Theorem 2.** *Given two point sequences P and Q in \mathbb{R}^2 , each with n points, there exists an $O(n \log^2 n)$ -time randomized algorithm that computes an $O(\sqrt{n})$ -approximate monotone matching for geometric edit distance with high probability.*

The intuitive idea behind this algorithm is very simple. We check if the GED is less than each of several geometrically increasing values g , each of which is less than $O(\sqrt{n})$. For each g , we transform the geometric sequences into strings using a randomly shifted grid, and run the $O(n + k^2)$ time exact algorithm for strings [15]. If the GED is less than g , then we get an $O(\sqrt{n})$ approximate matching. If we never find a matching of cost $O(\sqrt{n})$, we simply leave all points unmatched as this empty matching is an $O(\sqrt{n})$ -approximation for GED with high probability. We give the details for this $O(\sqrt{n})$ -approximation algorithm in Section 2.

► **Theorem 3.** *Given two point sequences P and Q in \mathbb{R}^2 , each with n points, there exists an $O(n \log^2 n + \frac{n^2}{\alpha^2} \log n)$ -time randomized algorithm that computes an $O(\alpha)$ -approximate monotone matching for geometric edit distance with high probability for any $\alpha \in [1, \sqrt{n}]$.*

² We say an event occurs with high probability if it occurs with probability at least $1 - \frac{1}{n^c}$ for some constant $c > 0$.

The second algorithm uses similar techniques to the former, except we can no longer use the string edit distance algorithm as a black box. In particular, we cannot achieve our desired time-approximation tradeoff by just directly altering some parameters in our first algorithm. We discuss why in Section 3.1. To overcome these difficulties, we develop a constant-factor approximation algorithm to compute the GED of point sequences obtained by snapping points of the original input sequences to grid cell corners. Our algorithm for these snapped points is based on the exact algorithm for string edit distance [15] but necessarily more complicated to handle geometric distances. So, we first introduce the $O(n + k^2)$ time algorithm for strings in Section 4.1, and then describe our constant approximation algorithm for points in Section 4.2. We note that a key component of the string algorithm and our extension is a fast method for finding maximal length common substrings from a given pair of starting positions in two strings A and B . A similar procedure was needed in the discrete Fréchet distance approximation of Chan and Rahmati [9]. In Section 3, we present the algorithm for Theorem 3 using our approximation algorithm for snapped point sequences as a black box.

2 $O(\sqrt{n})$ -Approximation for GED

Recall that the main part of our algorithm is a decision procedure to check if the GED is less than a guess value g . There are two steps in this process:

1. Transform the point sequences into strings. To be specific, we partition nearby points into common groups and distant points into different groups to simulate the identical characters and different characters in the string version of edit distance.
2. Run a modification of the exact string edit distance algorithm of Landau *et al.* [15]. To better serve us when discussing geometric edit distance, we aim to minimize the number of insertions and deletions to turn S into T *only*; we consider substitution to have infinite cost. Details on this modified algorithm appear in Section 4.1.³

We explain how to transform the point sequences into strings in Section 2.1, and we analyze the approximation factor and running time in Sections 2.2 and 2.3.

For convenience, we refer to the string edit distance algorithm as $SED(S, T, k)$, where S and T are two strings with equal length. This algorithm will return a matching in $O(n + k^2)$ time if the edit distance is at most k . We give an outline of our algorithm as Algorithm 1. Here, c is a sufficiently large constant, and we use \lg to denote the logarithm of base 2.

2.1 Transformation by a random grid

As stated above, the transformation technique should partition nearby points into common groups and distant points into different groups. We use a randomly shifted grid to realize this ideal, see [13] for example.

Recall P and Q lie in \mathbb{R}^2 . We cover the space with a grid. Let the side length of each grid cell be Δ , and let b be a vector chosen uniformly at random from $[0, \Delta]^2$. Starting from an arbitrary position, the grid shifts b_i units in each dimension i . For a point p , let $id_{\Delta, b}(p)$ denote the cell which contains p in this configuration. We consider two points $p_1 = (x_1, y_1)$, and $p_2 = (x_2, y_2)$ in this space.

³ Computing this variant of the string edit distance is really us computing the shortest common super-sequence length of the strings rather than the traditional Levenshtein distance, but we stick with “edit distance” for simplicity.

■ **Algorithm 1** $O(\sqrt{n})$ -approximation algorithm for GED.

Input: Point sequences P and Q
Output: An approximately optimal matching for GED

```

1 if  $\sum_{i=1}^n \text{dist}(p_i, q_i) \leq 1$  then
2   | return matching  $\{(1, 1), \dots, (n, n)\}$ 
3 else
4   | for  $i := 0$  to  $\lceil \lg \sqrt{n} \rceil$  do
5     |  $g := 2^i$ 
6     | for  $j := 1$  to  $\lceil c \lg n \rceil$  do
7       | Transform  $P, Q$  to strings  $S, T$  using a randomly shifted grid
8       |  $\text{out} := \text{SED}(S, T, 12\sqrt{n} + 2g)$ 
9       | if  $\text{out} \neq \text{false}$  then
10      | | return out
11      | end
12    | end
13  end
14  return the empty matching
15 end

```

► **Lemma 4.** We have $P(\text{id}_{\Delta,b}(p_1) \neq \text{id}_{\Delta,b}(p_2)) \leq \min\{\frac{|x_1-x_2|+|y_1-y_2|}{\Delta}, 1\}$.

We use this observation in our algorithm and set $\Delta = \frac{g}{\sqrt{n}}$ as each cell's side length.

2.2 Time complexity

We claim the running time for Algorithm 1 is $O(n \log^2 n)$. Computing $\sum_{i=1}^n \text{dist}(p_i, q_i)$ takes $O(n)$ time. In the inner loop, the transformation operation (line 7) takes $O(n)$ time assuming use of a hash table. The running time for $\text{SED}(S, T, 12\sqrt{n} + 2g)$ is $O(n)$ for $g = O(\sqrt{n})$. Summing over the outer loop and inner loop, the overall running time for Algorithm 1 is

$$\sum_{i=1}^{\lceil \lg \sqrt{n} \rceil} \sum_{j=1}^{\lceil c \lg n \rceil} O(n) = O(n \log^2 n).$$

2.3 Approximation ratio

In this section, we show that Algorithm 1 returns an $O(\sqrt{n})$ -approximate matching with high probability.

Notation

For any monotone matching \mathcal{M} , we define $C_S(\mathcal{M})$ as the cost of the corresponding edit operations for \mathcal{M} in the string case and $C_G(\mathcal{M})$ to be $\delta(\mathcal{M})$ as defined in (1) for the geometric case (as stated, there is no substitution operation in our modified string case). Let \mathcal{M}_G^* be the optimal matching for geometric edit distance, and \mathcal{M}_S^* be the optimal matching under the string configuration during one iteration of the loop. Our final goal is to establish the relationship between $C_G(\mathcal{M}_G^*)$ and $C_G(\mathcal{M}_S^*)$.

► **Lemma 5.** If $\text{GED}(S, T) \leq g$, with a probability at least $1 - \frac{1}{n^c}$, at least one of the $\lceil c \lg n \rceil$ iterations of the inner for loop will return a matching \mathcal{M}_S^* where $C_S(\mathcal{M}_S^*) \leq 12\sqrt{n} + 2g$.

23:6 Approximating the Geometric Edit Distance

Proof. Let \mathcal{M} be a monotone matching, and let $UM_{\mathcal{M}}$ be the set of unmatched indices. There are four subsets of pairs in \mathcal{M} :

- $OC_{\mathcal{M}}$: In each pair, both indices' points fall into One cell, and the distance between the two points is less and equal to $\frac{g}{\sqrt{n}}$ (Close).
- $OF_{\mathcal{M}}$: In each pair, both indices' points fall into One cell, and the distance between the two points is larger than $\frac{g}{\sqrt{n}}$ (Far).
- $DC_{\mathcal{M}}$: In each pair, the indices' points are in Different cells, and the distance between the two points is less and equal to $\frac{g}{\sqrt{n}}$ (Close).
- $DF_{\mathcal{M}}$: In each pair, the indices' points are in Different cells and the distance between the two points is larger than $\frac{g}{\sqrt{n}}$ (Far).

These sets are disjoint, so

$$\begin{aligned} C_G(\mathcal{M}_G^*) &= |UM_{\mathcal{M}_G^*}| + \sum_{(i,j) \in OC_{\mathcal{M}_G^*}} dist(p_i, q_j) + \sum_{(i,j) \in OF_{\mathcal{M}_G^*}} dist(p_i, q_j) \\ &\quad + \sum_{(i,j) \in DC_{\mathcal{M}_G^*}} dist(p_i, q_j) + \sum_{(i,j) \in DF_{\mathcal{M}_G^*}} dist(p_i, q_j). \end{aligned} \quad (2)$$

Recall that there is no substitution operation in our version of the string case. So to understand optimal matchings for string edit distance, we must unmatched all the pairs in $DC_{\mathcal{M}_G^*}$ and $DF_{\mathcal{M}_G^*}$, forming a new matching \mathcal{M}_G^* . Points in one cell are regarded as identical characters while those in different cells are different characters. Therefore,

$$\begin{aligned} C_S(\mathcal{M}_S^*) &= |UM_{\mathcal{M}_G^*}| + 0 \cdot (|OC_{\mathcal{M}_G^*}| + |OF_{\mathcal{M}_G^*}|) + 2 \cdot (|DC_{\mathcal{M}_G^*}| + |DF_{\mathcal{M}_G^*}|) \\ &= |UM_{\mathcal{M}_G^*}| + 2 \cdot (|DC_{\mathcal{M}_G^*}| + |DF_{\mathcal{M}_G^*}|). \end{aligned}$$

Observe that there are at most $\frac{g}{g/\sqrt{n}} = \sqrt{n}$ pairs in $DF_{\mathcal{M}_G^*}$ if $C_G(\mathcal{M}_G^*) \leq g$. Therefore,

$$\begin{aligned} C_S(\mathcal{M}_S^*) &\leq C_S(\mathcal{M}_G^*) \\ &= |UM_{\mathcal{M}_G^*}| + 2|DC_{\mathcal{M}_G^*}| + 2|DF_{\mathcal{M}_G^*}| \leq g + 2\sqrt{n} + 2|DC_{\mathcal{M}_G^*}| \end{aligned} \quad (3)$$

For any two points p_i, q_j , let $P_D(i, j)$ be the probability that p_i and q_j are assigned into different cells. From Lemma 4, we can infer $P_D(i, j) \leq \frac{2dist(p_i, q_j)}{g/\sqrt{n}}$.

Then,

$$\begin{aligned} E(|DC_{\mathcal{M}_G^*}|) &\leq \sum_{(i,j) \in \mathcal{M}_G^*} P_D(i, j) \leq \sum_{(i,j) \in \mathcal{M}_G^*} \frac{2dist(p_i, q_j)}{g/\sqrt{n}} \\ &\leq 2\sqrt{n}. \end{aligned} \quad (4)$$

Therefore,

$$E(C_S(\mathcal{M}_S^*)) \leq 6\sqrt{n} + g.$$

By Markov's inequality,

$$P[C_S(\mathcal{M}_S^*) \geq 12\sqrt{n} + 2g] \leq \frac{1}{2}.$$

In other words, $SED(S, T, 12\sqrt{n} + 2g)$ will fail with probability at most $\frac{1}{2}$ if $GED(P, Q) \leq g$. So, if we test $SED(S, D, 12\sqrt{n} + 2g)$ $\lceil c \lg n \rceil$ times, at least one iteration will return a value if $GED(P, Q) \leq g$ with a probability greater than or equal to

$$1 - \prod_1^{\lceil c \lg n \rceil} P[C_S(\mathcal{M}_S^*) \geq 12\sqrt{n} + 2g] \geq 1 - \prod_1^{\lceil c \lg n \rceil} \frac{1}{2} = 1 - \frac{1}{n^c}.$$

We conclude the proof of Lemma 5. ◀

According to Lemma 5, if all test procedures return false, we can say $C_G(\mathcal{M}_G^*) > g$ with high probability; otherwise, we obtain a matching \mathcal{M}_S^* and $C_S(\mathcal{M}_S^*) \leq 12\sqrt{n} + 2g$.

We now consider $C_G(\mathcal{M}_S^*)$. Again, $UM_{\mathcal{M}}$ is the set of unmatched indices for a matching \mathcal{M} . Observe, for all $(i, j) \in \mathcal{M}_S^*$, points p_i and q_j lie in the same grid cell. Therefore, $\text{dist}(p_i, q_j) \leq \frac{\sqrt{2}g}{\sqrt{n}}$ if $(i, j) \in \mathcal{M}_S^*$. We have:

$$\begin{aligned} C_G(\mathcal{M}_S^*) &= |UM_{\mathcal{M}_S^*}| + \sum_{(i,j) \in \mathcal{M}_S^*} \text{dist}(p_i, q_j) \\ &\leq 12\sqrt{n} + 2g + n \cdot \left(\frac{\sqrt{2}g}{\sqrt{n}}\right) = 12\sqrt{n} + 2g + \sqrt{2}g\sqrt{n} \end{aligned} \quad (5)$$

If $\text{GED}(P, Q) \leq \sqrt{n}$, then, with high probability, we obtain a matching \mathcal{M}_S^* during the iteration where $g \geq \text{GED}(P, Q) \geq \frac{1}{2}g$. The cost of this matching is at most $12\sqrt{n} + 2g + \sqrt{2}g\sqrt{n} = O(\sqrt{n})\text{GED}(P, Q)$. The same approximation bound holds if $\text{GED}(P, Q) > \sqrt{n}$, whether or not we find a matching during the outer for loop. We conclude the proof of Theorem 2.

3 $O(\alpha)$ -Approximation for GED

We now discuss our $O(\alpha)$ -approximation algorithm for any $\alpha \in [1, \sqrt{n}]$. A natural approach for extending our $O(\sqrt{n})$ -approximation is using the same reduction to string edit distance but let the cell's side length be a variable depending on the approximation factor α . However, this method does not appear to work well.

3.1 Flaws in $O(\sqrt{n})$ -algorithm to achieve tradeoff

Let Δ_α be the cell's side length which depends on the approximation factor α . For our analysis we need $C_G(\mathcal{M}_S^*) \leq g \cdot O(\alpha)$.

There can be at most n matched pairs in \mathcal{M}_S^* . Following (5), we derive $n \cdot \Delta_\alpha \leq g \cdot O(\alpha)$, implying

$$\Delta_\alpha \leq O\left(\frac{g\alpha}{n}\right).$$

On the other hand, we require $C_S(\mathcal{M}_S^*) \leq g \cdot O(\alpha)$ in our analysis; in particular, we need to replace the $2\sqrt{n}$ in (3) with $g \cdot O(\alpha)$. We derived $2\sqrt{n}$ as $2\frac{g}{\Delta_\alpha}$. We now need $2\frac{g}{\Delta_\alpha} \leq g \cdot O(\alpha)$, implying

$$\Delta_\alpha \geq \Omega\left(\frac{1}{\alpha}\right).$$

This is fine for $\alpha = \sqrt{n}$ or for large values of g . But for small α and small g , we cannot have both inequalities be true. Therefore, we do grid-snapping that let us ignore the second inequality.

3.2 $O(\alpha)$ -algorithm based on grid-snapping

Grid-snapping

Instead of grouping points into different cells as the $O(\sqrt{n})$ -approximation algorithm, we snap points to the lower left corners of their respective grid cells. Let $P' = \langle p'_1, \dots, p'_n \rangle$, $Q' = \langle q'_1, \dots, q'_n \rangle$ be the sequences after grid-snapping. We immediately obtain the following observations:

► **Observation 1.** If p_i and q_j are in the same cell, $\text{dist}(p'_i, q'_j) = 0$, and $\text{dist}(p_i, q_j) \leq \sqrt{2}\Delta < 2\sqrt{2}\Delta$.

► **Observation 2.** If p_i and q_j are in different cells, $\Delta \leq \text{dist}(p'_i, q'_j) \leq \text{dist}(p_i, q_j) + 2\sqrt{2}\Delta$.

We can then obtain our $O(\alpha)$ -approximation algorithm by altering the bound in the outer loop and the test procedure of Algorithm 1. See Algorithm 2. Here, $\text{AGED}(P', Q', k)$ attempts to Approximate $\text{GED}(P', Q')$ given that P' and Q' have their points on the corners of the grid cells. If $\text{GED}(P', Q') \leq k$, then it returns an $O(1)$ -approximate matching for the edit distance of the point sequences after grid-snapping. Otherwise, it either returns an $O(1)$ -approximate matching or it returns false.

■ **Algorithm 2** $O(\alpha)$ -approximation algorithm.

Input: Point sequences P and Q
Output: An approximately optimal matching for GED

```

1 if  $\sum_{i=1}^n \text{dist}(p_i, q_i) \leq 1$  then
2   | return matching  $\{(1, 1), \dots, (n, n)\}$ 
3 else
4   | for  $i := 0$  to  $\lceil \lg \frac{n}{\alpha} \rceil$  do
5     |    $g := 2^i$ 
6     |   for  $j := 1$  to  $\lceil c \lg n \rceil$  do
7       |   Obtain  $P', Q'$  by doing grid-snapping to  $P, Q$  based on a randomly
8         |   shifted grid
9         |    $out := \text{AGED}(P', Q', (12\sqrt{2} + \sqrt{2})g)$ 
10        |   if  $out \neq false$  then
11          |   | return out
12          |   end
13        |   end
14      |   end
15 end

```

We describe how to implement $\text{AGED}(P', Q', k)$ in Section 4.2. The running time of our implementation is $O(n + \frac{k^2}{\Delta})$ where Δ is the cell side length of the grid. We do grid snapping in $O(n)$ time. For each $g = 2^i$, we use cells of side length $\frac{g\alpha}{n}$ and set k to $(12\sqrt{2} + 2)g$, so the overall running time of our $O(\alpha)$ -approximation algorithm is

$$O(n) + \sum_{i=0}^{\lceil \lg \frac{n}{\alpha} \rceil} \sum_{j=1}^{\lceil c \lg n \rceil} O(n + \frac{2^i n}{\alpha}) = \sum_{i=0}^{\lceil \lg \frac{n}{\alpha} \rceil} O(n \log n + \frac{2^i n}{\alpha} \log n) = O(n \log^2 n + \frac{n^2}{\alpha^2} \log n).$$

The analysis for the $O(\alpha)$ -approximation algorithm is similar to the first algorithm. The major difference is that for any $g \geq \text{GED}(P, Q)$, if we compute the cost of the optimal matching for GED under the new point sequences, it will increase to only $(12\sqrt{2} + 2)g$ with constant probability despite our small choice for the grid cell side length. But as argued above, the small grid cell side length means the optimal matching of the point sequences after grid-snapping does not increase its cost much when returning the snapped points to their original positions. See Appendix A for details.

4 Constant Approximation Algorithm $AGED(P', Q', k)$

Recall that our constant factor approximation algorithm for GED of grid corner points is based on a known $O(n + k^2)$ time exact algorithm for string edit distance [15]. We first describe this exact algorithm for strings, which we refer as $SED(S, T, k)$, in Section 4.1. Then in Section 4.2, we modify this string algorithm to obtain an $O(1)$ -approximate matching for edit distance between point sequences P' and Q' assuming the points lie on the corners of grid cells and $GED(P', Q') \leq k$.

4.1 The exact $O(n + k^2)$ string edit distance algorithm

Dynamic programming matrix and its properties

Let $S = \langle s_1, s_2, \dots, s_n \rangle$ and $T = \langle t_1, t_2, \dots, t_n \rangle$ be two strings of length n . Let D denote a $(n + 1) \times (n + 1)$ matrix where $D(i, j)$ is the edit distance between substrings $S_i = \langle s_1, s_2, \dots, s_i \rangle$ and $T_j = \langle t_1, t_2, \dots, t_j \rangle$. We give a label h to every diagonal in this matrix such that for any entry (i, j) in this diagonal, $j = i + h$. See Fig. 1 (a).

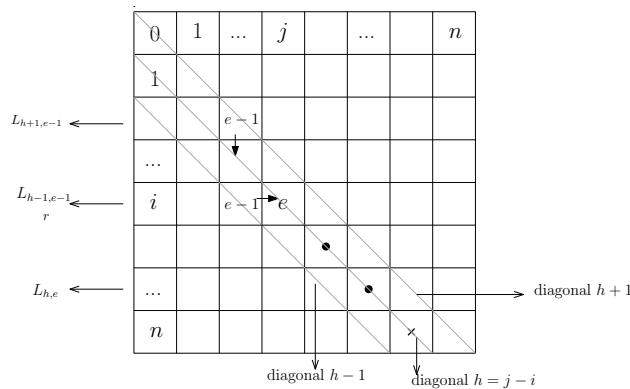


Figure 1 (a) The diagonal containing any entry $(i, i + h)$ is diagonal h . (b) The algorithm slides down the diagonal until finding an entry representing distinct characters. A circle means the corresponding two characters are the same; a cross means they are different.

Recall, we aim to minimize only the number of insertions and deletions to turn S into T . There are four important properties in this matrix which are used in the $O(n + k^2)$ time algorithm.

► Property 1.
$$D(i, j) = \min \begin{cases} D(i - 1, j) + 1 \\ D(i, j - 1) + 1 \\ D(i - 1, j - 1) + |s_i t_j| \end{cases} \quad \text{where } |s_i t_j| = \begin{cases} 0, & \text{if } s_i = t_j \\ \infty, & \text{otherwise} \end{cases} .$$

- Property 2. $D(i, 0) = i$, and $D(0, j) = j$.
- Property 3. $D(i, i + h)$ is even if and only if h is even.
- Property 4. $D(i, j) - D(i - 1, j - 1) \in \{0, 2\}$.

Property 4 can be easily derived from Property 3 and induction on $i + j$ (see Lemma 3 of [20]). From Property 4, we know all the diagonals are non-decreasing. In particular, all values on diagonal h are greater than $|h|$ considering Property 2. So, we can just search the band from diagonal $-k$ to k if the edit distance between S and T is at most k .

Algorithm for edit distance at most k

We use a greedy approach to fill the entries along each diagonal. For each value $e \in \{0, \dots, k\}$ (the outer loop), we locate the elements whose value is e by inspecting diagonals $-e$ to e (the inner loop). Finally, we return the best matching if $D(n, n)$ is covered by the above search. Otherwise, the edit distance is greater than k .

The key insight is that we can implicitly find all entries containing e efficiently in each round. We first define $L_{h,e}$ as the row index of the *farthest* e entry in diagonal h .

► **Definition 6.** $L_{h,e} = \max\{i \mid D(i, i+h) = e\}$.

Note by Property 3, $L_{h,e}$ is well-defined only if $h \equiv e \pmod{2}$. Observe that all values on diagonal h are at least $|h|$, which means that we can define our initial values as:

$$L_{h,h-2} = \begin{cases} |h| - 1, & \text{if } h < 0; \\ -1, & \text{otherwise} \end{cases}, \text{ where } h \in [-k, k].$$

Let $r = \max\{L_{h-1,e-1}, L_{h+1,e-1} + 1\}$. Then, $D(r, r+h) = e$ by Properties 1 and 4. Also, if $D(r, r+h) = e$ and $s_{r+1} = t_{r+1+h}$, then $D(r+1, r+1+h) = e$. From these observations, we can compute $L_{h,e}$ in each inner loop using Algorithm 3 below.

■ **Algorithm 3** Computing $L_{h,e}$ in each inner loop.

```

1  $r := \max\{L_{h-1,e-1}, L_{h+1,e-1} + 1\}$ 
2 while  $r + 1 \leq n, r + h + 1 \leq n$ , and  $s_{r+1} == t_{r+1+h}$  do
3   |  $r := r + 1$ ; /* slide */
4 end
5 if  $r > n$  or  $r + h > n$  then
6   |  $L_{h,e} := \infty$ 
7 else
8   |  $L_{h,e} := r$ 
9 end

```

We call lines 2 through 4 “the slide”. It is straightforward to recover the optimal matching by using the $L_{h,e}$ values to trace backwards through the dynamic programming matrix. Fig. 1 (b) demonstrates this process.

We can perform slides in constant time each after some $O(n)$ -time preprocessing at the beginning of the algorithm. In short, the length of a slide can be computed using a lowest common ancestor query in the suffix tree of a string based on S and T [15]. The overall running time is $O(n + k^2)$.

4.2 $O(1)$ -approximation algorithm by modifying the string version**Notation**

Similar to the string algorithm, we have a dynamic programming matrix; $D'(i, j)$ is the edit distance between subsequence $P'_i = \langle p'_1, \dots, p'_i \rangle$ and $Q'_j = \langle q'_1, \dots, q'_j \rangle$. This matrix also meets Property 1 stated earlier except that we use $\text{dist}(p'_i, q'_j)$ instead of $|s_i t_j|$. In addition, we also have the following property which is a refinement of Property 4.

► **Property 5.** $D'(i, j) - D'(i-1, j-1) \in [0, 2]$.

Clearly, the upper bound is 2 (just unmatch p_i and q_j). The lower bound can be proved by induction. Because the values in any diagonal are non-decreasing, we need only consider diagonals $-k$ through k .

(Implicit) label rules

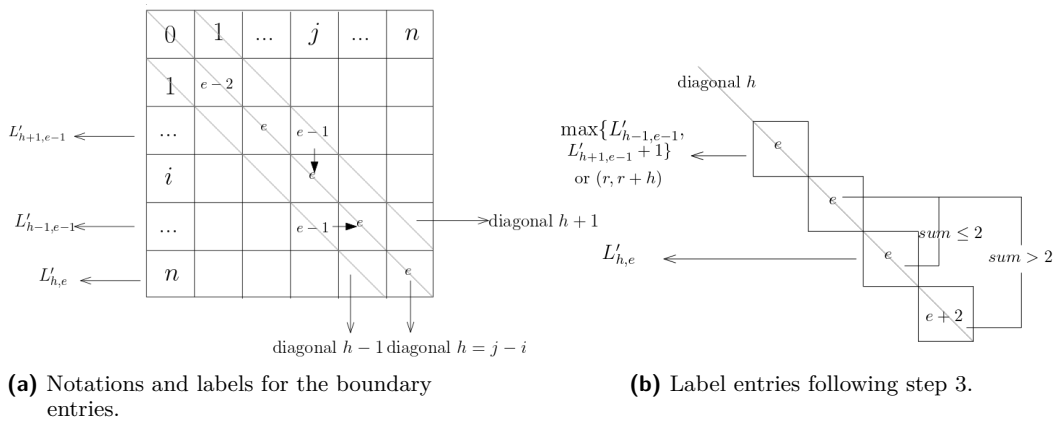
To obtain an approximate matching for the edit distance of snapped point sequences, we now label each entry in the dynamic programming matrix with an approximately tight lower bound on its value. Inspired by the string algorithm, we use non-negative integers for our labels, and the entries of any diagonal h only receive labels e where $e \equiv h \pmod 2$. Let $LA(i, j)$ be the label of entry (i, j) and $L'_{h,e}$ be the row index of the farthest entry whose label is e in diagonal h .

► **Definition 7.** $L'_{h,e} := \max\{i \mid LA(i, i+h) = e\}$.

For each e from 0 to k , for each diagonal h where $h \equiv e \pmod 2$, we (implicitly) assign labels e to each entry on diagonal h .

1. If $h = -e$ or e , i.e., this is the first iteration to assign labels to this diagonal, then we label the very beginning entry in diagonal h as e , i.e., if $h = -e$, $LA(|h|, 0) = e$; otherwise, $LA(0, h) = e$.
2. We define a *start entry* $(r, r+h)$ for each diagonal h . If $h = -e$ or e , r is the row index of the first one entry in diagonal h ; otherwise, $r = \max\{L'_{h-1,e-1}, L'_{h+1,e-1} + 1\}$.
3. We assign the label e to entries $(r, r+h)$ to $(r+s, r+h+s)$ where $\sum_{i=r+1}^s \text{dist}(p'_i, q'_{i+h}) \leq 2$ and $\sum_{i=r+1}^{s+1} \text{dist}(p'_i, q'_{i+h}) > 2$. $L'_{h,e} = r + s$. These entries correspond to a slide in the string algorithm.
4. Finally, if $(r-1, r+h-1)$ is unlabeled, we go backward up the diagonal labeling entries as e until we meet an entry that has been assigned a label previously. (Again, this step is implicit. As explained below, the actual algorithm only finds the $L'_{h,e}$ entries.)

Fig. 2 illustrates our rules.



■ **Figure 2** Notations and rules for approximating SGED.

Computing an approximately optimal matching

Assume we have set the initial values. Our algorithm only needs to compute each $L'_{h,e}$ as before. See Algorithm 4. Then, we guarantee the following theorem:

► **Theorem 8.** *We can recover a matching $M_{GS}^{*'} using all $L'_{h,e}$ from Algorithm 4. The cost of M_{GS}^{*}' for point sequences P', Q' is less and equal to $3GED(P', Q')$.$*

In short, we argue each label $LA(i, j) \leq D'(i, j)$. We then follow a path through the matrix as suggested by the way we pick labels in Algorithm 4. The final matching has cost at most $3LA(n, n)$ which is less and equal to $3GED(P', Q')$. The full proof appears in Appendix B.

■ **Algorithm 4** Computing $L'_{h,e}$ for the fixed h and e .

```

1  $r := \max\{(L'_{h-1,e-1}), (L'_{h+1,e-1} + 1)\}$ 
2  $sum := 0$ 
3 while  $r + 1 \leq n, r + h + 1 \leq n,$  and  $sum + dist(p'_{r+1}, q'_{r+h+1}) \leq 2$  do
4   |  $r := r + 1$ 
5   |  $sum := sum + dist(p'_r, q'_{r+h})$ 
6 end
7 if  $r > n$  or  $r + h > n$  then
8   |  $L'_{h,e} := \infty$ 
9 else
10  |  $L'_{h,e} := r$ 
11 end

```

We conclude by discussing the time complexity for our algorithm. Using the same $O(n)$ preprocessing as in [15], we can slide down maximal sequences of consecutive entries $(r, r + h)$ with $dist(p'_r, q'_{r+h}) = 0$ in constant time per slide. Let Δ be the cell side length of the grid whose cell corners contain points of P' and Q' . For $dist(p'_r, q'_{r+h}) \neq 0$, we know $dist(p'_r, q'_{r+h}) \geq \Delta$ from Observations 1 and 2. Therefore, we only need to manually add distances and restart faster portions of each slide of distances summing to 2 a total of $\frac{2}{\Delta}$ times. Thus, the total running time is

$$O(n + \sum_{e=0}^k \sum_{h=-e}^e \frac{1}{\Delta}) = O(n + \frac{k^2}{\Delta}).$$

References

- 1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *Proceedings of the IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 59–78, 2015.
- 2 Pankaj K Agarwal, Kyle Fox, Jiangwei Pan, and Rex Ying. Approximating dynamic time warping and edit distance for a pair of point sequences. In *Proceedings of the 32nd International Symposium on Computational Geometry*, pages 6:1–6:16, 2016.
- 3 Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *Proceedings of the IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 377–386, 2010.
- 4 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing*, pages 51–58, 2015.
- 5 Karl Bringmann. Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails. In *Proceedings of the IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 661–670, 2014.

- 6 Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *Proceedings of the IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 79–97, 2015.
- 7 Karl Bringmann and Wolfgang Mulzer. Approximability of the discrete Fréchet distance. *JoCG*, 7(2):46–76, 2016.
- 8 Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucky, and Michael Saks. Approximating edit distance within constant factor in truly sub-quadratic time. In *Proceedings of the 2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 979–990. IEEE, 2018.
- 9 Timothy M Chan and Zahed Rahmati. An improved approximation algorithm for the discrete Fréchet distance. *Information Processing Letters*, pages 72–74, 2018.
- 10 Lei Chen and Raymond Ng. On the marriage of Lp-norms and edit distance. In *Proceedings of the 30th International Conference on Very Large Databases*, pages 792–803, 2004.
- 11 Lei Chen, M Tamer Özsu, and Vincent Oria. Robust and fast similarity search for moving object trajectories. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, pages 491–502, 2005.
- 12 Omer Gold and Micha Sharir. Dynamic time warping and geometric edit distance: Breaking the quadratic barrier. *ACM Transactions on Algorithms*, 14(4):50, 2018.
- 13 Sarel Har-Peled. *Geometric approximation algorithms*, chapter 11, Random Partition via Shifting, pages 151–162. American Mathematical Soc., 2011.
- 14 William Kuszmaul. Dynamic Time Warping in Strongly Subquadratic Time: Algorithms for the Low-Distance Regime and Approximate Evaluation. In *Proceedings of the 46th International Colloquium on Automata, Languages and Programming*, 2019.
- 15 Gad M Landau, Eugene W Myers, and Jeanette P Schmidt. Incremental string comparison. *SIAM Journal on Computing*, 27(2):557–582, 1998.
- 16 Pierre-François Marteau. Time warp edit distance with stiffness adjustment for time series matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(2):306–318, 2009.
- 17 William J Masek and Michael S Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20(1):18–31, 1980.
- 18 Swaminathan Sankararaman, Pankaj K Agarwal, Thomas Mølhave, Jiangwei Pan, and Arnold P Boedihardjo. Model-driven matching and segmentation of trajectories. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 234–243, 2013.
- 19 Aleksandar Stojmirovic and Yi-kuo Yu. Geometric aspects of biological sequence comparison. *Journal of Computational Biology*, 16(4):579–611, 2009.
- 20 Esko Ukkonen. Algorithms for approximate string matching. *Information and Control*, 64(1-3):100–118, 1985.
- 21 Robert A Wagner and Michael J Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, 1974.
- 22 Xiaoyue Wang, Abdullah Mueen, Hui Ding, Goce Trajcevski, Peter Scheuermann, and Eamonn Keogh. Experimental comparison of representation methods and distance measures for time series data. *Data Mining and Knowledge Discovery*, 26(2):275–309, 2013.

A Analysis for $O(\alpha)$ -approximation algorithm

We introduce some additional notations to those used in Section 2.3.

Let $C_{GS}(\mathcal{M})$ be the cost of any monotone matching \mathcal{M} using distances between the grid-snapped points of P' and Q' . Let \mathcal{M}_{GS}^* be the optimal matching for P' and Q' , i.e., $C_{GS}(\mathcal{M}_{GS}^*) = GED(P', Q')$. Let \mathcal{M}'_{GS} be the matching returned by $AGED(P', Q', (12\sqrt{2} + 2)g)$.

We have the following lemma.

23:14 Approximating the Geometric Edit Distance

► **Lemma 9.** *If $GED(P, Q) \leq g$, with a probability at least $1 - \frac{1}{n^c}$, at least one of the $\lceil c \lg n \rceil$ iterations will return a matching \mathcal{M}_{GS}' .*

Proof. Similar to (2), and with Observations 1 and 2, we have

$$\begin{aligned} C_{GS}(\mathcal{M}_G^*) &= |UM_{\mathcal{M}_G^*}| + 0 \cdot (|OC_{\mathcal{M}_G^*}| + |OF_{\mathcal{M}_G^*}|) \\ &\quad + \sum_{(i,j) \in DC_{\mathcal{M}_G^*}} \text{dist}(p'_i, q'_j) + \sum_{(i,j) \in DF_{\mathcal{M}_G^*}} \text{dist}(p'_i, q'_j) \\ &\leq |UM_{\mathcal{M}_G^*}| + 2\sqrt{2}\Delta \cdot |DC_{\mathcal{M}_G^*}| + \sum_{(i,j) \in DF_{\mathcal{M}_G^*}} \left(\text{dist}(p_i, q_j) + 2\sqrt{2}\Delta \right). \\ &= |UM_{\mathcal{M}_G^*}| + \sum_{(i,j) \in DF_{\mathcal{M}_G^*}} \text{dist}(p_i, q_j) + 2\sqrt{2}\Delta (|DC_{\mathcal{M}_G^*}| + |DF_{\mathcal{M}_G^*}|) \end{aligned}$$

If $C_G(\mathcal{M}_G^*) \leq g$, then

$$C_{GS}(\mathcal{M}_{GS}^*) \leq C_{GS}(\mathcal{M}_G^*) \leq g + 2\sqrt{2}\Delta \cdot (|DC_{\mathcal{M}_G^*}| + |DF_{\mathcal{M}_G^*}|).$$

We have the same observation for $DF_{\mathcal{M}_G^*}$ as before, that is there are at most $\frac{g}{\Delta}$ pairs in $DF_{\mathcal{M}_G^*}$. Using the same algebra as (4), we have $E(|DC_{\mathcal{M}_G^*}|) \leq \frac{2g}{\Delta}$. So,

$$E(C_{GS}(\mathcal{M}_{GS}^*)) \leq g + 2\sqrt{2}\Delta \cdot \left(\frac{g}{\Delta} + \frac{2g}{\Delta} \right) = 6\sqrt{2}g + g.$$

According to Markov's inequality, we know

$$P\left(C_{GS}(\mathcal{M}_{GS}^*) \geq (12\sqrt{2} + 2)g\right) \leq \frac{1}{2}.$$

In Section 4.2, we prove that if $C_{GS}(\mathcal{M}_{GS}^*) = GED(P', Q') \leq (12\sqrt{2} + 2)g$, then $AGED(P', Q', (12\sqrt{2} + 2)g)$ will return a constant approximate matching \mathcal{M}_{GS}' . So, if we test $AGED(P', Q', (12\sqrt{2} + 2)g)$ $\lceil c \lg n \rceil$ times (using different grids each time), with a probability at least $1 - \frac{1}{n^c}$, at least one $AGED(P', Q', (12\sqrt{2} + 2)g)$ will return a matching \mathcal{M}_{GS}' . We conclude the proof of Lemma 9. ◀

Finally, from Observation 2, for every pair (i, j) in \mathcal{M}_{GS}^* , we have $\text{dist}(p_i, q_j) \leq \text{dist}(p'_i, q'_j) + 2\sqrt{2}\Delta$. We can now return points to their original positions:

$$\begin{aligned} C_G(\mathcal{M}_{GS}') &= |UM_{\mathcal{M}_{GS}'}| + \sum_{(i,j) \in DC_{\mathcal{M}_{GS}'}} \text{dist}(p_i, q_j) + \sum_{(i,j) \in DF_{\mathcal{M}_{GS}'}} \text{dist}(p_i, q_j) \\ &\quad + \sum_{(i,j) \in OC_{\mathcal{M}_{GS}'}} \text{dist}(p_i, q_j) + \sum_{(i,j) \in OF_{\mathcal{M}_{GS}'}} \text{dist}(p_i, q_j) \\ &\leq |UM_{\mathcal{M}_{GS}'}| + \sum_{(i,j) \in DC_{\mathcal{M}_{GS}'}} \text{dist}(p'_i, q'_j) + \sum_{(i,j) \in DF_{\mathcal{M}_{GS}'}} \text{dist}(p'_i, q'_j) + \sum_{(i,j) \in OC_{\mathcal{M}_{GS}'}} \text{dist}(p'_i, q'_j) \\ &\quad + \sum_{(i,j) \in OF_{\mathcal{M}_{GS}'}} \text{dist}(p'_i, q'_j) + 2\sqrt{2}\Delta \left(|DC_{\mathcal{M}_{GS}'}| + |DF_{\mathcal{M}_{GS}'}| + |OC_{\mathcal{M}_{GS}'}| + |OF_{\mathcal{M}_{GS}'}| \right) \\ &\leq O(1) \cdot (12\sqrt{2} + 2)g + n \cdot 2\sqrt{2}\Delta. \end{aligned}$$

Recall, $\Delta = \frac{g\alpha}{n}$. If we obtain a matching \mathcal{M}_{GS}' during an iteration where $g \geq C_G(\mathcal{M}_G^*) = GED(P, Q) \geq \frac{1}{2}g$, then $C_G(\mathcal{M}_{GS}') \leq O(g\alpha) = O(\alpha) \cdot GED(P, Q)$. Using the same argument as in Theorem 2, we conclude our proof of Theorem 3.

B Proof of Theorem 8

We have the following properties for our labels and the following lemma.

- ▶ Property 6. $LA(i, i + h) - LA(i + 1, i + 1 + h) \in \{0, 2\}$.
- ▶ Property 7. $LA(i, i+h) - LA(i-1, i+h) \in \{-1, 1\}$ and $LA(i, i+h) - LA(i, i+h-1) \in \{-1, 1\}$.
- ▶ **Lemma 10.** For every entry (i, j) , $LA(i, j) \leq D'(i, j)$.

Note that in particular, $LA(n, n) \leq GED(P', Q')$.

Proof. From Property 5, we only need to prove e is the lower bound of the first entry whose label is e in each diagonal h .

We proceed by induction on e .

1. If $e = 0$, we only label the first entry in diagonal 0 as 0. We have $0 \leq D'(0, 0) = 0$. If $e = 1$, then for diagonals 1 and -1 , we have $1 \leq D'(0, 1) = D'(1, 0) = 1$.
2. Assume Lemma 10 for labels less than e . For e , we consider the diagonals $h = -e$ to e :
 If $h = -e$ or e , we know $e \leq D'(|h|, 0) = e$ or $e \leq D'(0, h) = e$.
 Otherwise, let $(f, f+h)$ be the first entry whose label is e . From Property 6, $f = L'_{h, e-2} + 1$. Fig. 3 shows the notations. From the refined Property 1, we need to discuss three cases:

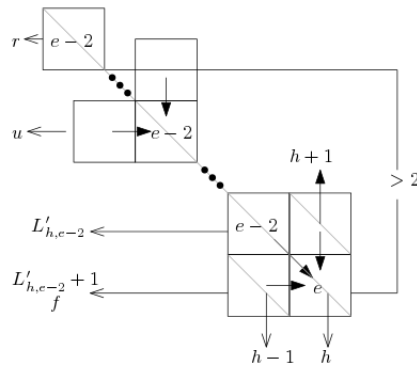


Figure 3 We compute the lower bound of entries which are labeled as e .

- a. $D'(f, f + h) = D'(f - 1, f + h) + 1$.
 From Property 7, we know $LA(f - 1, f + h) = e - 1$ or $e + 1$.
 - If $LA(f - 1, f + h) = e - 1$, $D'(f - 1, f + h) \geq e - 1$ from our assumption. So, $D'(f, f + h) = D'(f - 1, f + h) + 1 \geq e - 1 + 1 = e$.
 - If $LA(f - 1, f + h) = e + 1$, then we know $L'_{h+1, e-1}$ is less than $f - 1$. From non-decreasing property, $e - 1 \leq D'(L'_{h+1, e-1}, L'_{h+1, e-1} + h + 1) \leq D'(f - 1, f + h - 1)$.
- b. $D'(f, f + h) = D'(f, f + h - 1) + 1$.
 This case is similar to the above.
- c. $D'(f, f + h) = D'(f - 1, f + h - 1) + dist(p'_f, q'_{f+h})$.
 $LA(f - 1, f + h - 1) = e - 2$, because $f - 1 = L'_{h, e-2}$. Let r be the row index of the first entry to slide with label $e - 2$ in diagonal h , i.e., $r = \max\{L'_{h-1, e-3}, L'_{h+1, e-3} + 1\}$. See Fig. 3. We define u as the row index of the first entry walking backward from entry $(f, f + h)$ along the diagonal h where $D'(u, u + h) = \min\{D'(u, u + h - 1), D'(u - 1, u + h - 1)\} + 1$.

23:16 Approximating the Geometric Edit Distance

- If $u > r$, like Fig. 3, then $u > L'_{h-1,e-3}$ and $u - 1 > L'_{h+1,e-3}$. Combining our assumption, we have

$$D'(u, u + h - 1) \geq D'(L'_{h-1,e-3} + 1, L'_{h-1,e-3} + h) \geq e - 1$$

and

$$D'(u - 1, u + h) \geq D'(L'_{h+1,e-3} + 1, L'_{h+1,e-3} + h + 2) \geq e - 1.$$

So,

$$D'(u, u + h) = \min\{D'(u, u + h - 1), D'(u - 1, u + h - 1)\} + 1 \geq e - 1 + 1$$

implying $D'(u, u + h) \geq e$. Recall $f \geq u$, so $D'(f, f + h) \geq e$.

- If $u \leq r$, then

$$\begin{aligned} D'(f, f + h) &= D'(r, r + h) + \sum_{i=r+1}^f \text{dist}(p'_i, q'_{i+h}) \\ &> e - 2 + 2 = e. \end{aligned}$$

Examining all cases, we conclude the proof of Lemma 10. ◀

The bounds for the approximate matching $C_{GS}(\mathcal{M}_{GS}^*)$

From Algorithm 4, we note the label increases correspond to not matching a point in Line 1, and slides correspond to matching points. Let \mathcal{M}_{GS}^* be the resulting matching. So,

$$\begin{aligned} C_{GS}(\mathcal{M}_{GS}^*) &= |UM_{\mathcal{M}_{GS}^*}| + \sum_{(i,j) \in \mathcal{M}_{GS}^*} \text{dist}(p'_i, q'_j) \\ &\leq LA(n, n) + 2 \cdot LA(n, n) \leq 3LA(n, n) \leq 3GED(P', Q'). \end{aligned}$$

We conclude the proof of Theorem 8 and obtain an $O(1)$ -approximation algorithm for $GED(P', Q')$.

On Adaptivity Gaps of Influence Maximization Under the Independent Cascade Model with Full-Adoption Feedback

Wei Chen

Microsoft Research, Beijing, China
weic@microsoft.com

Binghui Peng¹

Columbia University, New York, United States
bp2601@columbia.edu

Abstract

In this paper, we study the *adaptivity gap* of the influence maximization problem under the independent cascade model when *full-adoption* feedback is available. Our main results are to derive upper bounds on several families of well-studied influence graphs, including in-arborescences, out-arborescences and bipartite graphs. Especially, we prove that the adaptivity gap for the in-arborescences is between $[\frac{e}{e-1}, \frac{2e}{e-1}]$, and for the out-arborescences the gap is between $[\frac{e}{e-1}, 2]$. These are the first constant upper bounds in the full-adoption feedback model. Our analysis provides several novel ideas to tackle the correlated feedback appearing in adaptive stochastic optimization, which may be of independent interest.

2012 ACM Subject Classification Theory of computation → Social networks

Keywords and phrases Adaptive influence maximization, adaptivity gap, full-adoption feedback

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.24

1 Introduction

Following the celebrated work of Kempe et al. [19], the *influence maximization* (IM) problem has been extensively studied over past decades. Influence maximization is the problem of selecting at most k *seed nodes* that maximize the influence spread on a given social network and diffusion model. Influence maximization has many real world applications such as viral markets, rumor controls, etc. In the past years, the IM problem has been studied in different context such as outbreak detection [20], topic-aware influence propagation [5], competitive and complementary influence maximization [23] etc., and both theoretically and practically efficient algorithms have been developed [13, 12, 7, 32, 31]. See the recent survey [11, 21] for more detailed reference.

Meanwhile, stimulated by the real life demand, researchers in recent years begin to consider this classical problem in the adaptive setting. In the *adaptive influence maximization* problem, instead of selecting the full seed set all at once, we are allowed to select seeds one after another, making future decisions based on the propagation feedback gathered from the previous seeds selected. Two feedback models are typically considered [15]: *myopic feedback*, where only the one-step propagation from the selected seed to its immediate out-neighbors are included in the feedback, and *full-adoption feedback*, where the entire cascade from the seed is included in the feedback. This adaptive decision process can potentially bring huge benefits but it also brings technical challenges, since adaptive policies are usually hard to

¹ Work is mostly done while Binghui was at Tsinghua University and visiting Microsoft Research Asia as an intern.



© Wei Chen and Binghui Peng;

licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 24; pp. 24:1–24:19

Leibniz International Proceedings in Informatics



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

design and analyze, and the adaptive decision process can be slow in practice. Thus, a crucial task in this area is to decide whether and how much adaptive policy is really superior over the non-adaptive policy. The *adaptivity gap* quantifies to what extent adaptive policy outperforms a non-adaptive one and it is defined as the supremum ratio between the influence spread of the optimal adaptive policy and that of the optimal non-adaptive policy. The above question has been answered recently when only myopic feedback are available [25, 14] and constant upper bounds on the adaptivity gap have been derived.

In this paper, we consider the influence maximization problem in the independent cascade (IC) model with *full-adoption feedback*. Even though the full-adoption feedback under the IC model satisfies an important property called *adaptive submodularity*, the analysis of its adaptivity gap is more challenging because the feedback obtained from different seed nodes are no longer independent – feedback from one seed contains multiple-step cascade results, and thus it may overlap with the feedback from other seed nodes. Therefore, results from existing studies on the adaptivity gap of general classes of stochastic adaptive optimization problems [3, 16, 17, 8] cannot be applied, since they all rely on the independent feedback assumption.

In this study, we are able to derive nontrivial constant upper bounds on several families of graphs, including in-arborescences, out-arborescence and bipartite graphs, which have been the targets of many studies in influence maximization (see Section 1.1 for more details). Formally, we have (i) when the influence graph is an in-arborescence, the adaptivity gap is between $[\frac{e}{e-1}, \frac{2e}{e-1}]$ (Section 3 and Section 6), (ii) when the influence graph is an out-arborescence, the adaptivity gap is between $[\frac{e}{e-1}, 2]$ (Section 4 and Section 6) and (iii) the adaptivity gap for the bipartite influence graph is $\frac{e}{e-1}$ (Section 5). Our upper bounds on arborescences are the first constant upper bounds in the full-adoption feedback model with multiple steps of cascades and thus dependent feedback, and our upper bound on bipartite graphs improves the results in [14, 18].

The main technical contributions in this paper are on the adaptivity gaps for arborescences, in which the feedback information can be correlated and all previous methods failed. We adopt two different proof strategies to overcome the difficulty of dependent feedback. For in-arborescences, we follow the framework in [3] and construct a Poisson process to relate the influence spread of the optimal adaptive policy and the *multilinear extension*. The analyses are non-trivial due to the correlated feedback. We need to delicately decompose the marginal gain of the Poisson process and give upper bounds on each terms. The key observation we have for in-arborescences is that the *boundary* of the active nodes shrinks during the diffusion process. For out-arborescences, we again relate the influence spread of the multilinear extension to the optimal policy, but using a completely different proof strategy. The key observation for out-arborescences is that the predecessors of each node form a directed line thus proving a stronger results on this line is sufficient. We derive a family of constraints on the optimal adaptive policy and telescope the marginal gains of the multilinear extension, combining these two could yield our results.

1.1 Related Work

A number of studies [6, 36, 35, 24, 22] have focused on the influence maximization problem on arborescences and interesting theoretical results have been found with this special structural assumption. Bharathi et al. [6] initiate the study on arborescences and derive a polynomial-time approximation scheme (PTAS) for bidirected trees. Wang et al. [36] convert diffusion in the IC model in the local region into the diffusion in in-arborescences to design efficient heuristic algorithms for influence maximization. For in-arborescences, Wang et al. [35] give a polynomial time algorithm in the linear threshold (LT) model and Lu et al. [24] prove NP hardness results under the independent cascade model.

The influence maximization problem on one-directional bipartite graphs has been studied by [2, 29, 18], and it has applications in advertisement selections. Especially, Hatano et al. [18] consider the problem in the adaptive setting and derive adaptive algorithms with theoretical guarantees.

Initiated by the pioneering work of [15], a recent line of work [33, 37, 26, 30, 14, 25] focus on the adaptive influence maximization problem and develop both theoretical results and practical methods. Golovn and Krause [15] propose the novel concept of adaptive submodularity and apply it to the adaptive influence maximization problem. They prove that with full-adoption feedback in the IC model, the influence spread function satisfies the adaptive submodularity, thus a simple adaptive greedy algorithm could achieve the $(1 - 1/e)$ approximation ratio. Fujii et al. [14] generalize the notion and propose weakly adaptive submodularity. They consider the adaptivity gap on both LT and IC models, when the influence graph is one-directional bipartite, which means that the diffusion is always done in one step and there is no difference between myopic feedback and full-adoption feedback. While they prove a tight upper bound of 2 for the LT model, their bound for IC model depends on the structure of the graph and can be far worse than $1 - 1/e$. In contrast, in this paper we provide the tight bound of $1 - 1/e$ with a simple analysis in this case. Recently, Peng and Chen [25] consider the myopic feedback model and prove an upper bound of 4 for the adaptivity gap in the IC model. Singer and his collaborators have done a series of studies on adaptive seeding and studied the adaptivity gap in their setting [27, 28, 4], but their model is a two-step adaptive model with the first step purely for referring to the seed candidates, and thus their model is very different from adaptive influence maximization of this paper and other related work above.

From the theoretical side, there are two lines of works [3, 1, 16, 17, 8] on the adaptivity gap that are most relevant to ours. Asadpour et al. [3] study the stochastic submodular optimization problem. They use multilinear extensions to transform an adaptive strategy to a non-adaptive strategy and give a tight upper bound of $\frac{e}{e-1}$. Their method inspires our work but theirs cannot be directly applied to our settings, since the feedback information is not independent in the full-adoption feedback model. We defer further discussion about this key difference to Section 3. Another line of work [16, 17, 8] focuses on the stochastic probing problem. They transform any adaptive policy to a random walk non-adaptive policy and Bradac et al. [8] finally prove a tight upper bound of 2 for prefix constraints.

2 Preliminaries

In this paper, we focus on the well known *independent cascade* (IC) model as the diffusion model. In the IC model, the social network is described by a directed influence graph $G = (V, E, p)$ ($|V| = n$), and there is a probability p_{uv} associated with each edge $(u, v) \in E$. The *live-edge* graph $L = (V, L(E))$ is a random subgraph of the influence graph G , where each edge $(u, v) \in E$ appears in $L(E)$ independently with probability p_{uv} . If the edge appears in $L(E)$, we say it is *live*, otherwise we say it is *blocked*. We use \mathcal{L} to denote all possible live-edge graphs and \mathcal{P} to denote the probability distribution over \mathcal{L} . The diffusion process can be described by the following discrete time process. At time $t = 0$, a seed set $S \subseteq V$ is activated and a live-edge graph L is sampled from the probability distribution \mathcal{P} (i.e., each edge (u, v) is live with probability p_{uv}). At time $t = 1, 2, \dots$, a node $u \in V$ is active if (i) u is active at time $t - 1$ or (ii) one of u 's in-neighbor in the live-edge graph L is active at time $t - 1$. The diffusion process ends at a step when there are no new nodes activated at this step. We use $\Gamma(S, L)$ to denote the set of active nodes at the end of diffusion, or equivalently,

24:4 On Adaptivity Gaps of Influence Maximization

the set of nodes reachable from set S under live-edge graph L . We define the *influence reach function* $f : \{0, 1\}^V \times \mathcal{L} \rightarrow \mathbb{R}^+$ as $f(S, L) := |\Gamma(S, L)|$. Then the influence spread of a set S , denoted as $\sigma(S)$, is defined as the expected number of active nodes at the end of the diffusion process, i.e., $\sigma(S) := \mathbb{E}_{L \sim \mathcal{P}}[f(S, L)]$.

We formally state the (non-adaptive) influence maximization problem as follows.

► **Definition 2.1** (Non-adaptive influence maximization). *The non-adaptive influence maximization (IM) problem is the problem of given an influence graph $G = (V, E, p)$ and a budget k , finding a seed set S^* of size at most k that maximizes the influence spread, i.e., finding $S^* = \operatorname{argmax}_{S \subseteq V, |S| \leq k} \sigma(S)$.*

In the adaptive setting, instead of committing the entire seed set all at once, we are allowed to select the seed node one by one. After we select a seed, we can get some feedback about the diffusion state from the node. Formally, a *realization* ϕ is a function $\phi : V \rightarrow O$, mapping a node u to its state, i.e., the feedback we obtain when we select the node u as a seed. The realization ϕ determines the status of all edges in the influence graph and it is one-to-one correspondence to a live-edge graph. Henceforth, in the rest of the paper, we would use ϕ to refer to both the realization and the live-edge graph interchangeably. The feedback information depends on the feedback model and in this paper, we consider the *full-adoption* feedback model. In the full-adoption feedback model, after we select a node u , the feedback $\phi(u)$ of u we see is the status of all out-going edges of nodes v that are reachable from u in the live-edge graph corresponding to ϕ . In other words, we get to see the full cascade starting from the node u . At each step of the adaptive seeding process, our observation so far is represented by a *partial realization* $\psi \subseteq V \times O$, which is a collection of nodes and states, $(u, \phi(u))$, we have observed so far. We use $\operatorname{dom}(\psi)$ to denote the set $\{u : u \in V, \exists (u, o) \in \psi\}$, that is, all nodes we have selected so far. For two partial realizations ψ and ψ' , we say ψ is a sub-realization of ψ' if $\psi \subseteq \psi'$ when treating ψ and ψ' as subsets of $V \times O$.

An adaptive policy π is a mapping from partial realizations to nodes. Given a partial realization ψ , we use $\pi(\psi)$ to represent the next seed selected by π . After selecting node $\pi(\psi)$, our observation (partial realization) grows as $\psi' = \psi \cup (\pi(\psi), \phi(\pi(\psi)))$ and the policy π would pick the next node based on the new partial realization ψ' . Given a realization ϕ , we use $V(\pi, \phi)$ to denote the seed set selected by the policy π . The *adaptive influence spread* of the policy π is defined as the expected number of active nodes under the policy π , i.e., $\sigma(\pi) := \mathbb{E}_{\Phi \sim \mathcal{P}}[f(V(\pi, \Phi), \Phi)]$. We define $\Pi(k)$ as the set of policies π , such that for any possible realization ϕ , $|V(\pi, \phi)| \leq k$. For convenience, we treat the adaptive policy π as deterministic, but our results apply to randomized policies too. The adaptive influence maximization problem is formally stated as follow.

► **Definition 2.2** (Adaptive influence maximization). *The adaptive influence maximization (AIM) problem is the problem of given an influence graph $G = (V, E, p)$ and a budget k , finding a feasible policy $\pi \in \Pi(k)$ that maximizes the adaptive influence spread, i.e., finding $\pi^* = \operatorname{argmax}_{\pi \in \Pi(k)} \sigma(\pi)$.*

In this paper, we study the *adaptivity gap* of the influence maximization problem under the full-adoption feedback model. The adaptivity gap measures the supremacy of the optimal adaptive policy over the optimal non-adaptive policy. We use $\operatorname{OPT}_N(G, k)$ (resp. $\operatorname{OPT}_A(G, k)$) to denote the influence spread of the optimal non-adaptive (resp. adaptive) policy for the IM problem on the influence graph G with a budget k .

► **Definition 2.3** (Adaptivity gap). *The adaptivity gap for the IM problem in the IC model with full-adoption feedback is defined as*

$$\sup_{G,k} \frac{\text{OPT}_A(G,k)}{\text{OPT}_N(G,k)}. \quad (1)$$

We prove constant upper bounds on the adaptivity gap for several classes of graphs, including the *in-arborescence* and the *out-arborescence*.

► **Definition 2.4** (In-arborescence). *We say an influence graph $G = (V, E, p)$ is an in-arborescence when the underlying graph is a directed tree with a root u , such that for any node $v \in V$, the unique path between nodes u and v is directed from v to u . In other words, the information propagates from leaves to the root.*

► **Definition 2.5** (Out-arborescence). *We say an influence graph $G = (V, E, p)$ is an out-arborescence when the underlying graph is a directed tree with a root u , such that for any node $v \in V$, the unique path between nodes u and v is directed from u to v . In other words, the information propagates from the root to leaves.*

A set function $f : V \rightarrow \mathbb{R}^+$ is said to be submodular if for any set $A \subseteq B \subseteq V$ and any element $u \in V \setminus B$, $\Delta_f(u|A) := f(A \cup \{u\}) - f(A) \geq f(B \cup \{u\}) - f(B) = \Delta_f(u|B)$. We call $\Delta_f(u|A)$ the marginal gain for adding element u to the set A . Moreover, the function f is said to be monotone if $f(B) \geq f(A)$. Under the IC model, the influence spread function $\sigma(\cdot)$ is proved to be *submodular* and *monotone* [19], thus given value oracles for $\sigma(\cdot)$, the greedy algorithm is $1 - 1/e$ approximate to the optimal non-adaptive solution.

In the adaptive submodular optimization scenario, a similar notion to the submodularity is called the *adaptive submodularity*. For a function f and a partial realization ψ , the conditional marginal gain for an element $u \notin \text{dom}(\psi)$ is defined as $\Delta_{f,\mathcal{P}}(u|\psi) := \mathbb{E}_{\Phi \sim \mathcal{P}} [f(\text{dom}(\psi) \cup \{u\}) - f(\text{dom}(\psi)) | \Phi \sim \psi]$, where we write $\Phi \sim \psi$ to say that the realization Φ is *consistent* with the partial realization ψ , i.e., $\Phi(u) = \psi(u)$ for every $u \in \text{dom}(\psi)$. A function f is said to be *adaptive submodular* with respect to \mathcal{P} if for any partial realizations $\psi \subseteq \psi'$ and any element $u \in V \setminus \text{dom}(\psi')$, $\Delta_{f,\mathcal{P}}(u|\psi) \geq \Delta_{f,\mathcal{P}}(u|\psi')$. Moreover, the function f is *adaptive monotone* with respect to \mathcal{P} if $\Delta_{f,\mathcal{P}}(u|\psi) \geq 0$ for any feasible partial realization ψ with $\Pr_{\Phi \sim \mathcal{P}}\{\Phi \sim \psi\} > 0$. Golovin and Krause [15] show the following important result, which will be used in our analysis.

► **Proposition 2.6** ([15]). *The influence reach function f is adaptive submodular and adaptive monotone with respect to the live-edge graph distribution \mathcal{P} under the independent cascade model with full-adoption feedback.*

The following two definitions are very important to our later analysis.

► **Definition 2.7** (Multilinear extension). *The multilinear extension $F : [0, 1]^V \rightarrow \mathbb{R}^+$ of the influence spread function σ is defined as*

$$F(x_1, \dots, x_n) = \sum_{S \subseteq V} \left[\left(\prod_{i \in S} x_i \prod_{i \notin S} (1 - x_i) \right) \sigma(S) \right]. \quad (2)$$

We remark that the multilinear extension $F(\cdot)$ is monotone and DR-submodular [19], when the original function $\sigma(\cdot)$ is monotone and submodular. A vector function f is DR-submodular if for any two vectors $(x_1, \dots, x_n) \leq (y_1, \dots, y_n)$ (coordinate-wise), for any $\delta > 0$, any $j \in [n]$, $f(x_1, \dots, x_j + \delta, \dots, x_n) - f(x_1, \dots, x_n) \geq f(y_1, \dots, y_j + \delta, \dots, y_n) - f(y_1, \dots, y_n)$. For any configuration (x_1, \dots, x_n) , we use $f^+(x_1, \dots, x_n)$ to denote the adaptive influence spread of the optimal adaptive strategy consistent with this configuration. Formally,

► **Definition 2.8** (Adaptive influence spread function based on an optimal adaptive policy). We define $f^+ : [0, 1]^{|V|} \rightarrow \mathbb{R}^+$ as:

$$f^+(x_1, \dots, x_n) = \sup_{\pi} \left\{ \sigma(\pi) : \Pr_{\Phi \sim \mathcal{P}} [i \in V(\pi, \Phi)] = x_i, \forall i \in [n] \right\}. \quad (3)$$

3 Adaptivity Gap for In-arborescence

In this section, we give an upper bound on the adaptivity gap when the influence graph is an in-arborescence, as stated in the following theorem.

► **Theorem 3.1.** *When the underlying influence graph is an in-arborescence, the adaptivity gap for the IM problem in the IC model with full-adoption feedback is at most $\frac{2e}{e-1}$.*

Our approach follows the general framework of [3], i.e., we use the multilinear extension to transform an adaptive policy to a non-adaptive policy, and construct a Poisson process to connect the influence spread of the non-adaptive policy to the adaptive policy. Once we have done this, combining with the rounding procedure in [9, 10], we can derive an upper bound on the adaptive policy. However, remembering that the main difficulty of our problem comes from the correlation of the feedback, directly applying the analyses in [3] does not work. Our methods have several key differences comparing to [3]. To be more specific, we can no longer directly relate the dynamic marginal gain of the Poisson process to the influence spread of the adaptive policy. Instead, we need to delicately decompose the marginal gain into two parts (see Lemma 3.3 and Lemma 3.9). The first part can be related to the optimal adaptive strategy by telescoping the summation and applying a coupling argument, while the second part can be related to a (randomized) non-adaptive policy. However, this non-adaptive policy is not guaranteed to be bounded by $\text{OPT}_N(G, k)$ (the optimal non-adaptive policy of budget k), because the size of the seed set is random and can potentially be very large. We utilize the “*weak concavity*” of the optimal solution to show that it is enough to consider the expected size of the (random) seed set, and then we give an upper bound on this expected size for an in-arborescence. This upper bound relies on a crucial property of the in-arborescence, i.e., the *boundary* (see Definition 3.7) size always shrinks during the diffusion process of the information (see Lemma 3.8). Putting things together, we get a differential inequation that relates the dynamic marginal gain of the Poisson process to both an optimal adaptive policy and the optimal non-adaptive policy. Solving the differential inequation yields a lower bound on the multilinear extension and it gives an upper bound on the adaptivity gap. We remark that one noticeable difference of our bound on the adaptivity gap is that it does not hold for the matroid constraint (which holds in [3]), even though the multilinear extension was originally designated to handle matroid constraints.

Following the work [3, 34], for any configuration (x_1, \dots, x_n) , we consider the following Poisson process, which indirectly relates the multilinear extension $F(x_1, \dots, x_n)$ to the optimal adaptive solution $f^+(x_1, \dots, x_n)$.

Poisson Process. There are n independent Poisson clocks C_1, \dots, C_n , the clock C_i ($i \in [n]$) sends signals with rate x_i . Whenever a clock C_i sends out a signal, we select node i as a seed and gather feedback $\phi(i)$ according to the underlying realization ϕ . We use $\Psi(t)$ to denote the partial realization at time t and we start with $\Psi(0)$ as \emptyset . We note that $\Psi(t)$ is a random partial realization that contains (a) random time points $t_i \leq t$ at which clock C_i sends a signal, for $i \in [n]$; and (b) for each $t_i \leq t$, the feedback $\phi(i)$ of seed node i based on

the live-edge graph corresponding to realization ϕ . The Poisson process ends at $t = 1$. Note that the Poisson process is parameterized by (x_1, x_2, \dots, x_n) , but we ignore these parameters in the notation $\Psi(t)$.

Given a partial realization ψ , we slightly abuse the notation and use $\Gamma(\psi)$ to denote all the nodes that seed nodes $\text{dom}(\psi)$ activate in the diffusion. Since we are considering the full-adoption feedback, all nodes activated from $\text{dom}(\psi)$ are included in the partial realization ψ , and thus $\Gamma(\psi)$ is a fixed node set when ψ is fixed. We define $f(\psi) = |\Gamma(\psi)|$. The following lemma states that at the end of the Poisson process, i.e., when $t = 1$, the expected influence spread $\mathbb{E}[f(\Psi(1))]$ is no greater than the influence spread of $F(x_1, \dots, x_n)$, so it links the Poisson process to the non-adaptive optimal solution.

► **Lemma 3.2.** $\mathbb{E}[f(\Psi(1))] = F(1 - e^{-x_1}, \dots, 1 - e^{-x_n}) \leq F(x_1, \dots, x_n)$.

Proof. Notice that in the Poisson process, the selection of seeds are actually independent of the realization of the influence graph. Moreover, seed nodes are selected independently. At the end of the process (when $t = 1$), the node i is selected as a seed with probability $1 - e^{-x_i}$. Thus we have

$$\begin{aligned} \mathbb{E}[f(\Psi(1))] &= \sum_{S \subseteq V} \left[\left(\prod_{i \in S} (1 - e^{-x_i}) \right) \prod_{i \notin S} (e^{-x_i}) \sigma(S) \right] \\ &= F(1 - e^{-x_1}, \dots, 1 - e^{-x_n}) \leq F(x_1, \dots, x_n). \end{aligned} \quad (4)$$

The inequality holds due to the monotonicity of the multilinear extension $F(\cdot)$ and the fact that $1 - e^{-x} \leq x$. ◀

In the following lemma, we give a lower bound on the dynamic marginal gain of the influence spread in the Poisson process, which links the process to the optimal adaptive solution f^+ .

► **Lemma 3.3.** *For any $t \in [0, 1]$ and any fixed partial realization ψ , we have*

$$\mathbb{E} \left[\frac{df(\Psi(t))}{dt} \mid \Psi(t) = \psi \right] \geq f^+(x_1, \dots, x_n) - \sigma(\Gamma(\psi)). \quad (5)$$

Before proving the above lemma, we need to first establish a useful and generic result concerning the marginal influence spread given a partial realization (Lemma 3.6), which is proved in turn from two basic and intuitive results (Lemmas 3.4 and 3.5). The result of Lemma 3.6 also provides an alternative and perhaps more intuitive proof of the adaptive submodularity of the independent cascade model under the full-adoption feedback, and thus it may be of independent interest.

Given an influence graph $G = (V, E, p)$, we use $\sigma^G(S)$ to denote the expected influence spread of the seed set S on influence graph G . For a node set $A \subseteq V$, we use $G \setminus A$ to denote a reduced influence graph from G in which all nodes in A and its incident edges are removed, and the remaining edges has the same influence probabilities as in G .

► **Lemma 3.4.** *Under the independent cascade model with full-adoption feedback, for any influence graph $G = (V, E, p)$, partial realization ψ , and any node $i \notin \Gamma(\psi)$, we have*

$$\Delta_f(i|\psi) = \sigma^{G \setminus \Gamma(\psi)}(\{i\}), \quad (6)$$

i.e., the marginal influence spread of node i given a partial realization equals to its influence on the reduced graph where nodes in $\Gamma(\psi)$ are removed.

Proof. We use E_1 to denote all edges that are included in ψ , i.e., all edges that have already revealed their status. These are the outgoing edges from nodes activated in ψ , i.e. outgoing edges from $\Gamma(\psi)$. We use E_2 to denote edges in graph $G \setminus \Gamma(\psi)$ and set $E_3 = E \setminus (E_1 \cup E_2)$. Then E_2 are the edges in G that are not incident to nodes in $\Gamma(\psi)$ and E_3 are incoming edges of nodes in $\Gamma(\psi)$ that come from nodes in $V \setminus \Gamma(\psi)$. We use Φ_i to denote a random realization of edges in E_i ($i = 1, 2, 3$), and we know that they are mutually independent in the IC model. Let $S = \text{dom}(\psi)$. Now, we have

$$\begin{aligned} \Delta_f(i|\psi) &= \mathbb{E}_{\Phi} [f(S \cup \{i\}, \Phi) - f(S, \Phi) | \Phi \sim \psi] \\ &= \mathbb{E}_{\Phi_1, \Phi_2, \Phi_3} [f(S \cup \{i\}, (\Phi_1, \Phi_2, \Phi_3)) - f(S, (\Phi_1, \Phi_2, \Phi_3)) | (\Phi_1, \Phi_2, \Phi_3) \sim \psi] \\ &= \mathbb{E}_{\Phi_2} \left[\mathbb{E}_{\Phi_3} [f(S \cup \{i\}, (\psi, \Phi_2, \Phi_3)) - f(S, (\psi, \Phi_2, \Phi_3))] \right]. \end{aligned} \quad (7)$$

The second equality comes from the fact that Φ_1 , Φ_2 and Φ_3 are independent and the third equality comes from the fact that $\Phi_1 = \psi$, due to the full-adoption feedback model. We also have

$$\sigma^{G \setminus \Gamma(\psi)}(\{i\}) = \mathbb{E}_{\Phi_2} [f(\{i\}, \Phi_2)]. \quad (8)$$

To prove that Eq.(7) is the same as Eq.(8), it is suffice to show that for any fixed ϕ_2, ϕ_3 that are realizations of edges in E_2 and E_3 respectively, we have

$$\Gamma(\{i\}, \phi_2) = \Gamma(S \cup \{i\}, (\psi, \phi_2, \phi_3)) \setminus \Gamma(S, (\psi, \phi_2, \phi_3)). \quad (9)$$

Recall that we equate a realization with a live-edge graph, and thus in the above notation, ϕ_2 is considered as a live-edge graph in graph $G \setminus \Gamma(\psi)$ and $\Gamma(\{i\}, \phi_2)$ means the set of nodes reachable from i in the live-edge graph of ϕ_2 . We first prove that $\Gamma(\{i\}, \phi_2) \subseteq \Gamma(S \cup \{i\}, (\psi, \phi_2, \phi_3)) \setminus \Gamma(S, (\psi, \phi_2, \phi_3))$. For any node $u \in \Gamma(\{i\}, \phi_2)$, we know that it can be reached from node i via a path in graph $G \setminus \Gamma(\psi)$, the path only contains edges in ϕ_2 , which also means u is in the graph $G \setminus \Gamma(\psi)$. Due to the existence of the path, we know that $u \in \Gamma(S \cup \{i\}, (\psi, \phi_2, \phi_3))$. We claim that $u \notin \Gamma(S, (\psi, \phi_2, \phi_3))$. This is because we are dealing with the full-adoption feedback model, and thus $\Gamma(S, (\psi, \phi_2, \phi_3))$ is the nodes reachable from $S = \text{dom}(\psi)$, which means $\Gamma(S, (\psi, \phi_2, \phi_3)) = \Gamma(\psi)$, and thus $u \in \Gamma(S, (\psi, \phi_2, \phi_3))$ would conflict with the fact u is in $G \setminus \Gamma(\psi)$. Therefore, $u \in \Gamma(S \cup \{i\}, (\psi, \phi_2, \phi_3)) \setminus \Gamma(S, (\psi, \phi_2, \phi_3))$.

On the other side, for any node $u \in \Gamma(S \cup \{i\}, (\psi, \phi_2, \phi_3)) \setminus \Gamma(S, (\psi, \phi_2, \phi_3))$, the node u can be reached from the node i via a path in the live-edge graph (ψ, ϕ_2, ϕ_3) , but u cannot be reached in the live-edge graph (ψ, ϕ_2, ϕ_3) from any node v that can be activated by $\text{dom}(\psi)$ in ψ , because otherwise $u \in \Gamma(S, (\psi, \phi_2, \phi_3))$. This implies that the path from i to u must be in the live-edge graph ϕ_2 in graph $G \setminus \Gamma(\psi)$. That is, $u \in \Gamma(\{i\}, \phi_2)$. Therefore, Eq.(9) holds, and this completes the proof. \blacktriangleleft

► Lemma 3.5. *Given an influence graph $G = (V, E, p)$. For any two node sets $A, B \subseteq V$ with $A \subseteq B$, for any $u \in V \setminus B$, we have $\sigma^{G \setminus B}(\{i\}) \leq \sigma^{G \setminus A}(\{i\})$.*

Proof (Sketch). The proof is intuitively straightforward, since removing nodes could only hurt the influence spread. A more rigorous proof could be carried out by arguing for any random realization Φ_1 for edges in $G \setminus B$ and any random realization Φ_2 for edges in $G \setminus A$ but not in $G \setminus B$, (a) Φ_1 and Φ_2 are independent in the IC model; (b) $\sigma^{G \setminus B}(\{i\}) = \mathbb{E}_{\Phi_1} [|\Gamma(\{i\}, \Phi_1)|]$; (c) $\sigma^{G \setminus A}(\{i\}) = \mathbb{E}_{\Phi_1, \Phi_2} [|\Gamma(\{i\}, \Phi_1 \cup \Phi_2)|]$; and (d) $\Gamma(\{i\}, \Phi_1) \subseteq \Gamma(\{i\}, \Phi_1 \cup \Phi_2)$. \blacktriangleleft

► **Lemma 3.6.** *Under the independent cascade model with full-adoption feedback, for any two partial realizations ψ_1 and ψ_2 , if $\Gamma(\psi_1) \subseteq \Gamma(\psi_2)$, then for any node $i \notin \Gamma(\psi_1)$, $\Delta_f(i|\psi_2) \leq \Delta_f(i|\psi_1)$.*

Proof. First, note that if $i \in \Gamma(\psi_2) \setminus \Gamma(\psi_1)$, then $\Delta_f(i|\psi_2) = 0$, so $\Delta_f(i|\psi_2) \leq \Delta_f(i|\psi_1)$. We now consider that $i \notin \Gamma(\psi_2)$. By Lemma 3.4, $\Delta_f(i|\psi_1) = \sigma^{G \setminus \Gamma(\psi_1)}(\{i\})$, and $\Delta_f(i|\psi_2) = \sigma^{G \setminus \Gamma(\psi_2)}(\{i\})$. Since $\Gamma(\psi_1) \subseteq \Gamma(\psi_2)$, by Lemma 3.5, we have $\sigma^{G \setminus \Gamma(\psi_2)}(\{i\}) \leq \sigma^{G \setminus \Gamma(\psi_1)}(\{i\})$. Thus, the lemma holds. ◀

We are now ready to prove Lemma 3.3.

Proof Lemma 3.3. First, we consider the left-hand side of Eq. (5). For any $t \in [0, 1]$, $i \in [n]$ and small enough amount of time dt , the clock C_i sends out signals with probability $x_i dt$ during the time interval $[t, t + dt]$. Since signals are sent out independently, the probability that more than one clock send out signals simultaneously in time interval $[t, t + dt]$ is of order $O((dt)^2)$, which can be considered negligible comparing to dt . Thus we have

$$\mathbb{E}[f(\Psi(t + dt)) - f(\Psi(t)) | \Psi(t) = \psi] = \sum_{i \notin \text{dom}(\psi)} x_i dt \cdot \Delta_f(i|\psi), \quad (10)$$

Rewriting the above equation, we derive that

$$\mathbb{E}\left[\frac{df(\Psi(t))}{dt} | \Psi(t) = \psi\right] = \sum_{i \notin \text{dom}(\psi)} x_i \Delta_f(i|\psi) = \sum_{i \notin \Gamma(\psi)} x_i \Delta_f(i|\psi). \quad (11)$$

The second equality holds because $\Delta_f(i|\psi) = 0$ for any node $i \in \Gamma(\psi)$ in the full-adoption feedback model.

Next, we consider the right-hand side of Eq. (5). We write $\mathbf{x} = (x_1, \dots, x_n)$ and use the indicator vector $\mathbf{I}_S \in \{0, 1\}^n$ to denote an n -dimensional 0-1 vector such that the coordinate i is 1 if and only if $i \in S$. By the monotonicity of the function $f^+(\cdot)$, we have

$$f^+(x_1, \dots, x_n) \leq f^+(\mathbf{x} \vee \mathbf{I}_{\Gamma(\psi)}). \quad (12)$$

Consider the optimal adaptive policy π^+ of $f^+(\mathbf{x} \vee \mathbf{I}_{\Gamma(\psi)})$ as defined in Definition 2.8. We can assume π^+ selects nodes in $\Gamma(\psi)$ at the beginning since they will eventually appear in the seed set regardless of the realization of the live-edge graph. For $i \notin \Gamma(\psi)$, π^+ would select node i as a seed with probability x_i , according to Definition 2.8. Let Ψ_i denote the partial realization just before π^+ selects node i , given that π^+ first selects nodes in $\Gamma(\psi)$ and sees their feedback. Then we have $\Gamma(\psi) \subseteq \Gamma(\Psi_i)$. Conditioned on Ψ_i , the selection of i provides a marginal gain of $\Delta_f(i|\Psi_i)$ for the influence spread. When we take its expectation over Ψ_i and then multiply it with x_i , we obtain the overall marginal gain of selecting i as a seed in policy π^+ . When summing over all $i \notin \Gamma(\psi)$, together with the non-adaptive influence spread of seed nodes in $\Gamma(\psi)$, we thus obtain:

$$f^+(\mathbf{x} \vee \mathbf{I}_{\Gamma(\psi)}) = \sum_{i \notin \Gamma(\psi)} x_i \cdot \mathbb{E}_{\Psi_i}[\Delta_f(i|\Psi_i)] + \sigma(\Gamma(\psi)). \quad (13)$$

Combining Eq. (11), (12), (13), it suffices to prove

$$\Delta_f(i|\Psi_i) \leq \Delta_f(i|\psi) \quad (14)$$

for any $i \notin \Gamma(\psi)$ and any partial realization Ψ_i such that $\Gamma(\psi) \subseteq \Gamma(\Psi_i)$, but this is exactly what is shown in Lemma 3.6. This completes our proof. ◀

24:10 On Adaptivity Gaps of Influence Maximization

A couple of remarks are now in order concerning Lemma 3.3 and its proof. First, Inequality (14) looks very much like the adaptive submodularity condition, and we could directly apply adaptive submodularity should we have $\psi \subseteq \Psi_i$. However, by our construction, we only have $\Gamma(\psi)$ as the initial seed set for Ψ_i , and the partial realization in Ψ_i from the seeds in $\text{dom}(\psi)$ may not be exactly the same as ψ , so we do not have $\psi \subseteq \Psi_i$. Instead, we only have $\Gamma(\psi) \subseteq \Gamma(\Psi_i)$. Therefore, we provide a separate proof (Lemmas 3.4, 3.5, and 3.6) to show that the above condition is enough to prove Inequality (14). Even though our proof does not directly apply the adaptive submodularity result of the IC model with the full-adoption feedback, the essence of proof related to Inequality (14) is still due to the diminishing return behavior of the model in the adaptive setting.

Second, the overall proof structure of Lemma 3.3 roughly follows the proof structure in [3]. However, because in the full-adoption feedback model the feedback from two different nodes may be correlated, we cannot exactly follow the proof structure in [3]. In particular, we have to incorporate $\Gamma(\psi)$ directly into the seeds of an adaptive strategy π^+ to avoid such correlation to interfere with our analysis. This results in the term $\sigma(\Gamma(\psi))$ in Inequality (5) of Lemma 3.3 instead of the term $|\Gamma(\psi)|$ that would be derived should we exactly follow [3], and eventually the term $\sigma(\Gamma(\psi))$ leads to the extra factor of 2 in the adaptivity gap given in Theorem 3.1, which does not appear in [3].

Next, we introduce the concept of the *boundary* of a partial realization. We need to use the property of the boundary in in-arborescences (Lemma 3.8) to properly bound the extra term $\sigma(\Gamma(\psi))$ (Lemma 3.9).

► **Definition 3.7** (Boundary of a partial realization). *In the full-adoption feedback model, for any partial realization ψ , we use $\partial(\psi)$ to denote the boundary of the partial realization, i.e., the set of nodes $\partial(\psi) \subseteq \Gamma(\psi)$ with minimum cardinality such that there is no directed edges in the original graph G from $\Gamma(\psi) \setminus \partial(\psi)$ to $V \setminus \Gamma(\psi)$. We remark that when there are more than one such sets, we take an arbitrary one.*

The main property we rely on the structure of an in-arborescence is that the boundary of any partial realization can be bounded by the number of seeds that have been selected. Formally, we have

► **Lemma 3.8.** *When the influence graph is an in-arborescence, for any partial realization ψ , we have $|\partial(\psi)| \leq |\text{dom}(\psi)|$.*

Proof. Consider any partial realization ψ and any node $v \in \text{dom}(\psi)$. Take the unique directed path from node v to the root u , let \bar{v} denote the node on the path which is (i) contained in $\Gamma(\psi)$ and (ii) closest to the root u . Then we set $S = \{\bar{v} : v \in \text{dom}(\psi)\}$. Clearly there is no directed edge from $\Gamma(\psi) \setminus S$ to $V \setminus \Gamma(\psi)$ and we have $|\partial(\psi)| \leq |S| \leq |\text{dom}(\psi)|$. ◀

Now, we give an upper bound on the term $\sigma(\Gamma(\psi))$.

► **Lemma 3.9.** *For any partial realization ψ*

$$\sigma(\Gamma(\psi)) \leq |\Gamma(\psi)| + \sigma(\partial(\psi)). \quad (15)$$

Moreover, when the influence graph is an in-arborescence, we have

$$\sigma(\Gamma(\psi)) \leq |\Gamma(\psi)| + \text{OPT}_N(G, |\text{dom}(\psi)|). \quad (16)$$

Proof. Fix any realization $\phi \sim \psi$, and then consider any node v in $\Gamma(\Gamma(\psi), \phi) \setminus \Gamma(\partial(\psi), \phi)$. There must exist a directed path P from $\Gamma(\psi) \setminus \partial(\psi)$ to v , and the path P does not contain any nodes in $\partial(\psi)$. According to the definition of the boundary set $\partial(\psi)$, there is no

directed path from $\Gamma(\psi) \setminus \partial(\psi)$ to $V \setminus \Gamma(\psi)$, unless it goes through a node in $\partial(\psi)$. Thus we conclude that $v \in \Gamma(\psi) \setminus \partial(\psi)$ and this gives proof for Eq. (15). With Lemma 3.8, we have $\sigma(\partial(\psi)) \leq \text{OPT}_N(G, |\text{dom}(\psi)|)$. Therefore, Inequality (16) holds. ◀

For any fixed influence graph G , we can view $\text{OPT}_N(G, k)$ as a function of the budget k , we prove that $\text{OPT}_N(G, k)$ is “weakly concave” for k , as stated in the following lemma.

► **Lemma 3.10.** *For any fixed influence graph G , let X be a random variable taking value from $\{0, 1, \dots, n\}$, with mean value $\mathbb{E}[X] = k$. Then we have*

$$\mathbb{E}[\text{OPT}_N(G, X)] \leq \frac{e}{e-1} \text{OPT}_N(G, \mathbb{E}[X]) = \frac{e}{e-1} \text{OPT}_N(G, k). \quad (17)$$

Proof. Let $\text{Greedy}_N(G, k)$ denote the non-adaptive greedy solution that selects k seed nodes. For $X \in \{0, 1, \dots, n\}$, the greedy solution is $1 - 1/e$ approximate to the optimal solution, i.e.,

$$\text{OPT}_N(G, X) \leq \frac{e}{e-1} \text{Greedy}_N(G, X). \quad (18)$$

We note that the greedy solution $\text{Greedy}_N(G, X)$ is concave in X , due to the submodularity of the influence spread function. Then taking expectation over both sides of Eq. (18), by Jensen’s inequality, we have

$$\begin{aligned} \mathbb{E}[\text{OPT}_N(G, X)] &\leq \frac{e}{e-1} \mathbb{E}[\text{Greedy}_N(G, X)] \leq \frac{e}{e-1} \text{Greedy}_N(G, \mathbb{E}[X]) \\ &\leq \frac{e}{e-1} \text{OPT}_N(G, \mathbb{E}[X]) = \frac{e}{e-1} \text{OPT}_N(G, k). \end{aligned} \quad (19)$$

This concludes the proof. ◀

Putting things together, we are able to prove Theorem 3.1 as follows.

Proof of Theorem 3.1. When the influence graph is an in-arborescence, for any configuration (x_1, \dots, x_n) satisfying $\sum_i x_i = k$, for any $t \in [0, 1]$ and any fixed partial realization ψ , we have

$$\begin{aligned} &\mathbb{E} \left[\frac{df(\Psi(t))}{dt} \mid \Psi(t) = \psi \right] \\ &\geq f^+(x_1, \dots, x_n) - \sigma(\Gamma(\psi)) && \text{by Lemma 3.3} \\ &\geq f^+(x_1, \dots, x_n) - |\Gamma(\psi)| - \text{OPT}_N(G, |\text{dom}(\psi)|) && \text{by Lemma 3.9} \\ &= f^+(x_1, \dots, x_n) - f(\Psi(t)) - \text{OPT}_N(G, |\text{dom}(\Psi(t))|). \end{aligned} \quad (20)$$

Taking expectation over $\Psi(t)$, we have for any $t \in [0, 1]$,

$$\begin{aligned} \frac{d}{dt} \mathbb{E}[f(\Psi(t))] &= \mathbb{E} \left[\frac{df(\Psi(t))}{dt} \right] = \mathbb{E}_{\Psi(t)} \mathbb{E} \left[\frac{df(\Psi(t))}{dt} \mid \Psi(t) \right] \\ &\geq f^+(x_1, \dots, x_n) - \mathbb{E}[f(\Psi(t))] - \mathbb{E}[\text{OPT}_N(G, |\text{dom}(\Psi(t))|)] \\ &\geq f^+(x_1, \dots, x_n) - \frac{e}{e-1} \text{OPT}_N(G, k) - \mathbb{E}[f(\Psi(t))]. \end{aligned} \quad (21)$$

The first equality above is by the linearity of expectation. The second equality above is by the law of total expectation. The first inequality is by Eq.(20), and the second inequality holds due to Lemma 3.10 and the fact that

$$\mathbb{E}[|\text{dom}(\Psi(t))|] = \sum_i (1 - e^{-tx_i}) \leq \sum_i tx_i \leq k$$

24:12 On Adaptivity Gaps of Influence Maximization

for any $t \leq 1$. Solving the above differential inequality in Eq.(21), we obtain

$$\mathbb{E}[f(\Psi(t))] \geq (1 - e^{-t}) \left[f^+(x_1, \dots, x_n) - \frac{e}{e-1} \text{OPT}_N(G, k) \right]. \quad (22)$$

In particular, when $t = 1$, we have

$$\mathbb{E}[f(\Psi(1))] \geq \left(1 - \frac{1}{e}\right) \left[f^+(x_1, \dots, x_n) - \frac{e}{e-1} \text{OPT}_N(G, k) \right]. \quad (23)$$

Finally, we have

$$\begin{aligned} \text{OPT}_N(G, k) &= \sup_{x_1 + \dots + x_n = k} F(x_1, \dots, x_n) \\ &\geq \sup_{x_1 + \dots + x_n = k} \mathbb{E}[f(\Psi(1))] \\ &\geq \sup_{x_1 + \dots + x_n = k} \left(1 - \frac{1}{e}\right) \left[f^+(x_1, \dots, x_n) - \frac{e}{e-1} \text{OPT}_N(G, k) \right] \\ &\geq \left(1 - \frac{1}{e}\right) \text{OPT}_A(G, k) - \text{OPT}_N(G, k). \end{aligned} \quad (24)$$

The first equality above comes from the pipage rounding procedure in [9]. The first inequality above is by Lemma 3.2. The second inequality is by Eq. (23). The last equality is by the definition of f^+ (Definition 2.8). Thus we conclude that the adaptivity gap is at most $\frac{2e}{e-1}$ in the case of an in-arborescence. \blacktriangleleft

4 Adaptivity Gap for Out-arborescence

In this section, we give an upper bound on the adaptivity gap when the influence graph is an out-arborescence. Formally,

► **Theorem 4.1.** *When the influence graph is an out-arborescence, the adaptivity gap for the IM problem in the IC model with full-adoption feedback is at most 2.*

We first introduce some notations. For any node $u \in V$ and any seed set $S \subseteq V$, we define $\sigma_u(S) := \Pr_{\Phi} [u \in \Gamma(S, \Phi)]$, i.e., the probability that the node u is activated when S is the seed set. Similarly, for any adaptive policy π , we define $\sigma_u(\pi) := \Pr_{\Phi} [u \in \Gamma(V(\pi, \Phi), \Phi)]$, i.e., the probability that the node u is activated under policy π . We will extend the definition for the multilinear extension (Definition 2.7) and the definition for f^+ (Definition 2.8) correspondingly. To be more specific, we define

$$F_u(x_1, \dots, x_n) = \sum_{S \subseteq [n]} \left[\left(\prod_{i \in S} x_i \prod_{i \notin S} (1 - x_i) \right) \sigma_u(S) \right], \quad (25)$$

and

$$f_u^+(x_1, \dots, x_n) = \sup_{\pi} \left\{ \sigma_u(\pi) : \Pr_{\Phi \sim \mathcal{P}} [i \in V(\pi, \Phi)] = x_i, \forall i \in [n] \right\}. \quad (26)$$

In order to show Theorem 4.1, we again transform an adaptive policy to a non-adaptive policy and compare their influence spread. Here, we utilize a new approach based on the structure of out-arborescences and prove a stronger result. That is, we would prove that the probability for any node u to become active in the multilinear extension (policy) is at least half of the optimal adaptive policy (Lemma 4.2). This requires us to give a fine-grained bound on the optimal adaptive policy (Lemma 4.3) and the multilinear extension (Lemma 4.4).

► **Lemma 4.2.** *When the influence graph is an out-arborescence, for any node $u \in V$ and any configuration (x_1, \dots, x_n) , we have*

$$f_u^+(x_1, \dots, x_n) \leq 2F_u(x_1, \dots, x_n). \quad (27)$$

Once we have the result of Lemma 4.2, we can prove Theorem 4.1 as follows.

Proof of Theorem 4.1. Given Lemma 4.2, we have

$$\begin{aligned} \text{OPT}_N(G, k) &= \sup_{x_1 + \dots + x_n = k} F(x_1, \dots, x_n) = \sup_{x_1 + \dots + x_n = k} \sum_u F_u(x_1, \dots, x_n) \\ &\geq \frac{1}{2} \cdot \sup_{x_1 + \dots + x_n = k} \sum_u f_u^+(x_1, \dots, x_n) \geq \frac{1}{2} \cdot \sup_{x_1 + \dots + x_n = k} f^+(x_1, \dots, x_n) \geq \frac{1}{2} \cdot \text{OPT}_A(G, k). \quad \blacktriangleleft \end{aligned}$$

We now show how to prove Lemma 4.2. We note that node u 's predecessors (nodes that can reach node u in the original graph) form a directed line when the influence graph is an out-arborescence. We slightly abuse the notation and use node i to indicate the $(i-1)^{\text{th}}$ predecessor of node u , notice that node u itself is represented as node 1. We ignore all other nodes since they do not affect either sides of Eq. (27). We use p_i to denote the probability that the node i can reach node 1. The following lemma gives an upper bound on the optimal adaptive strategy.

► **Lemma 4.3.** *For any i , $f_1^+(x_1, \dots, x_n) \leq \sum_{j=1}^i x_j p_j + p_{i+1}$.*

Proof. Let π be any adaptive strategy satisfying $\Pr_{\Phi \sim \mathcal{P}} [i \in V(\pi, \Phi)] = x_i, i \in [n]$. Let \mathcal{E}_i denote the event that node 1 becomes active right after π chooses node i . Furthermore, we use \mathcal{E}_i to denote the event that node 1 becomes active right after π chooses a node from $\{i, i+1, \dots, n\}$. We notice that events $\mathcal{E}_1, \dots, \mathcal{E}_n$ are disjoint and we have

$$f_1^+(x_1, \dots, x_n) = \sum_{j=1}^n \Pr[\mathcal{E}_j] = \sum_{j=1}^i \Pr[\mathcal{E}_j] + \Pr[\mathcal{E}_{i+1:}] \quad \forall i. \quad (28)$$

It is easy to see that

$$\Pr[\mathcal{E}_{i+1:}] \leq p_{i+1}, \quad (29)$$

since the event $\mathcal{E}_{i+1:}$ can only happen when the node $i+1$ can reach node 1. Moreover, let \mathcal{F}_i denote the event that the policy π selects the node i before any nodes in $\{1, \dots, i\}$ are active. Then we have for any $j \in [n]$,

$$\Pr[\mathcal{E}_j] = \Pr_{\Phi}[\mathcal{E}_j | \mathcal{F}_j] \cdot \Pr_{\Phi}[\mathcal{F}_j] \leq \Pr_{\Phi}[\mathcal{E}_j | \mathcal{F}_j] \cdot \Pr_{\Phi}[j \in V(\pi, \Phi)] = p_j \cdot x_j. \quad (30)$$

The first equality holds since the event \mathcal{E}_j can only happen when π selects the node j before any nodes $\{1, \dots, j\}$ are active. Combining Eq. (28) (29) (30), we complete the proof. ◀

We measure the marginal contribution of node i in the next lemma. Intuitively, we can see that $F_1(0, \dots, 0, x_i, \dots, x_n) - F_1(0, \dots, 0, x_{i+1}, \dots, x_n)$ measures the marginal contribution of i in activating node 1, when node i moves from no probability of being selected as a seed to probability of x_i being selected as the seed, under the situation that no nodes in $\{1, \dots, i-1\}$ can be seeds while node $j > i$ has probability x_j of being selected as a seed. Then this marginal contribution only happens when all three conditions hold: (a) possible seeds in $\{i+1, \dots, n\}$ cannot activate i , which has probability $1 - F_i(0, \dots, 0, x_{i+1}, \dots, x_n)$; (b) node i is activated as a seed, which has probability x_i , and (c) node i passes influence and activates node 1, which has probability p_i .

24:14 On Adaptivity Gaps of Influence Maximization

► **Lemma 4.4.** *For any i , we have*

$$F_1(0, \dots, 0, x_i, \dots, x_n) - F_1(0, \dots, 0, x_{i+1}, \dots, x_n) = x_i p_i (1 - F_i(0, \dots, 0, x_{i+1}, \dots, x_n)).$$

Proof. Since the node 1's predecessors form a directed line, for any i we have

$$\begin{aligned} & F_1(0, \dots, 0, x_i, \dots, x_n) - F_1(0, \dots, 0, x_{i+1}, \dots, x_n) \\ &= p_i \cdot (F_i(0, \dots, 0, x_i, \dots, x_n) - F_i(0, \dots, 0, x_{i+1}, \dots, x_n)) \\ &= p_i \cdot (1 - (1 - x_i)(1 - F_i(0, \dots, 0, x_{i+1}, \dots, x_n))) - F_i(0, \dots, 0, x_{i+1}, \dots, x_n) \\ &= p_i x_i (1 - F_i(0, \dots, 0, x_{i+1}, \dots, x_n)). \end{aligned}$$

The first two equalities hold because the realization and the selection of nodes are independent. ◀

We are now back to prove Lemma 4.2.

Proof of Lemma 4.2. We use j to denote the minimum index that satisfies $F_j(0, \dots, x_{j+1}, \dots, x_n) > \frac{1}{2}$. If such index does not exist, we simply set $j = n + 1$. Then, we have

$$\begin{aligned} & F_1(x_1, \dots, x_n) \\ &= \sum_{i=1}^{j-1} (F_1(0, \dots, 0, x_i, \dots, x_n) - F_1(0, \dots, 0, x_{i+1}, \dots, x_n)) + F_1(0, \dots, 0, x_j, \dots, x_n) \\ &= \sum_{i=1}^{j-1} x_i p_i (1 - F_i(0, \dots, 0, x_{i+1}, \dots, x_n)) + F_1(0, \dots, 0, x_j, \dots, x_n) \\ &= \sum_{i=1}^{j-1} x_i p_i (1 - F_i(0, \dots, 0, x_{i+1}, \dots, x_n)) + p_j \cdot F_j(0, \dots, 0, x_j, \dots, x_n) \\ &\geq \frac{1}{2} \sum_{i=1}^{j-1} x_i p_i + \frac{1}{2} p_j \\ &\geq \frac{1}{2} f_1^+(x_1, \dots, x_n). \end{aligned} \tag{31}$$

The second equality comes from Lemma 4.4 and the last inequality comes from Lemma 4.3. ◀

5 Adaptivity Gap for One-Directional Bipartite Graphs

In this section, we give an upper bound on the adaptivity gap of the influence maximization problem in the IC model with full-adoption feedback under one-directional bipartite graphs $G(L, R, E, p)$, where L and R are the two set of nodes on the left side and right side respectively, and $E \subseteq L \times R$ are a set of edges only pointing from a left-side node to a right-side node, and p maps each edge to a probability. Our upper bound is tight as it matches the lower bound derived in [25] and it also improves the results developed in [14, 18]. The proof strategy adopted for bipartite graphs is a relatively easy application of our approaches in previous sections, which relates the multilinear extension and the optimal strategy (See Appendix A).

► **Theorem 5.1.** *When the influence graph is a one-directional bipartite graph $G(L, R, E, p)$, the adaptivity gap on the influence maximization problem in the IC model with full-adoption feedback is $\frac{e}{e-1}$.*

6 Lower Bounds on the Adaptivity Gap

In this section, we give an example showing that the adaptivity gap is no less than $e/(e-1)$ in the full-adoption feedback model, even when the influence graph is a directed line, a special case of both the in-arborescence and the out-arborescence.

► **Theorem 6.1.** *The adaptivity gap for the IM problem in the IC model with full-adoption feedback is at least $e/(e-1)$, even when the influence graph is a directed line.*

Proof. Consider the following influence graph $G(V, E, p)$: the graph is a directed line with vertices $v_{11}, \dots, v_{1t}, v_{21}, \dots, v_{2t}, \dots, v_{k1}, \dots, v_{kt}$, and each edge is live with probability $1-1/t$. Moreover, we have a budget k . Combining the following two claims, we can conclude that the adaptivity gap is greater than or equal to $e/(e-1)$. The proofs of the claims are in Appendix B.

▷ Claim 6.2. For any $\epsilon > 0$, if $k \geq 8/\epsilon^3$, we have $\mathbb{E}[\text{OPT}_A(G, k)] \geq (1-\epsilon)kt$.

▷ Claim 6.3. The optimal non-adaptive strategy is to select v_{11}, \dots, v_{k1} as seeds. Thus, we have $\mathbb{E}[\text{OPT}_N(G, k)] = (1 - (1 - 1/t)^t)kt$. ◀

Discussion on Existing Approaches. There are two types of strategies for proving upper bounds on adaptivity gaps. One common strategy is to convert any adaptive strategy to the multilinear extension as in [27, 3] and our paper. The other is to convert the adaptive strategy to the *random walk non-adaptive strategy* [16, 17, 8]. Here we claim that using the instance constructed in Theorem 6.1, we can show that these two strategies can not yield better-than-2 upper bounds on the adaptivity gap. We defer the detailed discussions to Appendix B.

7 Conclusion

In this paper, we consider several families of influence graphs and give the first constant upper bounds on adaptivity gaps for them under the full-adoption feedback model. Our methods tackle the correlations on the feedback and hopefully can be applied to other adaptive stochastic optimization problems. For future directions, there are still gaps between our lower and upper bounds for both in-arborescences and out-arborescences, so it would be interesting to close the gap. Another open question is to settle down the adaptivity gap for general influence graphs under the IC model with the full-adoption feedback. The adaptive gap for the linear threshold model and other diffusion models are also open.

References

- 1 Marek Adamczyk, Maxim Sviridenko, and Justin Ward. Submodular stochastic probing on matroids. *Mathematics of Operations Research*, 41(3):1022–1038, 2016.
- 2 Noga Alon, Iftah Gamzu, and Moshe Tennenholtz. Optimizing budget allocation among channels and influencers. In *WWW*, pages 381–388. ACM, 2012.
- 3 Arash Asadpour and Hamid Nazerzadeh. Maximizing stochastic monotone submodular functions. *Management Science*, 62(8):2374–2391, 2015.
- 4 Ashwinkumar Badanidiyuru, Christos Papadimitriou, Aviad Rubinfeld, Lior Seeman, and Yaron Singer. Locally adaptive optimization: Adaptive seeding for monotone submodular functions. In *SODA*. SIAM, 2016.

24:16 On Adaptivity Gaps of Influence Maximization

- 5 Nicola Barbieri, Francesco Bonchi, and Giuseppe Manco. Topic-aware social influence propagation models. In *ICDM'12*, 2012.
- 6 Shishir Bharathi, David Kempe, and Mahyar Salek. Competitive influence maximization in social networks. In *WINE*, pages 306–311. Springer, 2007.
- 7 Christian Borgs, Michael Brautbar, Jennifer Chayes, and Brendan Lucier. Maximizing Social Influence in Nearly Optimal Time. In *SODA'14*, pages 946–957. ACM-SIAM, 2014.
- 8 Domagoj Bradac, Sahil Singla, and Goran Zuzic. (Near) Optimal Adaptivity Gaps for Stochastic Multi-Value Probing. *arXiv preprint*, 2019. [arXiv:1902.01461](https://arxiv.org/abs/1902.01461).
- 9 Gruia Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM Journal on Computing*, 40(6):1740–1766, 2011.
- 10 Chandra Chekuri, Jan Vondrak, and Rico Zenklusen. Dependent randomized rounding via exchange properties of combinatorial structures. In *FOCS*, pages 575–584. IEEE, 2010.
- 11 Wei Chen, Laks VS Lakshmanan, and Carlos Castillo. *Information and Influence Propagation in Social Networks*. Morgan & Claypool Publishers, 2013.
- 12 Wei Chen, Chi Wang, and Yajun Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *KDD'10*, 2010.
- 13 Wei Chen, Yajun Wang, and Siyu Yang. Efficient influence maximization in social networks. In *Proceedings of the 15th ACM SIGKDD*. ACM, 2009.
- 14 Kaito Fujii and Shinsaku Sakaue. Beyond Adaptive Submodularity: Approximation Guarantees of Greedy Policy with Adaptive Submodularity Ratio. In *ICML*, pages 2042–2051, 2019.
- 15 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. [arXiv version \(arXiv:1003.3967\)](https://arxiv.org/abs/1003.3967) includes discussions on the myopic feedback model.
- 16 Anupam Gupta, Viswanath Nagarajan, and Sahil Singla. Algorithms and adaptivity gaps for stochastic probing. In *SODA*. SIAM, 2016.
- 17 Anupam Gupta, Viswanath Nagarajan, and Sahil Singla. Adaptivity gaps for stochastic probing: Submodular and XOS functions. In *SODA*. SIAM, 2017.
- 18 Daisuke Hatano, Takuro Fukunaga, and Ken-Ichi Kawarabayashi. Adaptive Budget Allocation for Maximizing Influence of Advertisements. In *IJCAI*, pages 3600–3608, 2016.
- 19 David Kempe, Jon M. Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. *Theory of Computing*, 11(4):105–147, 2015. Conference version appeared in *KDD'2003*.
- 20 Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne M Vanbriesen, and Natalie Glance. Cost-effective outbreak detection in networks. In *ACM Knowledge Discovery and Data Mining*, pages 420–429, 2007.
- 21 Yuchen Li, Ju Fan, Yanhao Wang, and Kian-Lee Tan. Influence Maximization on Social Graphs: A Survey. *IEEE Trans. Knowl. Data Eng.*, 30(10):1852–1872, 2018.
- 22 Yishi Lin, Wei Chen, and John CS Lui. Boosting information spread: An algorithmic approach. In *ICDE*, pages 883–894. IEEE, 2017.
- 23 Wei Lu, Wei Chen, and Laks VS Lakshmanan. From competition to complementarity: comparative influence diffusion and maximization. *Proceedings of the VLDB Endowment*, 9(2):60–71, 2015.
- 24 Zaixin Lu, Zhao Zhang, and Weili Wu. Solution of Bharathi–Kempe–Salek conjecture for influence maximization on arborescence. *Journal of Combinatorial Optimization*, 33(2):803–808, 2017.
- 25 Binghui Peng and Wei Chen. Adaptive Influence Maximization with Myopic Feedback. In *NeurIPS*, 2019.
- 26 Guillaume Salha, Nikolaos Tziortziotis, and Michalis Vazirgiannis. Adaptive Submodular Influence Maximization with Myopic Feedback. In *ASONAM*, pages 455–462. IEEE, 2018.

- 27 Lior Seeman and Yaron Singer. Adaptive seeding in social networks. In *FOCS*, pages 459–468. IEEE, 2013.
- 28 Yaron Singer. Influence maximization through adaptive seeding. *ACM SIGecom Exchanges*, 15(1):32–59, 2016.
- 29 Tasuku Soma, Naonori Kakimura, Kazuhiro Inaba, and Ken-ichi Kawarabayashi. Optimal budget allocation: Theoretical guarantee and efficient algorithm. In *ICML*, 2014.
- 30 Lichao Sun, Weiran Huang, Philip S Yu, and Wei Chen. Multi-round influence maximization. In *KDD*, pages 2249–2258. ACM, 2018.
- 31 Youze Tang, Yanchen Shi, and Xiaokui Xiao. Influence maximization in near-linear time: A martingale approach. In *SIGMOD'15*, pages 1539–1554. ACM, 2015.
- 32 Youze Tang, Xiaokui Xiao, and Yanchen Shi. Influence maximization: near-optimal time complexity meets practical efficiency. In *SIGMOD'14*, 2014.
- 33 Guangmo Tong, Weili Wu, Shaojie Tang, and Ding-Zhu Du. Adaptive influence maximization in dynamic social networks. *IEEE/ACM Transactions on Networking (TON)*, 25(1):112–125, 2017.
- 34 Jan Vondrák. Submodularity in combinatorial optimization, 2007.
- 35 Ailian Wang, Weili Wu, and Lei Cui. On Bharathi–Kempe–Salek conjecture for influence maximization on arborescence. *Journal of Combinatorial Optimization*, 31(4):1678–1684, 2016.
- 36 Chi Wang, Wei Chen, and Yajun Wang. Scalable influence maximization for independent cascade model in large-scale social networks. *DMKD*, 2012.
- 37 Jing Yuan and Shaojie Tang. No Time to Observe: Adaptive Influence Maximization with Partial Feedback. In *IJCAI*, 2017.

A Missing Proof from Section 5

► **Theorem 5.1.** *When the influence graph is a one-directional bipartite graph $G(L, R, E, p)$, the adaptivity gap on the influence maximization problem in the IC model with full-adoption feedback is $\frac{e}{e-1}$.*

Proof. For each node u , it suffices to prove that for any configuration (x_1, \dots, x_n) ,

$$F_u(x_1, \dots, x_n) \geq \left(1 - \frac{1}{e}\right) f_u^+(x_1, \dots, x_n), \quad (32)$$

where F_u and f_u^+ are the same as defined in the proof of Theorem 4.1. We use p_i to denote the probability that node i can reach node u , then we have

$$F_u(x_1, \dots, x_n) = 1 - \prod_{i=1}^n (1 - p_i x_i). \quad (33)$$

On the other side, let \mathcal{E}_i denote the event that node u becomes active right after the optimal policy π^+ chooses node i . We know that $\Pr[\mathcal{E}_i] \leq x_i \cdot p_i$ and thus we can conclude that

$$f_u^+(x_1, \dots, x_n) = \sum_{i=1}^n \Pr[\mathcal{E}_i] \leq \sum_{i=1}^n x_i p_i. \quad (34)$$

Combining Eq. (33) (34) and the fact that

$$1 - \prod_{i=1}^n (1 - y_i) \geq \left(1 - \frac{1}{e}\right) \min\left\{1, \sum_{i=1}^n y_i\right\} \quad (35)$$

holds for all $y_i \in [0, 1]$, we can prove Eq. (32) and conclude the proof. ◀

B Missing Proofs and Further Discussions from Section 6

▷ **Claim 6.2.** For any $\epsilon > 0$, if $k \geq 8/\epsilon^3$, we have $\mathbb{E}[\text{OPT}_A(G, k)] \geq (1 - \epsilon)kt$.

Proof. Consider the following adaptive policy π : π always selects the inactive node that is closest to the origin of the directed line, until it reaches the budget. Let X_i ($i \in [k]$) denote the number of nodes that can be reached from the i^{th} seed and let $X = X_1 + \dots + X_k$. It is easy to see that $\mathbb{E}[\text{OPT}_A(G, k)] \geq \sigma(\pi) = \mathbb{E}[X]$. Let $Y_i \sim GE(1 - 1/t)$, i.e., Y_i is a geometric random variable parametrized with $1 - 1/t$. Y_1, \dots, Y_k are independent and we know that $\mathbb{E}[Y_i] = t$ and $\text{Var}[Y_i] = t^2 - t$. Our key observation is that $\mathbb{E}[X] = \mathbb{E}[\min\{Y_1 + \dots + Y_k, kt\}]$. By Chebyshev bounds, we have

$$\Pr[Y_1 + \dots + Y_k < (1 - \epsilon/2)kt] \leq \frac{4k(t^2 - t)}{\epsilon^2 k^2 t^2} \leq \frac{4}{\epsilon^2 k} \leq \epsilon/2. \quad (36)$$

Thus we know that

$$\begin{aligned} \mathbb{E}[\min\{Y_1 + \dots + Y_k, kt\}] &\geq \Pr[Y_1 + \dots + Y_k \geq (1 - \epsilon/2)kt] \cdot (1 - \epsilon/2)kt \\ &\geq (1 - \epsilon/2) \cdot (1 - \epsilon/2)kt \geq (1 - \epsilon)kt. \end{aligned} \quad (37)$$

This concludes the proof. ◁

▷ **Claim 6.3.** The optimal non-adaptive strategy is to select v_{11}, \dots, v_{k1} as seeds. Thus, we have $\mathbb{E}[\text{OPT}_N(G, k)] = (1 - (1 - 1/t)^t)kt$.

Proof. In the non-adaptive setting, for any node u and seed set S , we define the distance between the node u and the set S as the distance between u and the closest predecessor of u in S . We know that the probability that the node u is active only depends on the distance between u and S . Let N_i ($i \geq 0$) denote the set of nodes that has distance i with S . Then we know that (i) nodes in N_i are active with probability $(1 - 1/t)^i$, (ii) $N_0, N_1, \dots, N_{kt-1}$ are disjoint and $|N_i| \leq k$. Now we have that $\sigma(S) = \sum_{i=0}^{kt-1} (1 - 1/t)^i \cdot |N_i| \leq \sum_{i=0}^{t-1} (1 - 1/t)^i \cdot k$. Thus, we can conclude that the optimal non-adaptive solution is to select v_{11}, \dots, v_{k1} as seeds and $\mathbb{E}[\text{OPT}_N(G, k)] = \sum_{i=0}^{t-1} (1 - 1/t)^i \cdot k = (1 - (1 - 1/t)^t)kt$. ◁

Discussion on Existing Approaches. In this paragraph, we give a hard instance showing that existing approaches cannot yield better-than-2 upper bounds on the adaptivity gap. The hard instance is exactly the directed line constructed in Theorem 6.1, i.e., a directed line of length kt and each edge is live with probability $1 - 1/t$. We use node i to denote the $(i - 1)^{\text{th}}$ successor of the origin of the directed line, notice that the origin itself is denoted as node 1.

Multilinear Extension. One common strategy is to use the multilinear extension as in [27, 3]. In [3], they consider the *stochastic submodular optimization* problem and prove that $f^+(x_1, \dots, x_n) \leq \frac{e}{e-1} F(x_1, \dots, x_n)$ holds for any configuration (x_1, \dots, x_n) . We show that the ratio of $f^+(x_1, \dots, x_n)/F(x_1, \dots, x_n)$ can approach 2 in our example. To be more specific, consider the configuration $(1, 1/t, \dots, 1/t)$, we claim that $f^+(1, 1/t, \dots, 1/t) = kt$. Consider the adaptive policy π that always selects the inactive node that is closest to the origin of the directed line. The policy π will select the first node with probability 1 and other nodes with probability $1/t$, since it will seed a node if and only if its incoming edge is blocked, this can happen with probability $1/t$. On the other side, we have $F(1, 1/t, \dots, 1/t) \leq F(1 - 1/t, 0, \dots, 0) + F(1/t, \dots, 1/t) \leq (1 - 1/t)t + F(1/t, \dots, 1/t) \leq t + \frac{1}{2}kt + k$. The first

inequality holds because of the DR-submodularity of the multilinear extension and the third one holds because every node u in the line is active with probability

$$\begin{aligned} & \sum_{i=0}^{\infty} \Pr [u \text{ is activated by its } i^{\text{th}} \text{ predecessor}] \\ & \leq \sum_{i=0}^{\infty} \frac{1}{t} \cdot \left(1 - \frac{1}{t}\right)^i \cdot \left(1 - \frac{1}{t}\right)^i = \frac{1}{t} \cdot \frac{1}{1 - (1 - 1/t)^2} = \frac{t}{2t - 1}. \end{aligned} \quad (38)$$

We conclude that when $t, k \rightarrow \infty$, $f^+(1, 1/t, \dots, 1/t)/F(1, 1/t, \dots, 1/t) \rightarrow 2$.

Random Walk Non-adaptive Strategy. In [16, 17, 8], the authors consider the *adaptive stochastic probing* problem and they convert an adaptive policy to a non-adaptive policy by sampling a random leaf of the decision tree of the adaptive policy. Using our hard instance in the previous paragraph, we can show that this approach (i.e., random walk non-adaptive strategy) can give an upper bound of at most 2. To be more specific, we again consider the adaptive strategy π and its corresponding non-adaptive strategy $\mathcal{W}(\pi)$, where $\mathcal{W}(\pi)$ picks a random leaf of the decision tree of the policy π . We are going to show that $f^+(1, 1/t, \dots, 1/t)/\sigma(\mathcal{W}(\pi))$ approaches 2 asymptotically and it is sufficient to show that $\sigma(\mathcal{W}(\pi)) \leq t + k + \frac{1}{2}kt$. We imagine that node 1 appears in $\mathcal{W}(\pi)$ with probability $1/t$ instead of 1, this is for ease of analysis and it will decrease the influence spread for at most $(1 - 1/t) \cdot t$ due to the submodularity of the influence spread function. For any node u , u is activated by its i^{th} predecessor (if it has one) when (i) the random seed set $\mathcal{W}(\pi)$ does not contain nodes between u and its i^{th} predecessor (this happens with probability $(1 - \frac{1}{t})^i$), (ii) its i^{th} predecessor is included in the seed set (this happens with probability $\frac{1}{t}$) and (iii) node u can be reached from its i^{th} predecessor (this happens with probability $(1 - \frac{1}{t})^i$). Moreover, we know that the above three events are independent in the non-adaptive setting, thus the probability that node u is activated by the i^{th} predecessor is $\frac{1}{t} \cdot (1 - \frac{1}{t})^i \cdot (1 - \frac{1}{t})^i$ and the probability that it is active is no more than $\frac{t}{2t-1}$. This concludes our argument.

Minimum-Width Double-Strip and Parallelogram Annulus

Sang Won Bae 

Division of Computer Science and Engineering, Kyonggi University, Suwon, Korea
swbae@kgu.ac.kr

Abstract

In this paper, we study the problem of computing a minimum-width double-strip or parallelogram annulus that encloses a given set of n points in the plane. A double-strip is a closed region in the plane whose boundary consists of four parallel lines and a parallelogram annulus is a closed region between two edge-parallel parallelograms. We present several first algorithms for these problems. Among them are $O(n^2)$ and $O(n^3 \log n)$ -time algorithms that compute a minimum-width double-strip and parallelogram annulus, respectively, when their orientations can be freely chosen.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases geometric covering, parallelogram annulus, two-line center, double-strip

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.25

Related Version <https://arxiv.org/abs/1911.07504>

Funding Sang Won Bae: Supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2018R1D1A1B07042755).

1 Introduction

The *minimum-width annulus problem* asks to find an annulus of a certain shape with the minimum width that encloses a given set P of n points in the plane. An annulus informally depicts a ring-shaped region in the plane. As the most natural and classical example, a circular annulus is defined to be the region between two concentric circles. If one wants to find a circle that best fits an input point set P , then her problem can be solved by finding out a minimum-width circular annulus that encloses P . After early results on the circular annulus problem [13], the currently best algorithm that computes a minimum-width circular annulus that encloses n input points takes $O(n^{\frac{3}{2}+\epsilon})$ time [3, 4] for any $\epsilon > 0$. Analogously, such a problem of matching a point set into a closed curve class can be formulated into the minimum-width annulus problem for annuli of different shapes.

Along with applications not only to the points-to-curve matching problem but also to other types of facility location, the minimum-width annulus problem has been extensively studied for recent years, with a variety of variations and extensions. Abellanas et al. [1] considered minimum-width rectangular annuli that are axis-parallel, and presented two algorithms taking $O(n)$ or $O(n \log n)$ time: one minimizes the width over rectangular annuli with arbitrary aspect ratio and the other does over rectangular annuli with a prescribed aspect ratio, respectively. Gluchshenko et al. [9] presented an $O(n \log n)$ -time algorithm that computes a minimum-width axis-parallel square annulus, and proved a matching lower bound, while the second algorithm by Abellanas et al. can do the same in the same time bound. If one considers rectangular or square annuli in arbitrary orientation, the problem becomes more difficult. Mukherjee et al. [12] presented an $O(n^2 \log n)$ -time algorithm that computes a minimum-width rectangular annulus in arbitrary orientation and arbitrary aspect ratio. The author [5] showed that a minimum-width square annulus in arbitrary orientation can be computed in $O(n^3 \log n)$ time, and recently improved it to $O(n^3)$ time [6].



© Sang Won Bae;

licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 25; pp. 25:1–25:14

Leibniz International Proceedings in Informatics



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

In this paper, we consider a more generalized shape, namely, parallelograms, and annuli based on them in fixed or arbitrary orientation, which have at least one more degree of freedom than square and rectangular annuli have. More precisely, we define a parallelogram annulus as a closed region between two edge-parallel parallelograms, and address the problem of computing a minimum-width parallelogram annulus that encloses the input point set P . (See Figure 1(c).) We consider several restricted cases of the problem about two orientations of sides of parallelogram annuli. Our main results are summarized as follows:

- (1) When both orientations for sides are fixed, a minimum-width parallelogram annulus that encloses P can be computed in $O(n)$ time.
- (2) When one orientation for sides is fixed and the other can be chosen arbitrarily, a minimum-width parallelogram annulus that encloses P can be computed in $O(n^2)$ time.
- (3) A minimum-width parallelogram annulus that encloses P over all pairs of orientations can be computed in $O(n^3 \log n)$ time.

To obtain these algorithms for the problem, we also introduce another geometric optimization problem, called the *minimum-width double-strip problem*, which asks to compute a double-strip of minimum width that encloses P . A double-strip is defined to be the union of two parallel strips, where a strip is the region between two parallel lines in the plane. (See Figure 1(b).) We show that this new problem is closely related to the parallelogram annulus problem. The minimum-width double-strip problem has its own interest as a special case of the *two-line center problem*, in which one wants to find two strips, possibly being non-parallel, that encloses P and minimizes the width of the wider strip. After the first sub-cubic algorithm is presented by Agarwal and Sharir [2], the currently best algorithm for the two-line center problem takes $O(n^2 \log^2 n)$ time [8, 11].

To our best knowledge, however, no nontrivial result on the double-strip problem is known in the literature. In this paper, we obtain the following algorithmic results:

- (4) A minimum-width double-strip that encloses P over all orientations can be computed in $O(n^2)$ time.
- (5) We also consider a constrained version of the problem in which a subset $Q \subseteq P$ with $k = |Q|$ is given and one wants to find a minimum-width double-strip enclosing Q such that all points of P should lie in between its outer boundary lines. We show that this can be solved in $O(n \log n + kn)$ time.
- (6) We further address some online and offline versions of the dynamic constrained double-strip problem under insertions and/or deletions of a point on the subset Q to enclose.

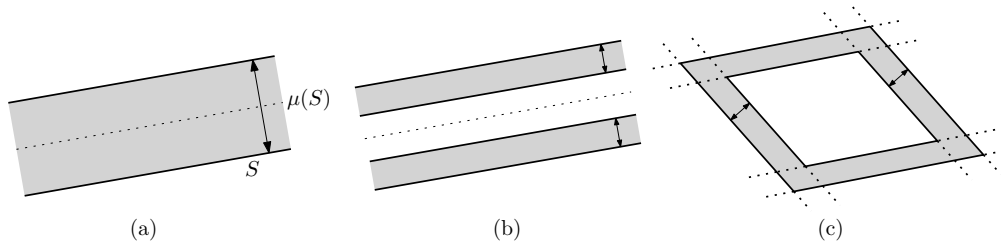
The rest of the paper is organized as follows: Section 2 introduces necessary definitions and preliminaries. Section 3 is devoted to solve the minimum-width double-strip problem and present an $O(n^2)$ time algorithm, which is generalized to the constrained double-strip problem in Section 4. The minimum-width parallelogram annulus problem is finally discussed and solved in Section 5.

Omitted proofs and additional figures will be provided in a full version.

2 Preliminaries

In this section, we introduce definitions of necessary concepts and preliminaries for further discussion. For any subset $A \subseteq \mathbb{R}^2$ of the plane \mathbb{R}^2 , its boundary and interior are denoted by ∂A and $\text{int}A$, respectively.

For two parallel lines ℓ and ℓ' in the plane \mathbb{R}^2 , the *distance* between ℓ and ℓ' denotes the length of any line segment that is orthogonal to ℓ and ℓ' and have endpoints one on ℓ and the other on ℓ' . A *strip* is a closed region bounded by two parallel lines in the plane. For any



■ **Figure 1** Illustrations to (a) a strip S and its middle line $\mu(S)$, (b) a double-strip, and (c) a parallelogram annulus. The arrows depict the width of each shape.

strip S , its *width* $w(S)$ is the distance between its bounding lines, and its *middle line* $\mu(S)$ is the line parallel to its bounding lines such that the distance between $\mu(S)$ and each of the bounding lines is exactly half the width $w(S)$ of S . See Figure 1(a).

A *double-strip* is the union of two disjoint parallel strips of equal width, or equivalently, is a closed region obtained by a strip S subtracted by the interior of another strip S' such that $\mu(S) = \mu(S')$ and $S' \subseteq S$. For any double-strip defined by two strips S and S' in this way, S is called its *outer strip* and S' its *inner strip*. The *width* of such a double-strip D , denoted by $w(D)$, is defined to be half the difference of the widths of S and S' , that is, $w(D) = (w(S) - w(S'))/2$. See Figure 1(b).

A parallelogram is a quadrilateral that is the intersection of two non-parallel strips. We define a parallelogram annulus to be a parallelogram R with a parallelogram hole R' , analogously as a circular annulus is a circle with a circular hole. Here, we add a condition that the outer and inner parallelograms R and R' should be side-wise parallel. There are several ways to define such a parallelogram annulus, among which we introduce the following definition. A *parallelogram annulus* A is defined by two double-strips D_1 and D_2 as follows:

1. The *outer parallelogram* R of A is the intersection of the outer strips of D_1 and D_2 .
2. The *inner parallelogram* R' of A is the intersection of the inner strips of D_1 and D_2 .
3. The parallelogram annulus A is the closed region between R and R' , that is, $A = R \setminus \text{int}R'$.
4. The *width* of A , denoted by $w(A)$, is taken to be the bigger one between the widths of D_1 and D_2 , that is, $w(A) = \max\{w(D_1), w(D_2)\}$.

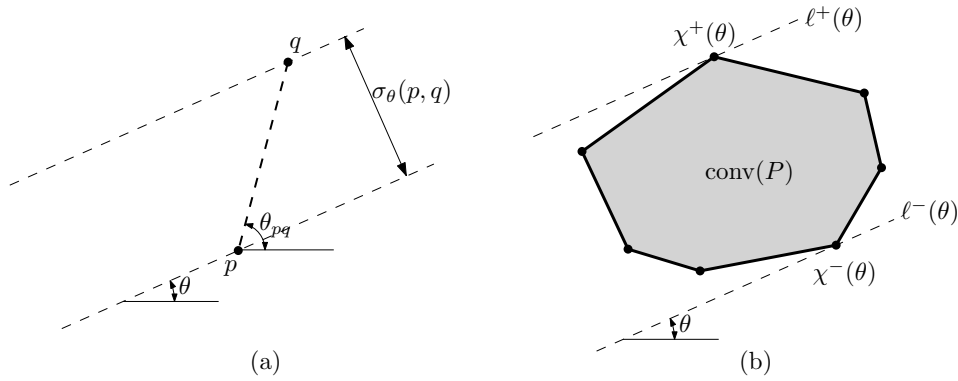
See Figure 1(c) for an illustration.

The main purpose of this paper is to solve the *minimum-width parallelogram annulus problem* in which we are given a set P of points in the plane and want to find a parallelogram annulus of minimum width that encloses P . As discussed above, a parallelogram annulus is closely related to strips and double-strips. The *minimum-width double-strip problem* asks to find a double-strip of minimum width that encloses P in fixed or arbitrary orientation.

The *orientation* of a line or line segment ℓ in the plane is a value $\theta \in [-\pi/2, \pi/2]^1$ such that the rotated copy of the x -axis by θ counter-clockwise is parallel to ℓ . If the orientation of a line or line segment is θ , then we say that the line or line segment is θ -aligned. A strip or a double-strip is also called θ -aligned for some $\theta \in [-\pi/2, \pi/2]$ if its bounding lines are θ -aligned. A parallelogram or a parallelogram annulus is (θ, ϕ) -aligned if it is defined by two double-strips that are θ -aligned and ϕ -aligned, respectively.

For any two points $p, q \in \mathbb{R}^2$, let \overline{pq} denote the line segment joining p and q , and $|\overline{pq}|$ denote the Euclidean length of \overline{pq} . We will often discuss the strip defined by two parallel lines through p and q , and its width. Let $\sigma_\theta(p, q)$ be the width of the strip defined by two θ -aligned lines through p and q , respectively. It is not difficult to see that $\sigma_\theta(p, q) = |\overline{pq}| \cdot |\sin(\theta - \theta_{pq})|$, where $\theta_{pq} \in [-\pi/2, \pi/2)$ denotes the orientation of \overline{pq} . See Figure 2(a).

¹ The orientation θ is indeed of period π . In this paper we choose $[-\pi/2, \pi/2)$ for the orientation domain.



■ **Figure 2** (a) The θ -aligned strip defined by p and q , and its width $\sigma_\theta(p, q)$. (b) The antipodal pair $(\chi^+(\theta), \chi^-(\theta))$ of P for orientation $\theta \in [-\pi/2, \pi/2)$.

A single-variate function of a particular form $a \sin(\theta + b)$ for some constants $a, b \in \mathbb{R}$ with $a \neq 0$ is called *sinusoidal function (of period 2π)*. Obviously, the equation $a \sin(\theta + b) = 0$ has at most one zero over $\theta \in [-\pi/2, \pi/2)$. The following property of sinusoidal functions is well known and easily derived.

► **Lemma 1.** *The sum of two sinusoidal functions is also sinusoidal. Therefore, the graphs of two sinusoidal functions cross at most once over $[-\pi/2, \pi/2)$.*

Note that, taking $\theta \in [-\pi/2, \pi/2)$ as a variable, the function $\sigma_\theta(p, q)$ for fixed points $p, q \in \mathbb{R}^2$ is piecewise sinusoidal with at most one breakpoint.

Let P be a finite set of points in \mathbb{R}^2 , and let $\text{conv}(P)$ be its convex hull. The corners of $\text{conv}(P)$ are called *extreme* of P . For each $\theta \in [-\pi/2, \pi/2)$, let $S(\theta)$ be the minimum-width θ -aligned strip enclosing P . Then, the two lines that define $S(\theta)$ pass through two extreme points of P , one on each. More precisely, let $\ell^+(\theta)$ and $\ell^-(\theta)$ be the two lines defining $S(\theta)$ such that $\ell^+(\theta)$ lies above $\ell^-(\theta)$ if $\theta \neq -\pi/2$, or $\ell^+(\theta)$ lies to the right of $\ell^-(\theta)$ if $\theta = -\pi/2$. There must be an extreme point of P on each of the two lines $\ell^+(\theta)$ and $\ell^-(\theta)$, denoted by $\chi^+(\theta)$ and $\chi^-(\theta)$. If there are two or more such points of P , then we choose the last one in the counter-clockwise order along the boundary of $\text{conv}(P)$. The width of $S(\theta)$ is equal to $\sigma_\theta(\chi^+(\theta), \chi^-(\theta))$.

For each $\theta \in [-\pi/2, \pi/2)$, the pair $(\chi^+(\theta), \chi^-(\theta))$ is called *antipodal*. It is known that there are at most $O(n)$ different antipodal pairs by Toussaint [14]. See Figure 2(b). Starting from $\theta = -\pi/2$, imagine the motion of the two lines $\ell^+(\theta)$ and $\ell^-(\theta)$ as θ continuously increases. Then, the antipodal pair $(\chi^+(\theta), \chi^-(\theta))$ for θ only changes when one of the two lines contains an edge of $\text{conv}(P)$. In this way, the orientation domain $[-\pi/2, \pi/2)$ is decomposed into maximal intervals I such that $(\chi^+(\theta), \chi^-(\theta))$ remains the same in I .

3 Minimum-Width Double-Strips

In this section, we address the problem of computing a minimum-width double-strip that encloses a given set P of n points in \mathbb{R}^2 .

We start with the problem in a given orientation $\theta \in [-\pi/2, \pi/2)$. Let $w(\theta)$ be the minimum possible width of a θ -aligned double-strip enclosing P . The following observation can be obtained by a simple geometric argument.

► **Observation 2.** For each $\theta \in [-\pi/2, \pi/2)$, there exists a minimum-width θ -aligned double-strip $D(\theta)$ enclosing P whose outer strip is $S(\theta)$ and inner strip is $S'(\theta)$, where

- $S(\theta)$ is the minimum-width θ -aligned strip enclosing P , and
- $S'(\theta)$ is the maximum-width θ -aligned strip such that its interior is empty of any point in P and $\mu(S'(\theta)) = \mu(S(\theta))$.

Proof. Let D be a minimum-width θ -aligned double-strip enclosing P . Let S_1 and S_2 be the two strips such that $D = S_1 \cup S_2$. If the boundary of S_1 does not contain an extreme point of P , then we can slide S_1 inwards until its boundary hits an extreme point of P . Let S'_1 be the resulting strip after this sliding process. Then, we have $S_1 \cap P \subset S'_1$. In the same way, we slide S_2 until the boundary of S_2 hits an extreme point of P , and let S'_2 be the resulting strip. Then, it holds that $S_2 \cap P \subset S'_2$. As a result, $P \subset (S'_1 \cup S'_2)$ since $P \subset (S_1 \cup S_2)$. Note that the outer strip of the double-strip $D' := S'_1 \cup S'_2$ is exactly $S(\theta)$, the minimum-width θ -aligned strip enclosing P .

Now, let S' be the inner strip of D' . By definition, we have $\mu(S(\theta)) = \mu(S')$. Suppose that the inner strip S' of D' is not equal to $S'(\theta)$, the maximum-width θ -aligned strip such that its interior is empty of any point in P and $\mu(S'(\theta)) = \mu(S(\theta))$. Then, the boundary of S' does not contain any point of P . Hence, the width of $S'(\theta)$ is strictly larger than the width of S' , a contradiction that D' is of minimum width. Therefore, the inner strip of D' should be $S'(\theta)$. ◀

We focus on finding the minimum-width double-strip $D(\theta)$ described in Observation 2. The outer strip $S(\theta)$ of $D(\theta)$ is determined by $\ell^+(\theta)$ and $\ell^-(\theta)$ on which the two extreme points $\chi^+(\theta)$ and $\chi^-(\theta)$ lie. For $p \in P$, let $d_p(\theta) := \min\{\sigma_\theta(p, \chi^+(\theta)), \sigma_\theta(p, \chi^-(\theta))\}$. Then, the width $w(\theta)$ of $D(\theta)$ in orientation θ is determined by

$$w(\theta) = \max_{p \in P} d_p(\theta).$$

It is not difficult to see that $w(\theta)$ can be evaluated in $O(n)$ time for a given $\theta \in [-\pi/2, \pi/2)$.

► **Theorem 3.** Given a set P of n points and an orientation $\theta \in [-\pi/2, \pi/2)$, a minimum-width θ -aligned double-strip enclosing P and its width $w(\theta)$ can be computed in $O(n)$ time.

Proof. In this proof, we describe our algorithm that computes $D(\theta)$. Given θ , one can compute the two extreme points $\chi^+(\theta)$ and $\chi^-(\theta)$ in $O(n)$ time by computing the minimum and maximum among the inner products of

$$(\cos(\theta + \pi/2), \sin(\theta + \pi/2))$$

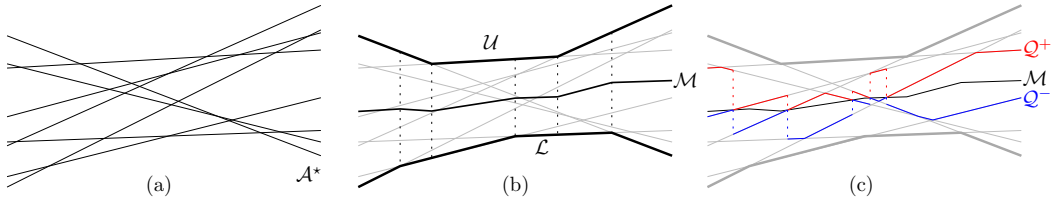
and all points in P as vectors. The antipodal pair also determines the outer strip $S(\theta)$.

After identifying $\chi^+(\theta)$ and $\chi^-(\theta)$, the value of $d_p(\theta)$ for each $p \in P$ can be computed in $O(1)$ time. Hence, $w(\theta) = \max_{p \in P} d_p(\theta)$ can be found in additional $O(n)$ time. This determines the inner strip $S'(\theta)$. ◀

Next, we turn to finding a minimum-width double-strip over all orientations. This is equivalent to computing the minimum value of $w(\theta)$ over $\theta \in [-\pi/2, \pi/2)$, denoted by w^* . Let θ^* be an optimal orientation such that $w(\theta^*) = w^*$. Consider the corresponding double-strip $D(\theta^*)$ whose outer strip is $S(\theta^*)$ and inner strip is $S'(\theta^*)$, as described in Observation 2. We then observe the following for the minimum-width double-strip $D(\theta^*)$ enclosing P .

► **Lemma 4.** Let θ^* be an orientation such that $w(\theta^*) = w^*$. Then, either

- (a) three extreme points of P lie on the boundary of $S(\theta^*)$, or
- (b) two points of P lie on the boundary of $S'(\theta^*)$.



■ **Figure 3** (a) The arrangement \mathcal{A}^* of lines in P^* . (b) The upper envelope \mathcal{U} , the lower envelope \mathcal{L} , and \mathcal{M} of \mathcal{A}^* . (c) \mathcal{Q}^+ and \mathcal{Q}^- depicted by red and blue chains, respectively.

Regard $d_p(\theta)$ as a function of θ in domain $[-\pi/2, \pi/2]$. This function depends on the antipodal pair of extreme points $(\chi^+(\theta), \chi^-(\theta))$ for θ . Since there are $O(n)$ antipodal pairs [14], the function d_p for each $p \in P$ is piecewise sinusoidal with $O(n)$ breakpoints. The width function $w(\theta)$ is the upper envelope of the n functions $d_p(\theta)$ for $p \in P$, consisting of $O(n^2)$ sinusoidal curves in total. Thus, we can compute the minimum width $w^* = \min_{\theta \in [-\pi/2, \pi/2]} w(\theta)$ by computing the upper envelope of these $O(n^2)$ sinusoidal curves and finding a lowest point in the envelope. Since any two sinusoidal curves cross at most once by Lemma 1, this can be done in $O(n^2 \log n)$ time [10]. The width function w is piecewise sinusoidal with $O(n^2 \alpha(n))$ breakpoints, where α denotes the inverse Ackermann function, and its lowest point always occurs at one of the breakpoints by Lemma 4. Hence, $O(n^2 \log n)$ time is sufficient to solve the problem.

In the following, we improve this to $O(n^2)$ time. As observed above, the double-strip $D(\theta)$ of width $w(\theta)$ is determined by its outer strip $S(\theta)$ and inner strip $S'(\theta)$. Let $\mu(\theta) := \mu(S(\theta)) = \mu(S'(\theta))$ be the middle line of $D(\theta)$. The middle line $\mu(\theta)$ separates P into two subsets $P^+(\theta)$ and $P^-(\theta)$, where $P^+(\theta)$ is the set of points $p \in P$ lying in the strip defined by $\ell^+(\theta)$ and $\mu(\theta)$, and $P^-(\theta) = P \setminus P^+(\theta)$. Define $q^+(\theta) \in P^+(\theta)$ to be the closest point to line $\mu(\theta)$ among $P^+(\theta)$, and similarly $q^-(\theta) \in P^-(\theta)$ to be the closest point to $\mu(\theta)$ among $P^-(\theta)$. We then observe that the width of $D(\theta)$ is

$$w(\theta) = \max\{\sigma_\theta(q^+(\theta), \chi^+(\theta)), \sigma_\theta(q^-(\theta), \chi^-(\theta))\}.$$

Hence, $D(\theta)$ is completely determined by these four points: $\chi^+(\theta)$, $\chi^-(\theta)$, $q^+(\theta)$, and $q^-(\theta)$.

In order to analyze and specify the change of pair $(q^+(\theta), q^-(\theta))$ as θ continuously increases from $-\pi/2$ to $\pi/2$, we adopt an interpretation under a geometric dualization [7, Chapter 8]. We shall call the plane \mathbb{R}^2 – in which we have discussed objects so far – the *primal plane* with the x - and y -axes. Let \mathbb{D} be another plane, called the *dual plane*, with u - and v -axes that correspond to its horizontal and vertical axes, respectively. Here, we use a standard duality transform $*$ which maps a point $p = (a, b) \in \mathbb{R}^2$ into a line $p^*: v = au - b \subset \mathbb{D}$ and a non-vertical line $\ell: y = ax - b \subset \mathbb{R}^2$ into a point $\ell^* = (a, b) \in \mathbb{D}$. This duality transform is also defined in the reversed way for points and lines in the dual plane \mathbb{D} to be mapped to lines and points in the primal plane \mathbb{R}^2 , so that we have $(p^*)^* = p$ and $(\ell^*)^* = \ell$ for any point p and any non-vertical line ℓ either in \mathbb{R}^2 or in \mathbb{D} . We say that a geometric object and its image under the duality transform are *dual* to each other. Note that p lies above (on or below, resp.) ℓ if and only if ℓ^* lies above (on or below, resp.) p^* .

3.1 Scenes from the dual plane

Suppose that the input point set P is given in the primal plane \mathbb{R}^2 . Consider the set $P^* := \{p^* \mid p \in P\}$ of n lines in the dual plane \mathbb{D} and their arrangement $\mathcal{A}^* := \mathcal{A}(P^*)$. See Figure 3 for illustration. Let \mathcal{U} and \mathcal{L} be the upper and lower envelopes in \mathcal{A}^* . The

envelopes \mathcal{U} and \mathcal{L} can also be considered as two functions of $u \in \mathbb{R}$; in this way, $\mathcal{U}(u)$ and $\mathcal{L}(u)$ are the v -coordinates in \mathbb{D} of points on \mathcal{U} and \mathcal{L} , respectively, at $u \in \mathbb{R}$. Let $\mathcal{M}(u) := (\mathcal{U}(u) + \mathcal{L}(u))/2$ be the v -coordinates of the midpoint of the vertical segment connecting \mathcal{L} and \mathcal{U} at $u \in \mathbb{R}$. Similarly, we regard \mathcal{M} as the function itself and simultaneously as its graph $\mathcal{M} = \{(u, \mathcal{M}(u)) \mid u \in \mathbb{R}\}$ drawn in \mathbb{D} .

As well known, the upper envelope \mathcal{U} corresponds to the lower chain of $\text{conv}(P)$ and the lower envelope \mathcal{L} to its upper chain. More precisely, each vertex of \mathcal{U} and \mathcal{L} is dual to the line supporting an edge of $\text{conv}(P)$. Thus, the total number of vertices of \mathcal{U} and \mathcal{L} is no more than $n = |P|$. Also, observe that the number of vertices of \mathcal{M} is equal to the total number of vertices of \mathcal{U} and \mathcal{L} by definition.

Now, consider the portions of lines in P^* above \mathcal{M} and the lower envelope of those pieces cut by \mathcal{M} , denoted by \mathcal{Q}^+ . Analogously, let \mathcal{Q}^- be the upper envelope of portions of lines in P^* below \mathcal{M} . The following observations follow directly from the basic properties of duality.

► **Observation 5.** *For each $\theta \in [-\pi/2, \pi/2)$, let $u := \tan \theta$. Then, the following hold:*

- (1) *The dual $(\ell^+(\theta))^*$ of the θ -aligned line through $\chi^+(\theta)$ is the point $(u, \mathcal{L}(u))$ in \mathbb{D} . Similarly, we have $(\ell^-(\theta))^* = (u, \mathcal{U}(u))$.*
- (2) *The dual $(\mu(\theta))^*$ of the middle line $\mu(\theta)$ is the point $(u, \mathcal{M}(u))$ in \mathbb{D} .*
- (3) *The dual of the θ -aligned line through $q^+(\theta)$ is the point $(u, \mathcal{Q}^-(u))$ in \mathbb{D} . Similarly, the dual of the θ -aligned line through $q^-(\theta)$ is the point $(u, \mathcal{Q}^+(u))$.*

From Observation 5, one would say informally that ℓ^+ is dual to \mathcal{L} , ℓ^- to \mathcal{U} , μ to \mathcal{M} , q^+ to \mathcal{Q}^- , and q^- to \mathcal{Q}^+ .

We are ready to describe our algorithm. We first compute the arrangement \mathcal{A}^* in $O(n^2)$ time. The envelopes \mathcal{U} and \mathcal{L} can be traced in $O(n)$ time from \mathcal{A}^* , and we also can compute \mathcal{M} in $O(n)$ time. We then compute \mathcal{Q}^+ and \mathcal{Q}^- .

► **Lemma 6.** *The complexity of \mathcal{Q}^+ and \mathcal{Q}^- is $O(n^2)$, and we can compute them in $O(n^2)$ time.*

By Lemma 6 together with Observation 5, we know that there are $O(n^2)$ changes in pair $(q^+(\theta), q^-(\theta))$ as θ increases from $-\pi/2$ to $\pi/2$. On the other hand, we already know that the antipodal pair $(\chi^+(\theta), \chi^-(\theta))$ changes $O(n)$ times as θ increases from $-\pi/2$ to $\pi/2$. Consequently, there are $O(n^2)$ changes in tuple $(\chi^+(\theta), \chi^-(\theta), q^+(\theta), q^-(\theta))$, and thus the orientation domain $[-\pi/2, \pi/2)$ is decomposed into $O(n^2)$ intervals in each of which the tuple is fixed. For each such interval I , we minimize $w(\theta) = \max\{\sigma_\theta(q^+(\theta), \chi^+(\theta)), \sigma_\theta(q^-(\theta), \chi^-(\theta))\}$ over $\theta \in I$. Since the four points $\chi^+(\theta), \chi^-(\theta), q^+(\theta), q^-(\theta)$ are fixed in I , the function w on I is the upper envelope of at most four sinusoidal functions by Lemma 1. By Lemma 4, the minimum occurs either (a) at an endpoint of I or (b) when the equality $\sigma_\theta(q^+(\theta), \chi^+(\theta)) = \sigma_\theta(q^-(\theta), \chi^-(\theta))$ holds. Hence, we can minimize $w(\theta)$ over $\theta \in I$ in $O(1)$ time. Since $w^* = \min_I \min_{\theta \in I} w(\theta)$, we can compute w^* by taking the minimum over such intervals I .

► **Theorem 7.** *Given a set P of n points in the plane, a minimum-width double-strip enclosing P can be computed in $O(n^2)$ time.*

4 Constrained Double-Strip Problem

In this section, we discuss a constrained version of the minimum-width double-strip problem, called the *constrained double-strip problem*. The constrained double-strip problem has its own interest, while it can also be used, in particular, to obtain efficient algorithms for the parallelogram annulus problem, which will be discussed in the following section.

In the constrained double-strip problem, we are given a set P of n points and a subset $Q \subseteq P$ with $k = |Q|$. A P -constrained double-strip is a double-strip whose outer strip contains all points in P . Then, the problem asks to find a P -constrained double-strip of minimum width that encloses subset Q .

Analogously to Observation 2, we observe the following for the constrained problem.

► **Observation 8.** *For each $\theta \in [-\pi/2, \pi/2)$, there exists a minimum-width θ -aligned P -constrained double-strip $D_Q(\theta)$ enclosing Q such that its outer strip is $S(\theta)$ and its inner strip is $S'_Q(\theta)$, where $S'_Q(\theta)$ is the maximum-width θ -aligned strip such that its interior is empty of any point in Q and $\mu(S'_Q(\theta)) = \mu(\theta)$.*

Let $w_Q(\theta)$ be the width of the minimum-width P -constrained double-strip $D_Q(\theta)$ enclosing Q described in Observation 8, and $w_Q^* := \min_{\theta \in [-\pi/2, \pi/2)} w_Q(\theta)$. Hence, we focus on computing the minimum possible width w_Q^* and its corresponding double-strip $D_Q(w_Q^*)$. We also redefine $q^+(\theta)$ and $q^-(\theta)$ to be the closest points to $\mu(\theta)$ among points in Q above and below $\mu(\theta)$, respectively. Then, we have

$$w_Q(\theta) = \max\{\sigma_\theta(q^+(\theta), \chi^+(\theta)), \sigma_\theta(q^-(\theta), \chi^-(\theta))\}.$$

The following lemma is analogous to Lemma 4.

► **Lemma 9.** *Let θ^* be an orientation such that $w_Q(\theta^*) = w_Q^*$. Then, either*

- (a) *three extreme points of P lie on the boundary of $S(\theta^*)$, or*
- (b) *two points of Q lie on the boundary of $S'_Q(\theta^*)$.*

To solve the constrained problem, we extend our approach for the unconstrained problem. Define $\mathcal{A}_Q^* := \mathcal{A}(Q^* \cup \mathcal{U} \cup \mathcal{L})$ to be the arrangement of k lines in $Q^* = \{p^* \mid p \in Q\}$ plus the envelopes \mathcal{U} and \mathcal{L} of $\mathcal{A}(P^*)$. Let \mathcal{Q}_Q^+ be the lower envelope of portions of lines in Q^* above \mathcal{M} , and \mathcal{Q}_Q^- be the upper envelope of portions of lines in Q^* below \mathcal{M} .

Our algorithm for the constrained double-strip problem runs as follows: First, compute the convex hull $\text{conv}(P)$ and extract \mathcal{U} and \mathcal{L} from $\text{conv}(P)$ in $O(n \log n)$ time. Then, we add lines in Q^* incrementally one by one to build \mathcal{A}_Q^* . Since every line in Q^* lies between \mathcal{U} and \mathcal{L} , this takes additional $O(k^2)$ time. Next, we compute \mathcal{Q}_Q^+ and \mathcal{Q}_Q^- .

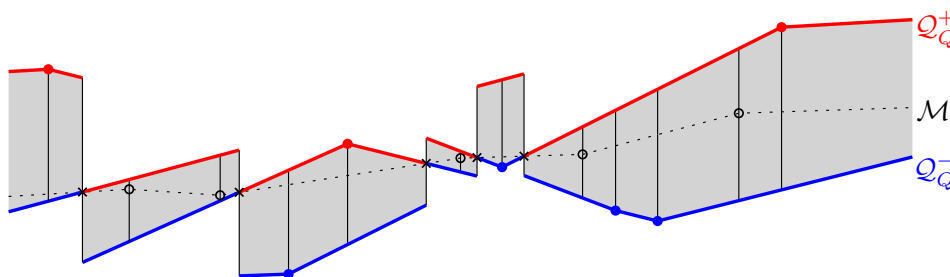
► **Lemma 10.** *The complexity of \mathcal{Q}_Q^+ and \mathcal{Q}_Q^- is $O(kn)$, and we can compute them in $O(kn)$ time.*

The rest of the algorithm is the same as that described in the previous section. As we increase $\theta \in [-\pi/2, \pi/2)$ continuously, there are $O(kn)$ changes in tuple $(\chi^+(\theta), \chi^-(\theta), q^+(\theta), q^-(\theta))$ by Lemma 10, and thus the orientation domain $[-\pi/2, \pi/2)$ is decomposed into $O(kn)$ intervals in each of which the four points are fixed. For each such interval I , we minimize $w_Q(\theta) = \max\{\sigma_\theta(q^+(\theta), \chi^+(\theta)), \sigma_\theta(q^-(\theta), \chi^-(\theta))\}$ over $\theta \in I$. By Lemma 9, the minimum occurs either (a) at an endpoint of I or (b) when the equality $\sigma_\theta(q^+(\theta), \chi^+(\theta)) = \sigma_\theta(q^-(\theta), \chi^-(\theta))$ holds. Hence, we can minimize $w(\theta)$ over $\theta \in I$ in $O(1)$ time, and obtain the following result.

► **Theorem 11.** *Given a set P of n points and a subset $Q \subseteq P$ with $k = |Q|$, a minimum-width P -constrained double-strip enclosing Q can be computed in $O(n \log n + kn)$ time.*

4.1 Dynamic maintenance under insertion and deletion

In the following, we consider a dynamic situation in which a point in P can be inserted into or deleted from Q . Our goal is to report a minimum-width P -constrained double-strip enclosing Q over all orientations and its width, that is, w_Q^* and $D_Q(w_Q^*)$, whenever a change in Q occurs, faster than computing from scratch by Theorem 11.



■ **Figure 4** Illustration to the trapezoidal map \mathcal{T}_Q with 18 trapezoids. In this example, we take $Q = P$ for input point set P , being the same as the one used as in Figure 3. Red and blue dots are vertices of \mathcal{A}_Q^* that belong to \mathcal{Q}_Q^+ and \mathcal{Q}_Q^- , respectively. Small circles depict the vertices of \mathcal{M} and cross marks are the intersections between \mathcal{M} and any line p^* for $p \in Q$.

For the purpose, we keep the following invariants updated under insertions and deletions: the arrangement \mathcal{A}_Q^* and a trapezoidal map \mathcal{T}_Q . The map \mathcal{T}_Q is a vertical trapezoidal decomposition of the region between \mathcal{Q}_Q^+ and \mathcal{Q}_Q^- . More precisely, let \mathcal{Q}_Q be the region between \mathcal{Q}_Q^+ and \mathcal{Q}_Q^- , and $\{u_1, u_2, \dots, u_m\}$ be the set of u -coordinates of the vertices of \mathcal{Q}_Q^+ , \mathcal{Q}_Q^- , and \mathcal{M} such that $u_1 < u_2 < \dots < u_m$. By adding into \mathcal{Q}_Q a vertical line segment at $u = u_i$ for each $i = 1, 2, \dots, m$ between two points $(u_i, \mathcal{Q}_Q^-(u_i))$ and $(u_i, \mathcal{Q}_Q^+(u_i))$, we obtain the trapezoidal decomposition \mathcal{T}_Q consisting of $m + 1$ trapezoids. For convenience, let $u_0 := -\infty$ and $u_{m+1} := \infty$. The order of trapezoids in \mathcal{T}_Q is naturally induced along the u -axis in \mathbb{D} . The i -th trapezoid τ in \mathcal{T}_Q for $i = 1, \dots, m + 1$ is bounded by two vertical segments at $u = u_{i-1}$ and $u = u_i$, and two segments from \mathcal{Q}_Q^+ and \mathcal{Q}_Q^- . See Figure 4 for an illustration. The two segments of τ from \mathcal{Q}_Q^+ and \mathcal{Q}_Q^- are called the *ceiling* and *floor* of τ , respectively. Let $U_\tau \subset \mathbb{R}$ be an interval consisting of the u -coordinates of all points in τ .

At each trapezoid τ , we store four points $\chi_\tau^+, \chi_\tau^-, q_\tau^+, q_\tau^- \in P$ such that $\chi_\tau^+ = \chi^+(\theta)$, $\chi_\tau^- = \chi^-(\theta)$, $q_\tau^+ = q^+(\theta)$, and $q_\tau^- = q^-(\theta)$ for all θ with $\tan \theta \in U_\tau$. We also store at τ the value $w_\tau := \min_{\tan \theta \in U_\tau} \{\max\{\sigma_\theta(q_\tau^+, \chi_\tau^+), \sigma_\theta(q_\tau^-, \chi_\tau^-)\}\}$, which can be computed in $O(1)$ time per trapezoid by Lemma 9.

Note that the union of ceilings of all trapezoids in \mathcal{T}_Q forms \mathcal{Q}_Q^+ , and the union of their floors forms \mathcal{Q}_Q^- . By Lemma 10, the number $m + 1$ of trapezoids in \mathcal{T}_Q is $O(|Q|n)$. More importantly, for each $\theta \in [-\pi/2, \pi/2)$, there is a unique trapezoid τ in \mathcal{T}_Q such that $\tan \theta$ lies in interval U_τ of τ and thus we have $w_Q(\theta) = \max\{\sigma_\theta(q_\tau^+, \chi_\tau^+), \sigma_\theta(q_\tau^-, \chi_\tau^-)\}$. This implies that $\min\{w_Q(\theta) \mid \tan \theta \in U_\tau\} = w_\tau$, and hence $w_Q^* = \min_{\tau \in \mathcal{T}_Q} w_\tau$. Thus, by efficiently maintaining \mathcal{T}_Q , the problem is reduced to finding the minimum of w_τ over all trapezoids τ in \mathcal{T}_Q .

Updating our invariants can be done in $O(n)$ time thanks to the Zone Theorem for the arrangement of lines.

► **Lemma 12.** *Let $Q \subseteq P$, $p \in P$, and Q' be either $Q \cup \{p\}$ or $Q \setminus \{p\}$. Then, $\mathcal{A}_{Q'}^*$ and $\mathcal{T}_{Q'}$ can be obtained in $O(n)$ time, provided the description of \mathcal{A}_Q^* and \mathcal{T}_Q . More specifically, the number of trapezoids in the symmetric difference $(\mathcal{T}_{Q'} \setminus \mathcal{T}_Q) \cup (\mathcal{T}_Q \setminus \mathcal{T}_{Q'})$ is $O(n)$, and those trapezoids can be identified in the same time bound.*

Now, we are ready to describe our overall algorithm. We assume that we start with an empty set $Q = \emptyset$ and a sequence of insertions and deletions on Q is given. For $Q = \emptyset$, it is easy to initialize \mathcal{A}_\emptyset^* and \mathcal{T}_\emptyset , after computing \mathcal{U} , \mathcal{L} , and \mathcal{M} in $O(n \log n)$ time as described above. Namely, \mathcal{A}_\emptyset^* is just the union of \mathcal{U} and \mathcal{L} , and \mathcal{T}_\emptyset can be obtained from the fact that $\mathcal{Q}_\emptyset^+ = \mathcal{U}$ and $\mathcal{Q}_\emptyset^- = \mathcal{L}$. Whenever an insertion or deletion of a point is given, we update our

25:10 Double-Strip and Parallelogram Annulus

invariants as described in Lemma 12, spending $O(n)$ time. We then report the minimum possible width w_Q^* for the current subset Q and the corresponding double-strip. Since \mathcal{T}_Q consists of $O(|Q|n)$ trapezoids by Lemma 10, a linear scan of \mathcal{T}_Q is too costly. We instead use a basic priority queue, such as the binary heap, and conclude the following.

► **Theorem 13.** *Let P be a set of n points, and $Q_0 = \emptyset, Q_1, Q_2, \dots$ be a sequence of subsets of P such that the difference between Q_{i+1} and Q_i is only a single point in P . Suppose that each Q_i is given at time i by its difference from Q_{i-1} . Then, whenever Q_i is specified for each $i \geq 0$, we can exactly compute a P -constrained double-strip of minimum width $w_{Q_i}^*$ that encloses Q_i in $O(n \log n)$ time.*

If one only wants to decide whether or not the minimum possible width w_Q^* is at least a given target value $w \geq 0$, then the complexity can be reduced as follows.

► **Theorem 14.** *Let P be a set of n points, and $Q_0 = \emptyset, Q_1, Q_2, \dots$ be a sequence of subsets of P such that the difference between Q_{i+1} and Q_i is only a single point in P . Suppose that each Q_i is given at time i by its difference from Q_{i-1} . Let $w \geq 0$ be a given fixed real number. Then, after spending $O(n \log n)$ time for preprocessing, whenever Q_i is specified for each $i \geq 0$, we can decide whether $w \geq w_{Q_i}^*$ or $w < w_{Q_i}^*$ in $O(n)$ time.*

4.2 Offline version under insertions only

Note that the above theorems give us solutions to the online optimization and decision versions of the P -constrained double-strip problem under insertions and deletions. Here, we consider the offline optimization version of the problem under insertions only.

Let $Q = \{p_1, p_2, \dots, p_k\} \subseteq P$ be a subset of P , and $Q_i := \{p_1, \dots, p_i\}$ for $i = 0, \dots, k$. Suppose that we know Q_i for each $i = 0, \dots, k$ for the first time and want to compute a minimum-width P -constrained double-strip enclosing Q_i for all $i = 0, \dots, k$.

For the purpose, we observe the following.

► **Lemma 15.** *For each $i = 0, \dots, k-1$, it holds that $w_{Q_i}^* = \min\{w_{Q_{i+1}}^*, \min_{\tau \in T_i} w_\tau\}$, where $T_i := \mathcal{T}_{Q_i} \setminus \mathcal{T}_{Q_{i+1}}$ denotes the set of trapezoids removed from \mathcal{T}_{Q_i} by the insertion of p_{i+1} .*

Lemma 15 suggests computing $w_{Q_i}^*$ backwards from $i = k$ to $i = 0$. By maintaining \mathcal{T}_{Q_i} from $i = 0$ to k and storing the sets $T_i = \mathcal{T}_{Q_i} \setminus \mathcal{T}_{Q_{i+1}}$, this can be done in $O(kn)$ time.

More precisely, we first build $\mathcal{A}_{Q_0}^*$ and \mathcal{T}_{Q_0} as described above in $O(n \log n)$ time. We then insert p_i for $i = 1, \dots, k$, one by one, and compute $\mathcal{A}_{Q_i}^*$ and \mathcal{T}_{Q_i} in $O(n)$ time per insertion by Lemma 12, but we do not compute the minimum width $w_{Q_i}^*$ at every insertion. Instead, we collect all trapezoids τ that have been deleted, that is, the set $T_i = \mathcal{T}_{Q_i} \setminus \mathcal{T}_{Q_{i+1}}$. Then, we apply Lemma 15 to compute $w_{Q_i}^*$ for each $i = 0, \dots, k$ and its corresponding double-strip.

We first compute $w_{Q_k}^* = \min_{\tau \in \mathcal{T}_{Q_k}} w_\tau$. Then, we iterate i from $k-1$ to 0 , and compute $w_{Q_i}^*$ based on Lemma 15. Since $|T_i| = O(n)$ for each i by Lemma 12, this takes $O(kn)$ additional time. We thus conclude the following theorem.

► **Theorem 16.** *Let P be a set of n points and $p_1, \dots, p_k \in P$ be $k \geq 1$ points in P , and let $Q_i := \{p_1, \dots, p_i\}$ for $0 \leq i \leq k$. Then, in time $O(n \log n + kn)$, we can exactly compute $w_{Q_i}^*$ for all $0 \leq i \leq k$ and corresponding P -constrained double-strips of width $w_{Q_i}^*$ enclosing Q_i .*

5 Minimum-Width Parallelogram Annuli

In this section, we present algorithms that compute a minimum-width parallelogram annulus that encloses a set P of n points in \mathbb{R}^2 . As introduced in Section 2, a parallelogram annulus is defined by two double-strips and its orientation is represented by a pair of parameters (θ, ϕ) with $\theta, \phi \in [-\pi/2, \pi/2)$.

Here, we consider several cases depending on how many of the two side orientations, θ and ϕ , are fixed or not. The easiest case is certainly when both θ and ϕ are fixed.

► **Observation 17.** *For any $\theta, \phi \in [-\pi/2, \pi/2)$, there exists a minimum-width (θ, ϕ) -aligned parallelogram annulus that encloses P such that its outer parallelogram $R(\theta, \phi)$ is the intersection of $S(\theta)$ and $S(\phi)$, the minimum-width θ -aligned and ϕ -aligned strip enclosing P , respectively.*

The above observation gives us a structural property of an optimal annulus which we should look for, and leads to a linear-time algorithm.

► **Theorem 18.** *Given a set P of n points and $\theta, \phi \in [-\pi/2, \pi/2)$, a minimum-width (θ, ϕ) -aligned parallelogram annulus that encloses P can be computed in $O(n)$ time.*

Proof. Here, we describe an algorithm that finds a minimum-width (θ, ϕ) -aligned parallelogram annulus enclosing P whose outer parallelogram is $R(\theta, \phi) = S(\theta) \cap S(\phi)$, as described in Observation 17.

As described in the proof of Theorem 3, we can find the minimum-width θ -aligned strip $S(\theta)$ enclosing P in $O(n)$ time by identifying the corresponding antipodal pair $(\chi^+(\theta), \chi^-(\theta))$. In the same way, we identify the antipodal pair $(\chi^+(\phi), \chi^-(\phi))$ and strip $S(\phi)$.

We then need to find an inner parallelogram R' that minimizes the width of the resulting annulus defined by $R(\theta, \phi)$ and R' . For the purpose, we are done by checking the distance from each $p \in P$ to the boundary of R , which is equal to

$$\min\{w_\theta(p, \chi^+(\theta)), w_\theta(p, \chi^-(\theta)), w_\phi(p, \chi^+(\phi)), w_\phi(p, \chi^-(\phi))\}.$$

We can evaluate this in $O(1)$ time for each $p \in P$, so in $O(n)$ total time, and take the maximum over them, denoted by z . Since the interior of the inner parallelogram R' must avoid all points in P , at least one side of R' must be z distant from the boundary of R . Hence, the minimum width of a (θ, ϕ) -aligned parallelogram annulus enclosing P is exactly z . The value of z and a corresponding annulus can be computed in $O(n)$ time. ◀

In the following, let $w(\theta, \phi)$ be the smallest among the widths of all (θ, ϕ) -aligned parallelogram annuli enclosing P .

5.1 When one side orientation is fixed

Next, we consider the problem where one side of a resulting annulus should be ϕ -aligned for a fixed orientation $\phi \in [-\pi/2, \pi/2)$ while the other orientation parameter θ can be chosen arbitrarily. So, in the following, we regard $\phi \in [-\pi/2, \pi/2)$ to be fixed. Without loss of generality, we assume that $\phi = 0$.

From the definition of a parallelogram annulus A , it is defined by two double-strips. In addition, Observation 17 tells us that the two double-strips defining A can be chosen among the P -constrained double-trips enclosing a subset of P . Hence, for the case where $\phi = 0$ is fixed, the problem is reduced to find a best bipartition of P such that one part is covered by a 0-aligned P -constrained double-strip and the other by another P -constrained double-strip in any orientation $\theta \in [-\pi/2, \pi/2)$.

25:12 Double-Strip and Parallelogram Annulus

We first identify two extreme points $\chi^+(0)$ and $\chi^-(0)$, and the strip $S(0)$ in $O(n)$ time. Then, sort the points in P in the non-increasing order of the value

$$d_p(0) = \min\{\sigma_0(p, \chi^+(0)), \sigma_0(p, \chi^-(0))\}$$

for each $p \in P$, which is the distance to the boundary of $S(0)$. Let $p_1, p_2, \dots, p_{n-1}, p_n \in P$ be this order. Also, let $w_i := d_{p_i}(0) = \min\{\sigma_0(p_i, \chi^+(0)), \sigma_0(p_i, \chi^-(0))\}$ for $i = 1, \dots, n$.

Consider the double-strip D_i with width w_i and outer strip $S(0)$. The double-strip D_1 encloses all points of P , while D_2 misses one point p_1 and D_i misses $i - 1$ points p_1, \dots, p_{i-1} in general for $i = 1, \dots, n$. This means that there are only n different subsets of P covered by any 0-aligned double-strip. Thus, to enclose P by a $(\theta, 0)$ -aligned parallelogram annulus, the other double-strip with orientation θ should cover the rest of the points in P .

Note that each D_i is a minimum-width 0-aligned P -constrained double-strip that encloses $\{p_i, p_{i+1}, \dots, p_n\} \subseteq P$. Let $Q_i := \{p_1, p_2, \dots, p_{i-1}\}$ for $i = 1, \dots, n$. If we choose D_i for the 0-aligned double-strip, then $P \setminus Q_i \subset D_i$, so the points in Q_i should be covered by the second double-strip that define a parallelogram annulus. Let D'_i be the minimum-width P -constrained double-strip enclosing Q_i , and let w'_i be its width. We compute D'_i and w'_i for all $i \in \{1, \dots, n\}$ by applying Theorem 16 in $O(n^2)$ time. What remains is taking the minimum of $\max\{w_i, w'_i\}$ over $i = 1, \dots, n$.

► **Theorem 19.** *Given a set P of n points and a fixed orientation $\phi \in [-\pi/2, \pi/2)$, a (θ, ϕ) -aligned parallelogram annulus of minimum width over all $\theta \in [-\pi/2, \pi/2)$ that encloses P can be computed in $O(n^2)$ time.*

5.2 General case

Finally, we consider the general case where both θ and ϕ can be freely chosen from domain $[-\pi/2, \pi/2)$. Let $w^* := \min_{\theta, \phi \in [-\pi/2, \pi/2)} w(\theta, \phi)$ be the minimum possible width, and (θ^*, ϕ^*) be a pair of orientations such that $w^* = w(\theta^*, \phi^*)$.

We first consider the decision version of the problem in which a positive real number $w > 0$ is given and we want to decide if $w \geq w^*$ or $w < w^*$. For the purpose, we consider the function d_p defined above for each $p \in P$ to be $d_p(\theta) = \min\{\sigma_\theta(p, \chi^+(\theta)), \sigma_\theta(p, \chi^-(\theta))\}$. As observed above, the function d_p is piecewise sinusoidal with $O(n)$ breakpoints, so its graph $\{(\theta, y) \mid y = d_p(\theta), -\pi/2 \leq \theta < \pi/2\}$ consists of $O(n)$ sinusoidal curves. Let Γ_p be the set of these sinusoidal curves, and $\Gamma := \bigcup_{p \in P} \Gamma_p$. We build the arrangement $\mathcal{A}(\Gamma)$ of these sinusoidal curves in Γ . Note that each vertex of $\mathcal{A}(\Gamma)$ corresponds either to a breakpoint of function d_p for some $p \in P$ or to an intersection point between a curve in Γ_p and another in $\Gamma_{p'}$ for some $p, p' \in P$ with $p \neq p'$.

► **Lemma 20.** *The arrangement $\mathcal{A}(\Gamma)$ of curves in Γ consists of $O(n^3)$ vertices, edges, and cells, and can be computed in $O(n^3)$ time.*

Now, we describe our decision algorithm. Let $w > 0$ be a given positive real number. First, we intersect the horizontal line $\ell: \{y = w\}$ with arrangement $\mathcal{A}(\Gamma)$.

► **Lemma 21.** *Any horizontal line crosses the edges of $\mathcal{A}(\Gamma)$ in $O(n^2)$ points, and all these intersection points can be specified in $O(n^2)$ time.*

Our algorithm continuously increases θ from $-\pi/2$ to $\pi/2$ and checks if there exists a parallelogram annulus of width w that encloses P such that one of the two double-strips defining it is θ -aligned.

Let $\{\theta_1, \dots, \theta_m\}$ be the set of θ -values of every intersection point between ℓ and the edges of $\mathcal{A}(\Gamma)$ such that $-\pi/2 \leq \theta_1 < \theta_2 < \dots < \theta_m < \pi/2$. Note that $m = O(n^2)$ by Lemma 21. For each $i = 1, \dots, m$, let $P_i \subseteq P$ be the set of points $p \in P$ such that $d_p(\theta_i) \leq w$, and let $Q_i := P \setminus P_i$. Let D_i be the θ_i -aligned P -constrained double-strip of width w . Then, we have $P_i \subset D_i$ while $Q_i \cap D_i = \emptyset$. Let D'_i be a P -constrained double-strip of minimum width that encloses Q_i . Recall that the width of D'_i is denoted by $w_{Q_i}^*$ in the previous section. If the width $w_{Q_i}^*$ of D'_i is at most w , then the parallelogram annulus defined by D_i and D'_i indeed encloses P and its width is w , so we conclude that $w \geq w^*$. Otherwise, if $w_{Q_i}^* > w$, then we proceed to next θ -value θ_{i+1} .

We perform this test for each $i = 1, \dots, m$ in an efficient way with the aid of our online decision algorithm for the constrained double-strip problem. Initially, for $i = 1$, we compute P_1 , Q_1 , and D_1 in $O(n)$ time. Also, we initialize the data structures \mathcal{T}_\emptyset and fixed value w , as described in Theorem 14 in $O(n \log n)$ time, and insert points in Q_1 to have \mathcal{T}_{Q_1} in $O(|Q_1|n) = O(n^2)$ time. We then know that $w_{Q_1}^* \geq w$ or not.

For each $i \geq 2$, there is a point $p \in P$ such that either $P_i = P_{i-1} \setminus \{p\}$ or $P_i = P_{i-1} \cup \{p\}$. Since $Q_i = P \setminus P_i$, we have that either $Q_i = Q_{i-1} \cup \{p\}$ or $Q_i = Q_{i-1} \setminus \{p\}$. This implies that we can answer whether $w \geq w_{Q_i}^*$ or not in $O(n)$ time for each $i \geq 2$ by maintaining P_i , Q_i , and \mathcal{T}_{Q_i} by Theorem 14.

Since $m = O(n^2)$ by Lemma 21, we can solve the decision problem in $O(n^3)$ time.

► **Lemma 22.** *Given $w > 0$, we can decide whether or not $w \geq w^*$ in $O(n^3)$ time.*

Finally, we describe our algorithm to compute the exact value of w^* . To do so, we collect a set W of candidate width values in which w^* is guaranteed to exist, and perform a binary search using our decision algorithm summarized in Lemma 22.

► **Lemma 23.** *The minimum possible width w^* over all parallelogram annuli that enclose P is equal to the y -coordinate of a vertex of $\mathcal{A}(\Gamma)$.*

We thus define W to be the set of y -coordinates of all vertices of $\mathcal{A}(\Gamma)$. Lemma 23 guarantees that $w^* \in W$. After sorting the values in W , we perform a binary search on W using the decision algorithm. Since $|W| = O(n^3)$ by Lemma 20 and the decision algorithm runs in $O(n^3)$ time by Lemma 22, we can find the exact value of w^* in $O(n^3 \log n)$ time. Therefore, we conclude the following theorem.

► **Theorem 24.** *Given a set P of n points, a minimum-width parallelogram annulus over all pairs of orientations that encloses P can be computed in $O(n^3 \log n)$ time.*

References

- 1 M. Abellanas, Ferran Hurtado, C. Icking, L. Ma, B. Palop, and P.A. Ramos. Best Fitting Rectangles. In *Proc. Euro. Workshop Comput. Geom. (EuroCG 2003)*, 2003.
- 2 P. Agarwal and M. Sharir. Planar geometric location problems. *Algorithmica*, 11:185–195, 1994.
- 3 P.K. Agarwal and M. Sharir. Efficient randomized algorithms for some geometric optimization problems. *Discrete Comput. Geom.*, 16:317–337, 1996.
- 4 P.K. Agarwal, M. Sharir, and S. Toledo. Applications of parametric searching in geometric optimization. *J. Algo.*, 17:292–318, 1994.
- 5 Sang Won Bae. Computing a Minimum-Width Square Annulus in Arbitrary Orientation. *Theoret. Comput. Sci.*, 718:2–13, 2018.
- 6 Sang Won Bae. On the Minimum-Area Rectangular and Square Annulus Problem. *CoRR*, 2019. [arXiv:1904.06832](https://arxiv.org/abs/1904.06832).

25:14 Double-Strip and Parallelogram Annulus

- 7 Mark de Berg, Mark van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 2nd edition, 2000.
- 8 Alex Glzman, Klara Kedem, and Gregory Shpitalnik. On some geometric selection and optimization problems via sorted matrices. *Comput. Geom.: Theory Appl.*, 11(1):17–28, 1998.
- 9 Olga N. Gluchshenko, Horst W. Hamacher, and Arie Tamir. An optimal $O(n \log n)$ algorithm for finding an enclosing planar rectilinear annulus of minimum width. *Operations Research Lett.*, 37(3):168–170, 2009.
- 10 J. Hershberger. Finding the upper envelope of n line segments in $O(n \log n)$ time. *Inform. Proc. Lett.*, 33:169–174, 1989.
- 11 Jerzy Jaromczyk and Mirosław Kowaluk. The Two-Line Center Problem from a Polar View: A New Algorithm and Data Structure. In *Proc. 4th Int. Workshop Algo. Data Struct. (WADS 1995)*, volume 955 of *Lecture Notes Comput. Sci.*, pages 13–25, 1995.
- 12 J. Mukherjee, P.R.S. Mahapatra, A. Karmakar, and S. Das. Minimum-width rectangular annulus. *Theoretical Comput. Sci.*, 508:74–80, 2013.
- 13 U. Roy and X. Zhang. Establishment of a pair of concentric circles with the minimum radial separation for assessing roundness error. *Computer-Aided Design*, 24(3):161–168, 1992.
- 14 G.T. Toussaint. Solving geometric problems with the rotating calipers. In *Proc. IEEE MELECON*, 1983.

Small Candidate Set for Translational Pattern Search

Ziyun Huang

Department of Computer Science and Software Engineering, Penn State Erie,
The Behrend College, Erie, PA, USA
zxh201@psu.edu

Qilong Feng

School of Computer Science and Engineering, Central South University, P.R. China
csufeng@mail.csu.edu.cn

Jianxin Wang

School of Computer Science and Engineering, Central South University, P.R. China
jxwang@csu.edu.cn

Jinhui Xu

Department of Computer Science and Engineering, State University of New York at Buffalo, USA
jinhui@buffalo.edu

Abstract

In this paper, we study the following pattern search problem: Given a pair of point sets A and B in fixed dimensional space \mathbb{R}^d , with $|B| = n, |A| = m$ and $n \geq m$, the pattern search problem is to find the translations \mathcal{T} 's of A such that each of the identified translations induces a matching between $\mathcal{T}(A)$ and a subset B' of B with cost no more than some given threshold, where the cost is defined as the minimum bipartite matching cost of $\mathcal{T}(A)$ and B' . We present a novel algorithm to produce a small set of candidate translations for the pattern search problem. For any $B' \subseteq B$ with $|B'| = |A|$, there exists at least one translation \mathcal{T} in the candidate set such that the minimum bipartite matching cost between $\mathcal{T}(A)$ and B' is no larger than $(1 + \epsilon)$ times the minimum bipartite matching cost between A and B' under any translation (i.e., the optimal translational matching cost). We also show that there exists an alternative solution to this problem, which constructs a candidate set of size $O(n \log^2 n)$ in $O(n \log^2 n)$ time with high probability of success. As a by-product of our construction, we obtain a weak ϵ -net for hypercube ranges, which significantly improves the construction time and the size of the candidate set. Our technique can be applied to a number of applications, including the translational pattern matching problem.

2012 ACM Subject Classification Theory of computation \rightarrow Pattern matching; Theory of computation

Keywords and phrases Bipartite matching, Alignment, Discretization, Approximate algorithm

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.26

Funding The research of the first and last authors was supported in part by NSF through grant CCF-1716400. The research of the last author was also supported by NSF through grant IIS-1910492. The research of the second and third authors was supported in part by NSFC through grants 61872450, 61828205, and 61672536.

1 Introduction

Pattern search/matching is an important problem in computer science and finds applications in many different domains such as computer vision, pattern recognition, robotics, autonomous driving, and surveillance. In this paper, we consider a special variant of the problem, where the objective is to find a small pattern (e.g., the image of a car) from a large environment (called background; e.g., the image of a road with traffic) which may contain multiple copies



© Ziyun Huang, Qilong Feng, Jianxin Wang, and Jinhui Xu;
licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 26; pp. 26:1–26:17

Leibniz International Proceedings in Informatics



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

of the pattern or its deformations. The problem is often encountered in our daily life. For example, most of the smart phones have the capability of identifying human faces (or other types of objects) in their camera softwares. The core problem for such a functionality is to efficiently find all appearances of a given object from the pictures. The pattern search problem may also appear in higher ($d > 2$) dimensional space. One of such examples is the pattern extraction problem arising in biological image analysis, where the objective is to identify the 3D spatial positioning patterns of chromosomes [9, 10, 22].

We approach this pattern search problem from a geometric perspective, using a formulation from [1] with some slight modifications. The pattern is represented by a point set A and the background by a point set B in \mathbb{R}^d space for some fixed d . The sizes of A and B are m and n , respectively, with $n \geq m$ (note that n could be significantly larger than m). If there is a translation \mathcal{T} which moves A to a new location $\mathcal{T}(A)$ such that the *difference* between $\mathcal{T}(A)$ and some $B' \subseteq B$ with $|B'| = |A|$ is minimized, we say that an instance B' of pattern A is discovered by \mathcal{T} , where the difference of two sets is measured by their bipartite matching cost (see Section Preliminaries).

The pattern search problem is a natural extension of the pattern matching problem, whose aim is to find a rigid transformation that minimizes the difference of two given sets A and B . Extensive research has been done for the pattern matching problem using different metrics as the measurement for the similarity/distance of two sets [13, 11, 14]. Commonly used metrics include Euclidean distance in 1-to-1 matching, Hausdorff distance in 1-to-many or many-to-1 matching, and Earth's Mover Distance (EMD) in many-to-many matching. An early result on this problem is the paper [18] which provides an $\tilde{O}(mn^2)$ -time solution to the matching problem under translation and Hausdorff distance in \mathbb{R}^2 . A more recent result is the one in [11] which approximates (with ratio $(1 + \epsilon)$) the pattern matching problem under rigid transformations and EMD metric in \mathbb{R}^d . The running time of their algorithm is $\tilde{O}((mn)^{2d})$, which is near the lower bound (i.e., $\Omega(mn^{\Omega(d)})$ [5]) of the problem.

For the pattern search problem under translations, there is a number of results [21, 4] that are closely related to the work in this paper. Most of them use the concept of *partial-matching Voronoi diagram*.

For two given point sets A and B , their partial-matching Voronoi diagram (PMVD) is a partition of the translation space into regions so that each of them consists of translations \mathcal{T} sharing the same locally optimal bipartite matching between $\mathcal{T}(A)$ and a subset of B . The PMVD uses the sum of squared distances as the measurement for the matching cost. Clearly, such a Voronoi diagram is capable of solving the pattern search/matching problem, as only one translation from every cell needs to be determined for finding the optimal translational alignment of A and B . The best known upper bound on the size of PMVD is $O(m!m^d n^{2d})$ [17]. Ben-Avraham et al. [4] constructed a partial-matching Voronoi diagram in \mathbb{R}^2 of complexity $O(n^2 m^{3.5} \log^m m)$, and found *locally* min-cost translations in $O(m^6 n^3 \log n)$ time.

In this paper, we develop a novel method for finding a small set \mathbb{T} of candidate translations so that for any instance $B' \subseteq B$ of A , there is at least one translation \mathcal{T} in the candidate set that matches $\mathcal{T}(A)$ and B' approximately. A subset B' of B is called an instance of A if $|B'| = |A|$. Note that B' can be any subset of B as long as it has the same size as A and may have a large difference with A ; this is somewhat different from the normal meaning of instance. We say that a translation \mathcal{T} *discovers* an instance B' if it minimizes the bipartite matching cost of $\mathcal{T}(A)$ and B' . For any $\epsilon > 0$, a translation \mathcal{T} $(1 + \epsilon)$ -approximately discovers B' , if the bipartite matching cost between $\mathcal{T}(A)$ and B' is no more than $(1 + \epsilon)$ times the minimum difference between B' and A under any translation. Clearly, with such a candidate set \mathbb{T} , we are able to find all instances B' which are similar to A , where the level of similarity

is controlled by some threshold on the difference of B' and the translations of A . Note that if a value of the threshold is given in advance, it is possible to further reduce the size of the candidate set by removing (during the execution of our algorithm) those translations which induce higher matching cost (see the remark in Section 6 for more details). Also, if B has some exact (or congruent) instances of A , \mathbb{T} will contain all translations inducing zero-difference matchings of A .

The problem of finding the translations that match pattern A to all its exact instances could be quite challenging, as suggested by the exponential size of PMVD in [21, 4]. However, we are able to show that if approximation and implicit representation are allowed, the problem can be solved much more efficiently through identifying a small candidate set of translations with a (surprisingly) near linear size.

Particularly, we show that it is possible to build a candidate set \mathbb{T} with size $O_{d,\epsilon}(n \log n)$ for A and B in $O(mn \log mn)$ deterministic time. This bound is asymptotically near optimal, since it is easy to construct an example that needs $O(n)$ different translations to yield all perfect matches of a pattern (such an example will be given later). A trade-off between the running time and the size of \mathbb{T} can also be made, which provides a probabilistic algorithm to build a \mathbb{T} with a slightly larger size (i.e., $O(n \log^2 n)$) but a better time complexity (i.e., $O(n \log^2 n)$) which is independent of m . Our construction is based on a weak ϵ -net technique and a space discretization technique from [7]. Our approach shows a non-obvious connection between weak ϵ -net and the pattern search problem. A fast algorithm is also provided to build a small-size ϵ -net for ranges of axis-aligned hypercubes.

In some sense, candidate set \mathbb{T} can be viewed as an implicit and approximate representation of the exponential-size PMVD. Thus, it has the potential to be used in various applications of PMVD, such as moving object tracking and autonomous driving. Note that in such applications, all translations in \mathbb{T} (rather than those inducing better matchings) are needed.

2 Preliminaries

In this paper, we do not distinguish a point and its corresponding vector in \mathbb{R}^d , i.e., a point is equivalent to a vector that points from the origin to it. In this way, a point in \mathbb{R}^d naturally defines a translation (along the corresponding vector) in \mathbb{R}^d . Following the basic vector arithmetics, operators $+$ and $-$ can be applied to points in \mathbb{R}^d .

Given two point sets A_1 and A_2 of the same size, their bipartite matching is represented by a bijective mapping $\phi : A_1 \rightarrow A_2$. That is, each point $a \in A_1$ is matched to a distinct point $\phi(a) \in A_2$ and the cost C_{ϕ, A_1, A_2} of the matching is defined as $C_{\phi, A_1, A_2} = \sum_{a \in A_1} \|a - \phi(a)\|$. The difference of A_1 and A_2 , denoted by $\Delta(A_1, A_2)$, is then $\Delta(A_1, A_2) = \min_{\phi} C_{\phi, A_1, A_2}$.

Let $A, B \subset \mathbb{R}^d$ be the two point sets in the pattern search problem, with $|B| = n, |A| = m$ and $n \geq m$. We label points in A and B , respectively, as $A = \{a_1, a_2, \dots, a_m\}$ and $B = \{b_1, b_2, \dots, b_n\}$. The **reference set** P of A and B is defined as follows.

► **Definition 1.** Let $p_{i,j} = b_j - a_i$ for any $a_i \in A$ and $b_j \in B$. The reference set P of A and B is the multi-set that contains all $p_{i,j}$ for $i = 1, 2, \dots, m$ and $j = 1, 2, \dots, n$.

We use an injective mapping $\phi : \{1, 2, \dots, m\} \rightarrow \{1, 2, \dots, n\}$ to represent a perfect matching for A (under a certain translation) and a subset of B . $\phi(i) = j$ means that a_i is matched to b_j . The matching cost C_{ϕ} for a matching ϕ is defined as $\sum_{i=1}^m \|a_i - b_{\phi(i)}\|$.

From the definition of P , it is clear that for any $a_i \in A$, $b_j \in B$ and translation $\mathcal{T} \in \mathbb{R}^d$, $\|\mathcal{T}(a_i) - b_j\| = \|\mathcal{T} - p_{i,j}\|$. The matching cost $C_{\phi}(\mathcal{T})$ between $\mathcal{T}(A)$ and B for matching ϕ is then $\sum_{i=1}^m \|\mathcal{T} - p_{i,\phi(i)}\|$. In other words, $C_{\phi}(\mathcal{T})$ is the sum of distances from \mathcal{T} to m points in P . For convenience, we let $P(\phi) = \{p_{1,\phi(1)}, p_{2,\phi(2)}, \dots, p_{m,\phi(m)}\}$. Thus $C_{\phi}(\mathcal{T}) = \sum_{p \in P(\phi)} \|\mathcal{T} - p\|$.

In the rest of the paper, we study all possible matchings between A and B based on the relationship of $\mathcal{T} \in \mathbb{R}^d$ and P . This means that our algorithms work in the translational space of P , instead of the original space of A and B .

For any pair of point sets X and Y with the same cardinality, we use $\Delta(X, Y)$ to denote the minimum bipartite matching cost of X and Y . A set $B' \subseteq B$ is called an instance of A if $|B'| = |A|$. For any instance B' , let \mathcal{T} be the translation that minimizes $\Delta(\mathcal{T}(A), B')$. Then, we say that \mathcal{T} *discovers* B' , or B' is discoverable at \mathcal{T} . If there is another translation \mathcal{T}' satisfying the following inequality, $\Delta(\mathcal{T}'(A), B') \leq (1 + \epsilon)\Delta(\mathcal{T}(A), B')$, then, we say that B' is $(1 + \epsilon)$ -approximately discoverable at \mathcal{T}' .

3 Main Results

The main results of this paper are the following theorems which show that for any given pair of point sets A and B , and any constant $\epsilon > 0$, it is possible to efficiently construct a small-size candidate set \mathbb{T} of translations in \mathbb{R}^d such that any instance $B' \subseteq B$ of A is approximately discoverable at some $\mathcal{T} \in \mathbb{T}$. In other words, \mathbb{T} is a small-size candidate set of translations to find all instances of A approximately.

► **Theorem 2.** *For any pair of point sets A and B in fixed dimensional space \mathbb{R}^d with size m and n ($n \geq m$), respectively, and any small constant $0 < \epsilon < 1$, it is possible to construct, deterministically, a candidate set $\mathbb{T} \subset \mathbb{R}^d$ of size $O(n \log n)$ in $O(mn \log mn)$ time such that for any given instance $B' \subseteq B$ of A , there exists a translation $\mathcal{T} \in \mathbb{T}$ that $(1 + \epsilon)$ -approximately discovers B' .*

► **Theorem 3.** *For any pair of point sets A and B in fixed dimensional space \mathbb{R}^d with size m and n ($n \geq m$), respectively, and any small constant $0 < \epsilon < 1$, it is possible to construct a candidate set $\mathbb{T} \subset \mathbb{R}^d$ of size $O(n \log^2 n)$ in $O(n \log^2 n)$ time, with success probability at least $1 - 1/n$. For any given instance $B' \subseteq B$ of A , there exists a translation $\mathcal{T} \in \mathbb{T}$ that $(1 + \epsilon)$ -approximately discovers B' .*

With the above theorems, we immediately have the following corollary as their application to the classical pattern matching problem. (See the appendix for the proof.)

► **Corollary 4.** *It is possible to generate a set of $O(n \log n)$ candidate translations in \mathbb{R}^d in $O(mn \log mn)$ time such that one of them induces a $(1 + \epsilon)$ -approximation for the optimal translational matching between A and B .*

Another interesting conclusion from the above discussion is that if there exists an instance $B' \subseteq B$ which is identical to A under translation \mathcal{T} (i.e., the bipartite matching cost between B' and $\mathcal{T}(A)$ is 0), then $\mathcal{T} \in \mathbb{T}$.

The above theorems and corollary suggest an efficient way to identify a small number of translations that enable us to obtain approximate solutions to the translational pattern matching problem. The size of the candidate set is near optimal. This can be easily seen from the following simple example in 1-D: Consider $A = \{1, 2\}$ and $B = \{1, 2, \dots, n\}$; then all the n translations that align 1 in A to any of the $n - 1$ points $\{1, 2, \dots, n - 1\}$ in B are optimal translations.

After obtaining all the candidate translations, the approximate optimal matching can then be computed by solving a min-cost partial matching problem for fixed point sets $\mathcal{T}(A)$ and B for every candidate \mathcal{T} .

3.1 Overview of Techniques

Our main idea for constructing a small candidate set is via space discretization. We are able to prove a locality property for the pattern search problem: if the distance between two translations \mathcal{T}_1 and \mathcal{T}_2 (viewed as points in \mathbb{R}^d) is close compared to their closest distance to any point in the reference set P , then for any instance $B' \subseteq B$, $\Delta(\mathcal{T}_1(A), B')$ and $\Delta(\mathcal{T}_2(A), B')$ are also close. This suggests that we can decompose \mathbb{R}^d into “small” regions, so that the every region has a small diameter, comparing to its distance to P . For any $B' \subseteq B$, let \mathcal{T}_O be the translation that discovers B' , and \mathcal{T}_O lies in some “small” region C . Then, any $\mathcal{T}' \in C$ approximately discovers B' . This means that, if we choose one arbitrary point from each “small” region to form the candidate set \mathbb{T} , it is guaranteed that any instance B' will be approximately discoverable by some translation in \mathbb{T} . The details will be shown in Section 4.

However, making all regions of the entire space “small” seems to be challenging, if not impossible. Existing space discretization techniques, such as [7, 3, 15], are only able to ensure that regions distant from the points in P are “small”. For regions close to points in P (called “close” regions), new techniques are needed to select their candidate translations. In Section 5, we discuss how to choose translations from “close” regions so that every instance B' whose corresponding optimal translation falls in some “close” region can also be approximately discoverable.

In Section 4.4, we also describe how to use weak ϵ -net to “sketch” P (with size mn) using a much smaller set Q of size $O(n)$. This will allow us to significantly reduce the size of discretization (and thus the size of the candidate set) from $\tilde{O}(mn)$ to $\tilde{O}(n)$, which is near optimal. We are able to show that such a sketching still preserves the locality property of the pattern search problem.

4 Locality Based Discretization for Pattern Search

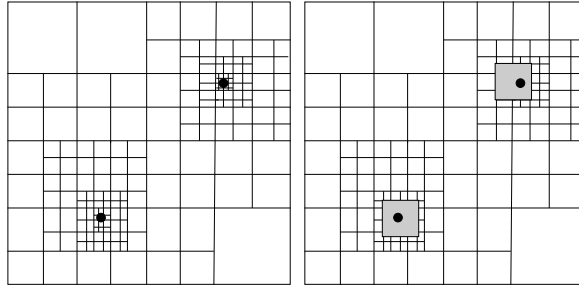
In this section, we present a discretization approach for the pattern search problem, based on the locality property of the problem.

4.1 Locality of Pattern Search

In the context of pattern search, locality refers to the following observation: for any instance $B' \subseteq B$ and two translations \mathcal{T}_1 and \mathcal{T}_2 , if \mathcal{T}_1 and \mathcal{T}_2 are close to each other, their induced minimum bipartite matching costs between the translations of A and B' are also close. Let $\Delta(X, Y)$ denote the minimum bipartite matching cost between two point sets X and Y . The following lemma shows the locality property with respect to the distance between translations and a point in the reference set P . (See Appendix for the proof.)

► **Lemma 5.** *Let \mathcal{T}_1 and \mathcal{T}_2 be two translations in \mathbb{R}^d , and $p \in P$ be the nearest neighbor of \mathcal{T}_1 in the reference set P . If $\|\mathcal{T}_1 - \mathcal{T}_2\| \leq \epsilon \|p - \mathcal{T}_1\|$ for some constant $0 < \epsilon < 1$, then $|\Delta(\mathcal{T}_1(A), B') - \Delta(\mathcal{T}_2(A), B')| \leq \epsilon \Delta(\mathcal{T}_1(A), B')$ for any instance $B' \subseteq B$.*

The above locality property suggests the following discretization approach to find a candidate set for the pattern search problem. The idea is to decompose \mathbb{R}^d into a number of “small” regions. Each region C is “small” enough in the sense that the minimum distance r between C and the points in P is large, comparing to the diameter $D(C)$ of C , i.e., $D(C) \leq \epsilon r$ for some constant $0 < \epsilon < 1$. With such a discretization, we may then simply choose one arbitrary translation from each region to form the candidate set \mathbb{T} . To see that this is indeed the desired candidate set, consider any instance $B' \subseteq B$. Let \mathcal{T} be the translation that discovers B' , C' be the region containing \mathcal{T} , and $\mathcal{T}_C \in \mathbb{T}$ be the translation chosen from C' . Then, by Lemma 5, we know that B' is $(1 + \epsilon)$ -approximately discoverable at \mathcal{T}_C .



■ **Figure 1** Illustrative figures for “small” and “close” regions. Left Figure: If every region needs to be “small”, then an infinite number of regions will be generated around each point in P . Right Figure: Possible “close” regions to prevent from yielding an infinite number of “small” regions.

4.2 Space Discretization and Close Regions

Unfortunately, an exact implementation of the above space discretization is not possible. This is because the size of some regions can be infinitely small if the distance of the region to a point in P is small enough. This means that an infinite number of regions can be generated around every point in P (see Figure 1). To overcome this difficulty, a possible way is to utilize some of the known space discretization techniques, such as [7, 3, 15]. However, a common issue of such techniques is that only part of the resulting regions can be viewed as “small” (i.e., $D \leq \epsilon r$, where D is the diameter of the region and r is the minimum distance between the a point in P and the region). Such regions are distant from points in P . All other regions are in close proximity to points in P , and cannot be viewed as “small”, even though their diameters might be small. We call such regions as “close” regions.

Clearly, for “small” regions, it will be sufficient (by Lemma 5) to choose one point arbitrarily from each of them and include it into the candidate set \mathbb{T} . The main issue is, thus, how to select candidate translations from the “close” regions. We will discuss our ideas on close regions in next section.

In this paper, we use the technique in [7] for space discretization. The main reason is that using this technique, we have some good geometric properties on close regions. This will help us ensure the correctness of our proposed approach and simplify the analysis.

For self completeness, below we summarize the space discretization technique in [7]. The main technique in [7] is an algorithm $\text{AIDecomposition}(P, \beta, \gamma)$, where P is a point set in \mathbb{R}^d and $0 < \beta, \gamma < 1$ are two small constants (to be determined in later analysis). The following lemma is the main result of the algorithm.

► **Lemma 6.** [7] $\text{AIDecomposition}(P, \beta, \gamma)$ generates a partition of \mathbb{R}^d in $O_{d,\beta,\gamma}(|P| \log|P|)$ time, where each region C of the partition satisfies one of the following conditions.

1. C is associated with a subset V of P and a point $v \in V$, such that
 - a. The diameter $D(V)$ of V is no larger than βr , where r is the closest distance between a point in C and a point in V .
 - b. $\|v - u\| \leq \gamma \|v' - u\|$ and $D(C) \leq \beta \|v' - u\|$, for any point $u \in C$ and any point $v' \in P \setminus V$
2. $D(C) \leq \beta r$, where r is the closest distance between a point in C and a point in P .

The regions that satisfy condition 1 are the close regions in our previous discussion, and the regions that satisfy condition 2 correspond to the small regions. Note that for a close region C generated by AIDecomposition , it is close to the associated point set $V \subset P$,

comparing to points in $P \setminus V$, where the closeness is controlled by the parameter γ . There are also some other interesting properties of close regions generated by AIDecomposition shown in Lemma 6. These properties will prove to be useful in later analysis.

4.3 Reducing the Number of Regions

If we directly apply the above AIDecomposition technique or any other space discretization technique (such as [3, 15]) to the reference set P , at least $\Omega(mn)$ regions will be generated, since $|P| = mn$. This results in a candidate set of size $\Omega(mn)$, which could be significantly larger than $O(n)$, i.e., the maximum number of possible translations that could yield a perfect matching for $\mathcal{T}(A)$ and B . Thus, it is tempting to ask whether it is possible to construct a discretization with size only near $O(n)$.

A natural approach for size reduction is to use a smaller set Q to “sketch” P , and build a discretization for Q , instead of P . Let $\xi > 1$ and $\mu > 0$ be some given constants. We require that Q be (ξ, μ) -dense for P , defined as follows.

► **Definition 7.** A point set $Q \subset \mathbb{R}^d$ is called (ξ, μ) -dense for P , if for any point $\mathcal{T} \in \mathbb{R}^d$, it satisfies $\mu \|p_\xi - \mathcal{T}\| \geq \|q - \mathcal{T}\|$, where p_ξ is the m/ξ -th closest point in P to \mathcal{T} , and $q \in Q$ is the nearest neighbor of \mathcal{T} in Q .

In other words, points in Q are “dense” enough, so that for any $\mathcal{T} \in \mathbb{R}^d$, it is possible to find a point $q \in Q$ that is closer to, or not much farther away from \mathcal{T} than p_ξ . Using such a formulation for “dense” allows us to use $\|q - \mathcal{T}\|$ to effectively lower bound $\Delta(\mathcal{T}(A), B')$ for any B' . Let ϕ be the matching realizing the minimum cost bipartite matching between $\mathcal{T}(A)$ and B' . Then, we have

$$\|q - \mathcal{T}\| \leq \mu \|p_\xi - \mathcal{T}\| \leq \sum_{p \in P(\phi)} \mu \|p - \mathcal{T}\| / ((1 - 1/\xi)m) = \mu \Delta(\mathcal{T}(A), B') / ((1 - 1/\xi)m). \quad (1)$$

Using an argument similar to the one in Lemma 5, we have the following improved version of locality property.

► **Lemma 8.** Let \mathcal{T}_1 and \mathcal{T}_2 be two translations in \mathbb{R}^d , and $p \in Q$ be the nearest neighbor of \mathcal{T}_1 in Q . If $\|\mathcal{T}_1 - \mathcal{T}_2\| \leq \beta \|p - \mathcal{T}_1\|$ for constant $0 < \beta < 1$, then $|\Delta(\mathcal{T}_1(A), B') - \Delta(\mathcal{T}_2(A), B')| \leq \beta \mu (1 - 1/\xi)^{-1} \Delta(\mathcal{T}_1(A), B')$ for any instance $B' \subseteq B$.

The above lemma enables us to use Q for the space discretization. More specifically, we run AIDecomposition(Q, β, γ) on an (ξ, μ) -dense Q (with β, ξ, μ and γ to be determined later). This yields a near linear size (in terms of the size of Q) discretization. From previous discussion, we know that for any instance $B' \subseteq B$, if the translation that discovers B' lies in a small region (note that since the discretization is based on Q instead of P , “small region” now means that their diameters are small compared to the distances to their nearest neighbors in Q), then any translation in the small region will $(1 + \epsilon)$ -approximately discover B' , if parameters β, ξ, μ and γ are properly chosen according to the desired approximate ratio ϵ . A formal argument will be provided later when analyzing the correctness of our algorithm. The remaining main challenge is then to deal with the case that the translation discovering B' lies in a close region. This will be covered in the next section.

4.4 Finding (ξ, μ) -dense Q for P

To conclude this section, we briefly describe how to find a (ξ, μ) -dense set Q for the discretization.

A set Q that is (ξ, μ) -dense can actually be found by constructing a weak ϵ -net of P [16]. ϵ -net is an important concept in combinatorial and computational geometry, and has been extensively studied in the past. For our problem, we use weak ϵ -net for axis-aligned hypercubes (i.e., hyper-boxes with equal edge length in every direction). Below is the definition.

► **Definition 9.** Let P be any point set in \mathbb{R}^d , and ϵ be any small constant between 0 and 1. A point set $Q \subset \mathbb{R}^d$ is called a weak ϵ -net of P for axis-aligned hypercubes, if any axis-aligned hypercube G containing $\epsilon|P|$ or more points in P also contains at least one point in Q .

Note that ϵ -net is a much more general concept than the above definition. In this paper, we only consider weak ϵ -net for axis-aligned hypercubes. For convenience, we will use thereafter the term “ ϵ -net” without specifying “weak” and “axis-aligned hypercubes”. For any constant $c > 1$, it is easy to see that a $1/cn$ -net Q of P is (c, \sqrt{d}) -dense.

A small-size ϵ -net for the reference set P can be built efficiently. (We leave the proofs of the following lemmas to the end of the paper.)

► **Lemma 10.** For any point multi-set $P \subset \mathbb{R}^d$ with size mn and any constant $c > 0$, a $1/cn$ -net of P with size $O_d(n)$ can be generated in $O_d(nm \log nm)$ time.

► **Lemma 11.** A $1/cn$ -net of the reference set P with size $O_d(n \log n)$ can be generated in $O_d(n \log n)$ time for any constant $c > 0$ with probability at least $1 - 1/n$.

5 Handling Close Regions

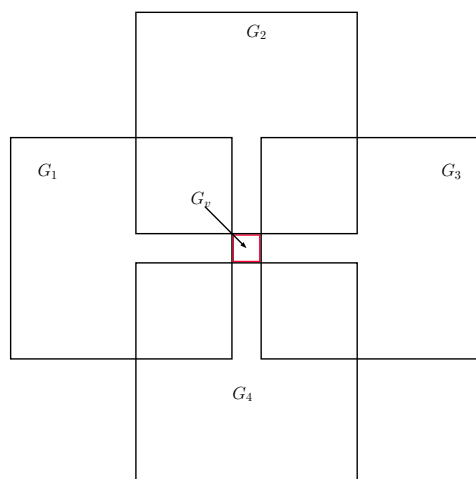
In last section, we discuss how to use (ξ, μ) -dense Q to obtain a space discretization and how to select translations from small regions. In this section, we show how to pick translations from close regions (i.e. regions that satisfy condition 1 in Lemma 6) so that they will be good approximations for all those translations that fall in close regions and discover some instances of B .

Below, we assume that γ, β, μ and ξ are chosen such that $\gamma < (16\sqrt{d} + 1)^{-1}$, $4\sqrt{d}\beta < 1$ and $\xi = 12d$. Let $B' \subseteq B$ be an instance of A and \mathcal{T}_O be the translation that discovers B' and lies in some close region C . Denote by $V \subset Q$ and $v \in V$, respectively, the subset of points in Q and its representative associated with C , as indicated in condition 1 of Lemma 6. Let ϕ_O be the matching between $\mathcal{T}_O(A)$ and B' that realizes the minimum bipartite matching cost. Let G_v be the smallest axis-aligned box containing V . We consider 2 cases: $|G_v \cap P(\phi_O)| \geq 2m/3$ or $|G_v \cap P(\phi_O)| < 2m/3$.

► **Lemma 12.** If $|G_v \cap P(\phi_O)| \geq 2m/3$, $\mathcal{T}_O = v$.

Proof. Assume by contradiction that $\mathcal{T}_O \neq v$. For any matching ϕ and any translation \mathcal{T} , we use notation $C_\phi(\mathcal{T})$ to denote the matching cost between $\mathcal{T}(A)$ and B' under ϕ . In the following, we analyze how the value of $C_{\phi_O}(\mathcal{T})$ changes, where variable \mathcal{T} is initially \mathcal{T}_O , and then changed to v . Note that $C_{\phi_O}(\mathcal{T}) = \sum_{p \in P(\phi_O)} \|p - \mathcal{T}\| = \sum_{p \in G_v \cap P(\phi_O)} \|p - \mathcal{T}\| + \sum_{p \in P(\phi_O) \setminus G_v} \|p - \mathcal{T}\|$.

To estimate the change of $\sum_{p \in G_v \cap P(\phi_O)} \|p - \mathcal{T}\|$, we note that for any $p \in G_v \cap P(\phi_O)$, $\|p - v\| \leq \sqrt{d}D(V) \leq \sqrt{d}\beta\|\mathcal{T}_O - v\|$, where the last inequality is from Condition 1 of Lemma 6 and the fact that \mathcal{T}_O is in C . From triangle inequality, we have $\|p - \mathcal{T}_O\| \geq \|v - \mathcal{T}_O\| - \|v - p\| \geq (1 - \sqrt{d}\beta)\|v - \mathcal{T}_O\|$. Thus, we get $\|p - \mathcal{T}_O\| - \|p - v\| \geq (1 - 2\sqrt{d}\beta)\|v - \mathcal{T}_O\|$. This means that moving \mathcal{T} from \mathcal{T}_O to v reduces the value of $\sum_{p \in G_v \cap P(\phi_O)} \|p - \mathcal{T}\|$ by at least $(2m/3)(1 - 2\sqrt{d}\beta)\|v - \mathcal{T}_O\|$.



■ **Figure 2** Illustration of the arrangement of $\{G_1, G_2, \dots, G_{2d}\}$ and G_v .

For the term of $\sum_{p \in P(\phi_O) \setminus G_v} \|p - \mathcal{T}\|$, we know (from triangle inequality and the assumption that $|P(\phi_O) \setminus G_v| \leq m/3$) that its change is smaller than $(m/3)\|v - \mathcal{T}_O\|$. Since $1 - 2\sqrt{d}\beta > 1/2$ (by the assumption that $4\sqrt{d}\beta < 1$), we get $(2m/3)(1 - 2\sqrt{d}\beta)\|v - \mathcal{T}_O\| > (m/3)\|v - \mathcal{T}_O\|$ when $\mathcal{T}_O \neq v$. Therefore, we have $C_{\phi_O}(v) < C_{\phi_O}(\mathcal{T}_O)$ (from previous discussion). However, this results in a contradiction, since from definition, \mathcal{T}_O discovers B' and thus should have the minimum matching cost between A and B' under any translation.

Thus, the lemma follows. ◀

For the case $|G_v \cap P(\phi_O)| < 2m/3$, we have the following lemma.

► **Lemma 13.** *If $|G_v \cap P(\phi_O)| < 2m/3$, then for any $\mathcal{T}' \in C$, $\Delta(\mathcal{T}'(A), B') \leq (1 + 48\beta\sqrt{d})\Delta(\mathcal{T}_O(A), B')$.*

Proof. For any matching ϕ and any translation \mathcal{T} , we use notation $C_{\phi}(\mathcal{T})$ to denote the matching cost between $\mathcal{T}(A)$ and B' under ϕ . We prove this lemma by showing that $C_{\phi_O}(\mathcal{T}') \leq (1 + 48\beta\sqrt{d})C_{\phi_O}(\mathcal{T}_O)$; the lemma then follows, since $C_{\phi_O}(\mathcal{T}') \geq \Delta(\mathcal{T}'(A), B')$.

Let P' be the closest $5m/6$ points to \mathcal{T}_O in $P(\phi_O)$. Since $|G_v \cap P(\phi_O)| < 2m/3$, we have $|P' \setminus G_v| \geq m/6$.

For analysis purpose, imagine that we “attach” $2d$ axis-aligned boxes $\{G_1, G_2, \dots, G_{2d}\}$ to each face of G_v , with centers aligned in G_v (see Figure 2 for an example in 2D), and each box has equal edge length r , where r is the smallest positive number such that P' is contained in the union of $\{G_1, G_2, \dots, G_{2d}\}$ and G_v . Let $F = G_v \cup G_1 \cup G_2 \dots \cup G_{2d}$.

By the fact that $|P' \setminus G_v| \geq m/6$, we know that one box of $\{G_1, G_2, \dots, G_{2d}\}$ contains more than $m/12d$ points in P . Since $\xi = 12d$ and Q is a $1/n\xi$ -net of P , we also know that the box contains a point q_t from Q . Thus, F contains a point q_t in $Q \setminus V$.

From Lemma 6, we have $\|\mathcal{T}_O - v\| \leq \gamma\|\mathcal{T}_O - q_t\|$. Thus,

$$\|v - q_t\| \geq (1/\gamma - 1)\|\mathcal{T}_O - v\|. \tag{2}$$

Let L_v denote the edge length of G_v . Then, we have $L_v \leq D(V) \leq \beta\|\mathcal{T}_O - v\|$. Since $\beta < 1/4\sqrt{d} < (1/4\sqrt{d})(1/\gamma - 1)$, from (2) we get

$$L_v \leq \|v - q_t\|/4\sqrt{d}. \tag{3}$$

26:10 Small Candidate Set for Translational Pattern Search

Let L denote the length of F : $L = 2r + L_v$. L is clearly no smaller than $\|v - q_t\|/\sqrt{d}$ in order to contain both v and q_t . From (3), we have $L_v \leq r$ and $r \geq 3\|v - q_t\|/8\sqrt{d}$. By (2) and the assumption that $1/\gamma - 1 \geq 16\sqrt{d}$, we get

$$r \geq 3(1/\gamma - 1)\|\mathcal{T}_O - v\|/8\sqrt{d} \geq 6\|\mathcal{T}_O - v\|. \quad (4)$$

Let O_v denote the center of G_v . Then, $\|O_v - v\| \leq \sqrt{d}L_v \leq \sqrt{d}D(V) \leq \sqrt{d}\beta\|\mathcal{T}_O - v\| \leq \|\mathcal{T}_O - v\|/4$. Thus, from triangle inequality we have

$$\|O_v - \mathcal{T}_O\| \leq 5\|\mathcal{T}_O - v\|/4. \quad (5)$$

Combining (4) and (5) gives us $\|O_v - \mathcal{T}_O\| \leq 5r/24$.

From the definition of r , we know that there must exist a point $p' \in P'$ on the boundary of F . From the arrangement of $\{G_1, G_2, \dots, G_{2d}\}$ and G_v , we have $\|O_v - p'\| \geq r/2$. Thus, $\|\mathcal{T}_O - p'\| \geq r/2 - 5r/24 > r/4$ (by triangle inequality).

Also, from the fact that $L_v \leq r$, we know that F can be covered by a box centered at O_v and with edge length $3r$. Since $q_t \in F$, we have $\|O_v - q_t\| \leq 3\sqrt{d}r/2$. Combining this with the fact that $\|O_v - \mathcal{T}_O\| \leq 5r/24$, we get $\|\mathcal{T}_O - q_t\| \leq 3\sqrt{d}r/2 + 5r/24 \leq 2\sqrt{d}r$. Thus, we have $\|\mathcal{T}_O - q_t\|/\|\mathcal{T}_O - p'\| \leq 8\sqrt{d}$.

In summary, from the above discussion and Lemma 6, we have the following.

1. There exists $q_t \in Q$ such that the following inequality holds $\|\mathcal{T}_O - q_t\|/\|\mathcal{T}_O - p'\| \leq 8\sqrt{d}$, where $p' \in P(\phi_O)$ satisfies the condition that less than $5m/6$ points in $P(\phi_O)$ are closer to \mathcal{T}_O than it.
2. Inequality $\|\mathcal{T}_O - \mathcal{T}'\| \leq \beta\|\mathcal{T}_O - q_t\|$ holds for any $\mathcal{T}' \in C$.

Following a similar argument given in Lemma 5, we can show that $|C_{\phi_O}(\mathcal{T}_O) - C_{\phi_O}(\mathcal{T}')| \leq 48\beta\sqrt{d}C_{\phi_O}(\mathcal{T}_O)$. This concludes the proof. \blacktriangleleft

Now for any $\epsilon > 0$, if we set $48\beta\sqrt{d} \leq \epsilon$, then by Lemmas 12 and 13, we know that if \mathcal{T}_O lies in a close region C generated by AIDecomposition, either v discovers B' (if $|G_v \cap P(\phi_O)| \geq 2m/3$), or an arbitrary $\mathcal{T}' \in C$ $(1 + \epsilon)$ -approximately discovers B' (if $|G_v \cap P(\phi_O)| < m/3$). Therefore, we may put v and an arbitrary point in C into the candidate set. This ensures that B' is $(1 + \epsilon)$ -approximately discoverable by at least one of these two points when \mathcal{T}_O lies in C .

6 The Algorithm and Analysis

In this section, we summarize the discussion so far and provide the algorithm to generate the candidate set of translations. The following Algorithm 1 shows in details how to generate the candidate set \mathbb{T} .

Depending on the method chosen to construct the ϵ -net in Step 3 (Lemma 10 or Lemma 11), the size of Q is $O(n)$ or $O(n \log n)$, the size of the discretization (in terms of number of regions) generated in step 4 is $O(n \log n)$ or $O(n \log^2 n)$, and the total running time of the algorithm is $O(mn \log mn)$ or $O(n \log^2 n)$. No matter which algorithm is chosen to construct \mathbb{T} , we have the following lemma.

► Lemma 14. *For any instance $B' \in B$ of A , there exists at least one translation $\mathcal{T} \in \mathbb{T}$, such that \mathcal{T} $(1 + \epsilon)$ -approximately discovers B' .*

Algorithm 1 Generate-Candidate-Set.

Input: Point sets A and B of \mathbb{R}^d with $|A| \leq |B|$. Approximate factor $0 < \epsilon < 1$.

Output: A set \mathbb{T} of translations in \mathbb{R}^d , such that each instance B' of A is $(1+\epsilon)$ -approximately discoverable.

- 1: Initialize \mathbb{T} to be \emptyset .
 - 2: Initialize constants β, γ, ξ , such that: $\xi = 12d$, $48\beta\sqrt{d} \leq \epsilon$, $\gamma < (16\sqrt{d} + 1)^{-1}$, $4\sqrt{d}\beta < 1$ and $\sqrt{d}\beta(1 - 1/\xi)^{-1} \leq \epsilon$.
 - 3: Build a $1/\xi n$ -net Q for P , where P is the reference set.
 - 4: Run $\text{AIDecomposition}(Q, \beta, \gamma)$ to generate a discretization which decomposes \mathbb{R}^d into close regions and small regions.
 - 5: For each small region C , pick an arbitrary point from C and put it into \mathbb{T} .
 - 6: For each close region C , suppose it is associated with point set V and $v \in V$. Pick an arbitrary point p in C . Put both v and p into \mathbb{T} .
 - 7: Output \mathbb{T} as the result.
-

Proof. Let \mathcal{T}_O be the translation that discovers B' .

If \mathcal{T}_O lies in a small region C , let $\mathcal{T}_C \in \mathbb{T}$ be the point chosen from C in step 5 of Algorithm 1. Let q be the nearest neighbor of \mathcal{T}_O in Q . By Lemma 6, we know that $\|\mathcal{T}_C - \mathcal{T}_O\| \leq \beta\|q - \mathcal{T}_O\|$. By Lemma 8 and the fact that Q is (ξ, \sqrt{d}) -dense, we have $\Delta(\mathcal{T}_C(A), B') \leq (1 + \sqrt{d}\beta(1 - \xi)^{-1})\Delta(\mathcal{T}_O(A), B') \leq (1 + \epsilon)\Delta(\mathcal{T}_O(A), B')$ (The last inequality comes from choice of parameters in Algorithm 1). Thus, we know that B' is $(1 + \epsilon)$ -approximately discoverable at $\mathcal{T}_C \in \mathbb{T}$.

If \mathcal{T}_O lies in a close region C , let $v \in V \subset Q$ be the representative point associated with C as stated in Lemma 6. Let $\mathcal{T}_C \in \mathbb{T}$ be the point chosen from C in step 6 of Algorithm 1. Then, by Lemmas 12 and 13, we know that either v discovers B' , or $\Delta(\mathcal{T}_C(A), B') \leq (1 + 48\beta\sqrt{d})\Delta(\mathcal{T}_O(A), B') \leq (1 + \epsilon)\Delta(\mathcal{T}_O(A), B')$. This means that either v or \mathcal{T}_C $(1 + \epsilon)$ -approximately discovers B' .

This completes the proof. ◀

From the above analysis, we immediately have our main results, Theorems 2 and 3.

7 Constructing ϵ -net for Hypercubes for P

To conclude this paper, we introduce efficient algorithms to construct a weak ϵ -net for the reference set P with axis-aligned hypercubic ranges. From the well known ϵ -net theorem, we know that a random sample of size $O((d'/\epsilon) \log(d'/\epsilon) + \log n/\epsilon)$ from P , where d' is the VC-dimension of the range space defined by hypercubes in \mathbb{R}^d (it is known that $d' \leq 2d$), is an ϵ -net with probability at least $1 - 1/n$. There are several previous results on ϵ -net for simple shapes like axis aligned rectangles, halfspace and disks in 2 or 3 dimension [20, 8, 2], which provide methods to build smaller size ϵ -nets. [19] provides a mathematical construction of ϵ -nets for axis-aligned hypercubes of size $O(1/\epsilon)$, which is optimal in size, although its efficient (*i.e.* in near $O(|P|)$ time) algorithmic implementation is unknown. [12] introduces a method to construct ϵ -nets for axis-aligned rectangles in any fixed dimension, which can be applied to generate the (ξ, μ) -dense subset Q . The running time of this method is $O(|P| \log^d |P|)$.

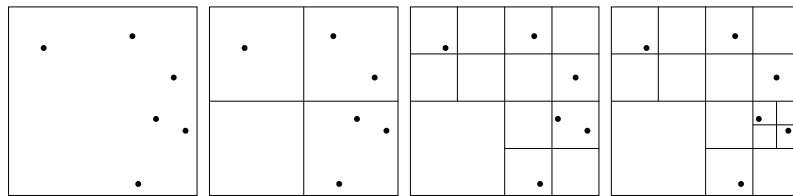
We show that if we further restrict the shapes to hypercubes, we are able to obtain an optimal size (*i.e.* $O(1/\epsilon)$) weak ϵ -net more efficiently. In the following we show how to deterministically construct a linear size $1/n$ -net Q for any multi-set P of size $O(mn)$, *i.e.*, a point set Q of size $O(n)$ such that if an axis-aligned hypercube G contains m points in

26:12 Small Candidate Set for Translational Pattern Search

P , it then contains at least one point in Q . The time of the construction is $O(nm \log nm)$ (Recall that the size of P is mn). Clearly, the same space and time complexity bounds for a $1/cn$ -net Q for any constant c are also achievable, thus proving Lemma 10. We also note that by applying the ϵ -net theorem, it is possible to construct such a Q of larger size ($O(n \log n)$), but in shorter ($O(n \log n)$) time, with high probability. This allows us to make a trade-off between the size of Q and the time complexity of the construction. We leave the discussion of the alternative construction to the end of section, and focus on the deterministic linear size ϵ -net construction in the following.

The construction is based on the quad-tree decomposition technique, which recursively partitions the regions inside the quad-tree boxes, and uses a 2^d -way tree structure to represent the partition. To build a quad-tree for P , we first start with a bounding box G which contains all points in P and is the root of the quad-tree. We then decompose G into 2^d smaller boxes with equal size, with each of them being a child of G . For each child box, we recursively perform the same decomposition. The recursion stops when a box contains no more than 1 point in P . The quad tree decomposition for P can be performed within $O(|P| \log |P|)$ time by maintaining a sorted list of P for each of the d axes [6], and compressing the tree properly to handle empty boxes during the decomposition.

Figure 3 below shows an example of quad-tree decomposition.



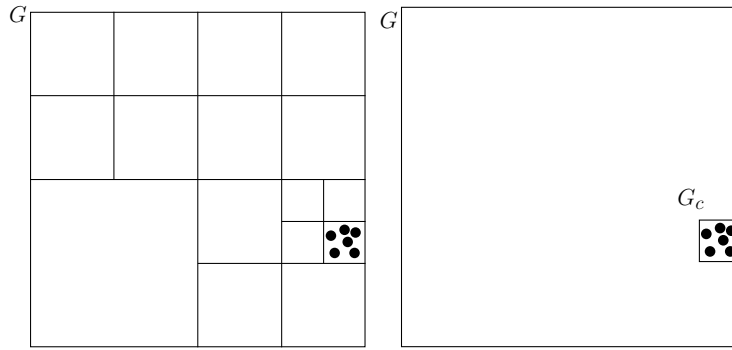
■ **Figure 3** Example of quad-tree decomposition.

A quad-tree decomposition may produce a large number of empty boxes, when a large number of points are aggregated in some region (see left of Figure 4 for an example). To resolve this issue, when decomposing a box G in the quad-tree decomposition, we first perform a quad-tree compression, which directly computes the smallest quad-tree box G_C that contains all the points in $P \cap G$ (see the right side of Figure 4). Then the quad-tree decomposition can continue on G_C . This will avoid generating many unnecessary empty boxes. Note that this compression step is not required if points are not concentrated, *i.e.*, if decomposing G yields at least 2 nonempty boxes (*i.e.*, containing points in P). In this case, we decompose G in the standard fashion.

With this compression step, the running time of the quad-tree decomposition is still $O(|P| \log |P|)$ [6].

Algorithm 2 and Algorithm 3 describe our quad-tree decomposition-based method for producing a weak ϵ -net Q . The decomposition scheme is a modification of the standard quad-tree decomposition. The main routine Algorithm 2 outputs the ϵ -net Q , together with a set U of boxes which is for analysis purpose. Algorithm 3 is the body for the recursion.

The Algorithm 2 and 3 are essentially trimmed versions of the standard quad-tree decomposition (by not decomposing some of the boxes in the process). Given a box G that contains multiple points in P , instead of simply decomposing it into 2^d sub-boxes and recursively building the quad-tree on them (which could generate boxes with few points in it and thus results in a quad-tree with high complexity), Algorithm 3 iteratively performs



■ **Figure 4** Example of quad-tree compression. It is possible that points in a box are aggregated at some location. Directly applying the quad-tree decomposition will generate many empty boxes. We can directly compute a box to contain all these points without really performing the decomposition.

■ **Algorithm 2** Construct- ϵ -Net.

Input: A set $P \subset \mathbb{R}^d$

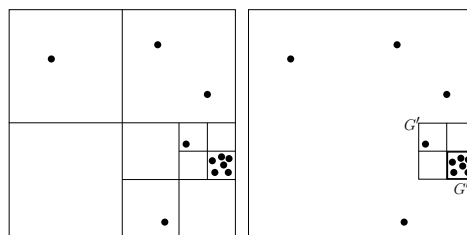
Output: An ϵ -net Q . A set U of \mathbb{R}^d boxes.

- 1: Initialize Q, U as empty sets. Initialize G as a box that contains P .
- 2: Start recursion by running Decompose-Single-Box subroutine on G

the quad-tree decomposition on *only one* sub-box which contains the maximum number of points in P , and tries to identify a box G' with the following properties. When the iteration (from step 4 to step 8) stops (at step 4 or 7), G' satisfies the following 2 conditions

1. There are less than $m/2^{d+1}$ points in $P \cap G \setminus G'$.
2. (a) All points in $P \cap G'$ have the same location, OR (b) There are at least $m/2^{d+1}$ points in $P \cap G \setminus G''$, where G'' is the child box of G' with the most number of points in P .

Only in case 2(b) we perform the recursion on the boxes generated by the decomposition of G' . See Figure 5 for illustration. In addition, we do not decompose G when there are only a small number ($\leq m/2^{d+1}$) of points in it. Since the algorithm is a trimmed version of the standard quad-tree decomposition, the running time is thus $O(|P| \log |P|) = O(mn \log mn)$.



■ **Figure 5** To achieve better performance, Algorithm 3 uses an iteration to find out G' with the desired properties. Recursion continues only (on sub-boxes of G') if $P \cap G \setminus G''$ contains an enough number of points, where G'' is the quad-tree child box of G' with the most number of points in P . This can greatly reduce the number of boxes generated.

It is quite clear that the size of Q is $O(n)$. The Decompose-Single-Box procedure stops immediately once the condition $|G \cap P| \leq m/2^{d+1}$ is satisfied. The procedure also makes sure that for any G'' of the 2^d child boxes generated for G (if the decomposition and recursion occur), inequality $|G \setminus G''| \geq m/2^{d+1}$ holds. This implies that the size of the recursion tree, and thus the size of U and Q , is $O(mn/(m/2^{d+1})) = O(n)$.

■ **Algorithm 3** Decompose-Single-Box.

Input: A box G

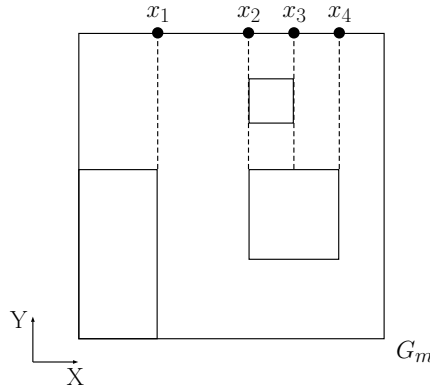
Output: A sub-quad tree with G as the root.

- 1: Add G into U . Add all the vertices of G to Q .
- 2: If G contains $\leq m/2^{d+1}$ points in P , return.
- 3: Initialize variable G' to be the box G .
- 4: If all points in P that lie in G' coincide at point p . Put p and all vertices of G' into Q . Put G' into U . Return.
- 5: Update variable G' to be the resulting box from the quad-tree compression of the current G' , if necessary (See Appendix A.3).
- 6: Decompose G' equally into 2^d sub-boxes. Let G'' be the one that contains the most number of points in P .
- 7: If $G \setminus G''$ contains $\geq m/2^{d+1}$ points in P , recursively call Decompose-Single-Box on the 2^d sub-boxes generated in the above step. Then return.
- 8: Otherwise, update variable G' to be G'' . Go to step 4.

► **Lemma 15.** *Set Q generated by the above algorithm is a weak $1/n$ -net for P , i.e., if any axis-aligned hypercube G_m contains at least m points in P , then G_m contains at least 1 point in Q .*

Proof. Let U_m denote the subset $U_m \subseteq U$ of hyperboxes G' in U such that the interior of G_m intersects G' .

For each coordinate axis e of \mathbb{R} , let $F(e)$ be the set of faces f of boxes in U_m , such that f is perpendicular to e and intersects the interior of G_m . Let the *cutting number* $x(e)$ of e be the possible number of distinct coordinate values of faces in $F(e)$ in the e axis. (See Figure 6 for an example.) We consider two cases.



■ **Figure 6** Example of cutting number for X axis in an configuration of interior G_m . In this example there are 3 boxes of U_m intersecting G_m . There are 4 different x values for faces of these boxes that lies in G_m and are perpendicular to the X axis. Thus the cutting number for X axis in this example is 4.

Case 1. $x(e) \leq 1$ for any axis e . Then boxes in U partitions G_m into no more than 2^d regions, since in every direction G_m is “cut” by boxes in U at most once. Since G_m contains at least m points in P , one of the regions will contain strictly more than $m/2^d$ points. However, from Algorithm 3, we know that all of the regions formed by U can

only have no more than $m/2^{d+1}$ points, with the exception that for some regions, all its contained points in P have the same location p . Thus, G_m intersects such a region and contains p . Since $p \in Q$ from Algorithm 3 (see Step 4), this case is proved.

Case 2. $x(e) \geq 2$ for some axis e . Since the quad-tree decomposition always divides boxes equally, the following fact is clear.

Fact. Let f_1, f_2 be the faces of boxes G_1 and G_2 in U , respectively, such that they are facing the same direction e . If the distance between f_1 and f_2 in the direction of e is $l > 0$, one of G_1 and G_2 has edge length $\leq l$.

If $x(e) \geq 2$, then there exist faces f_1 and f_2 facing the direction of e and intersect the interior of G_m . If these 2 faces belongs to the same box $G_f \in U$, then G_f is smaller than G_m in size. Thus, one of the vertices p_f of G_f must lie in G_m . From Algorithm 3, we know that $p_f \in Q$. Therefore, $G_m \cap Q \neq \emptyset$. If f_1 and f_2 belong to different boxes, say G_1 and G_2 in U , from the above fact, we know that the edge length of one of the boxes will be no larger than the distance between f_1 and f_2 in the direction of e . Since the box size is smaller than G_m , one of its vertices is in G_m . This again leads to the fact that $G_m \cap Q \neq \emptyset$. This completes the proof. \blacktriangleleft

Alternative Construction Using ϵ -net Theorem. Recall that the ϵ -net Theorem allows us to build a $1/cn$ -net of P with size $O(n \log n)$ by using $O(n \log n)$ samples, where $c > 0$ is a constant. Also note that, it is not necessary to explicitly compute P (whose size is mn) before conducting the sampling. From the definition of P (in Section 2, Definition 1), a random sample from P can be obtained by first sampling a from A , b from B , and then computing $b - a$ as the sample. This allows us to build a $1/cn$ -net of P in $O(n \log n)$ time.

References

- 1 Helmut Alt and Leonidas J Guibas. Discrete geometric shapes: Matching, interpolation, and approximation. In *Handbook of computational geometry*, pages 121–153. Elsevier, 2000.
- 2 Boris Aronov, Esther Ezra, and Micha Sharir. Small-Size ϵ -Nets for Axis-Parallel Rectangles and Boxes. *SIAM Journal on Computing*, 39(7):3248–3282, 2010.
- 3 Sunil Arya, Theocharis Malamatos, and David M Mount. Space-time tradeoffs for approximate nearest neighbor searching. *Journal of the ACM (JACM)*, 57(1):1, 2009.
- 4 Rinat Ben-Avraham, Matthias Henze, Rafel Jaume, Balázs Keszegh, Orit E Raz, Micha Sharir, and Igor Tubis. Minimum partial-matching and Hausdorff RMS-distance under translation: combinatorics and algorithms. In *European Symposium on Algorithms*, pages 100–111. Springer, 2014.
- 5 Sergio Cabello, Panos Giannopoulos, and Christian Knauer. On the parameterized complexity of d-dimensional point set pattern matching. In *International Workshop on Parameterized and Exact Computation*, pages 175–183. Springer, 2006.
- 6 Paul B Callahan and S Rao Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *Journal of the ACM*, 42(1):67–90, 1995.
- 7 Danny Z Chen, Ziyun Huang, Yangwei Liu, and Jinhui Xu. On Clustering Induced Voronoi Diagrams. *SIAM Journal on Computing*, 46(6):1679–1711, 2017.
- 8 Kenneth L Clarkson and Kasturi Varadarajan. Improved approximation algorithms for geometric set cover. *Discrete & Computational Geometry*, 37(1):43–58, 2007.
- 9 Hu Ding, Ronald Berezney, and Jinhui Xu. k-prototype learning for 3d rigid structures. In *Advances in Neural Information Processing Systems*, pages 2589–2597, 2013.
- 10 Hu Ding, Branislav Stojkovic, Ronald Berezney, and Jinhui Xu. Gauging association patterns of chromosome territories via chromatic median. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1296–1303, 2013.

- 11 Hu Ding and Jinhui Xu. FPTAS for minimizing earth mover’s distance under rigid transformations. In *European Symposium on Algorithms*, pages 397–408. Springer, 2013.
- 12 Esther Ezra. A note about weak ϵ -nets for axis-parallel boxes in d -space. *Information Processing Letters*, 110(18-19):835–840, 2010.
- 13 Martin Gavrilov, Piotr Indyk, Rajeev Motwani, and Suresh Venkatasubramanian. Combinatorial and experimental methods for approximate point pattern matching. *Algorithmica*, 38(1):59–90, 2004.
- 14 Michael T Goodrich, Joseph SB Mitchell, and Mark W Orletsky. Practical methods for approximate geometric pattern matching under rigid motions:(preliminary version). In *Proceedings of the tenth annual symposium on Computational geometry*, pages 103–112. ACM, 1994.
- 15 Sariel Har-Peled. A replacement for Voronoi diagrams of near linear size. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 94–103. IEEE, 2001.
- 16 David Haussler and Emo Welzl. ϵ -nets and simplex range queries. *Discrete & Computational Geometry*, 2(2):127–151, 1987.
- 17 Matthias Henze, Rafel Jaume, and Balázs Keszegh. On the complexity of the partial least-squares matching Voronoi diagram. In *Proc. 29th European Workshop on Computational Geometry*, pages 193–196, 2013.
- 18 Daniel P Huttenlocher, Klara Kedem, and Micha Sharir. The upper envelope of Voronoi surfaces and its applications. *Discrete & Computational Geometry*, 9(3):267–291, 1993.
- 19 Janardhan Kulkarni and Sathish Govindarajan. New ϵ -net constructions. In *Proceedings of the 22nd Annual Canadian Conference on Computational Geometry, Winnipeg, Manitoba, Canada*, pages 159–162. Citeseer, 2010.
- 20 Jiří Matoušek, Raimund Seidel, and Emo Welzl. How to net a lot with little: Small ϵ -nets for disks and halfspaces. In *Proceedings of the sixth annual symposium on Computational geometry*, pages 16–22. ACM, 1990.
- 21 Günter Rote. Partial least-squares point matching under translations. In *Proc. 26th European Workshop on Computational Geometry*, pages 249–251. Citeseer, 2010.
- 22 Nitasha Sehgal, Andrew J Fritz, Jaromira Vecerova, Hu Ding, Zihe Chen, Branislav Stojkovic, Sambit Bhattacharya, Jinhui Xu, and Ronald Berezney. Large-scale probabilistic 3D organization of human chromosome territories. *Human molecular genetics*, 25(3):419–436, 2015.

A Appendix

A.1 Proof of Corollary 4

Proof. Suppose that translation \mathcal{T} and $B' \subseteq B$ realize the minimum cost bipartite matching with A . Let C_{OPT} be the minimum bipartite matching cost of B' and $\mathcal{T}(A)$. C_{OPT} is then the optimal minimum bipartite matching cost between A and B under translations. Since B' is a C_{OPT} -instance of A , there exists $\mathcal{T}' \in \mathbb{T}$ such that the matching cost between B' and $\mathcal{T}'(A)$ is no larger than $(1 + \epsilon)C_{OPT}$. Thus, \mathcal{T}' induces a $(1 + \epsilon)$ -approximation for the optimal translational matching between A and B . ◀

A.2 Proof of Lemma 5

Proof. Let ϕ_1 (or ϕ_2) be the corresponding bipartite matching which gives rise to the minimum cost between B' and $\mathcal{T}_1(A)$ (or $\mathcal{T}_2(A)$). Then, $\Delta(\mathcal{T}_1(A), B') = \sum_{q \in P(\phi_1)} \|q - \mathcal{T}_1\|$, and $\Delta(\mathcal{T}_2(A), B') = \sum_{q \in P(\phi_2)} \|q - \mathcal{T}_2\|$. Note that

$$\begin{aligned}
\sum_{q \in P(\phi_1)} \|q - \mathcal{T}_2\| &= \sum_{q \in P(\phi_1)} \|q - \mathcal{T}_1 - \mathcal{T}_2 + \mathcal{T}_1\| \\
&\leq \sum_{q \in P(\phi_1)} \|\mathcal{T}_2 - \mathcal{T}_1\| + \sum_{q \in P(\phi_1)} \|q - \mathcal{T}_1\| \\
&= m\|\mathcal{T}_2 - \mathcal{T}_1\| + \Delta(\mathcal{T}_1(A), B') \\
&\leq m\epsilon\|p - \mathcal{T}_1\| + \Delta(\mathcal{T}_1(A), B') \\
&\leq \epsilon\Delta(\mathcal{T}_1(A), B') + \Delta(\mathcal{T}_1(A), B') \\
&= (1 + \epsilon)\Delta(\mathcal{T}_1(A), B'),
\end{aligned}$$

where the first inequality comes from the triangle inequality, and the third inequality comes from the fact that p is the nearest neighbor of \mathcal{T}_1 in P , which implies that $m\|p - \mathcal{T}_1\| \leq \sum_{q \in P(\phi_1)} \|q - \mathcal{T}_1\| = \Delta(\mathcal{T}_1(A), B')$. From the assumption that ϕ_2 is the minimum cost bipartite matching between $\mathcal{T}_2(A)$ and B' , we know that the value $\sum_{q \in P(\phi_1)} \|q - \mathcal{T}_2\|$, which is the matching cost between $\mathcal{T}_2(A)$ and B' under ϕ_1 , must be no smaller than $\Delta(\mathcal{T}_2(A), B')$. Therefore, from the above inequality, we have $\Delta(\mathcal{T}_2(A), B') \leq (1 + \epsilon)\Delta(\mathcal{T}_1(A), B')$.

Following a similar argument, we also have

$$\begin{aligned}
\Delta(\mathcal{T}_1(A), B') &\leq \sum_{q \in P(\phi_2)} \|q - \mathcal{T}_1\| \\
&= \sum_{q \in P(\phi_2)} \|q - \mathcal{T}_2 - \mathcal{T}_1 + \mathcal{T}_2\| \\
&\leq \sum_{q \in P(\phi_2)} \|\mathcal{T}_2 - \mathcal{T}_1\| + \sum_{q \in P(\phi_2)} \|q - \mathcal{T}_2\| \\
&= m\|\mathcal{T}_2 - \mathcal{T}_1\| + \Delta(\mathcal{T}_2(A), B') \\
&\leq m\epsilon\|p - \mathcal{T}_1\| + \Delta(\mathcal{T}_2(A), B') \\
&\leq \frac{\epsilon}{1 - \epsilon}\Delta(\mathcal{T}_2(A), B') + \Delta(\mathcal{T}_2(A), B') \\
&= (1 - \epsilon)^{-1}\Delta(\mathcal{T}_2(A), B'),
\end{aligned}$$

where the fourth inequality comes from the following argument. The closest distance from a point in P to \mathcal{T}_1 is $\|p - \mathcal{T}_1\|$. Since $\|\mathcal{T}_1 - \mathcal{T}_2\| \leq \epsilon\|p - \mathcal{T}_1\|$, we know that the closest distance from a point in P to \mathcal{T}_2 is at least $(1 - \epsilon)\|p - \mathcal{T}_1\|$. Therefore, $\Delta(\mathcal{T}_2(A), B') = \sum_{q \in P(\phi_2)} \|q - \mathcal{T}_2\| \geq m(1 - \epsilon)\|p - \mathcal{T}_1\|$.

Putting everything together, we have that $(1 - \epsilon)\Delta(\mathcal{T}_1(A), B') \leq \Delta(\mathcal{T}_2(A), B') \leq (1 + \epsilon)\Delta(\mathcal{T}_1(A), B')$. Thus, the lemma follows. \blacktriangleleft

The Weighted k -Center Problem in Trees for Fixed k

Binay Bhattacharya

Simon Fraser University, Burnaby, Canada

binay@cs.sfu.ca

Sandip Das

Indian Statistical Institute, Kolkata, India

sandipdas@isical.ac.in

Subhadeep Ranjan Dev

Indian Statistical Institute, Kolkata, India

srdev_r@isical.ac.in

Abstract

We present a linear time algorithm for the weighted k -center problem on trees for fixed k . This partially settles the long-standing question about the lower bound on the time complexity of the problem. The current time complexity of the best-known algorithm for the problem with k as part of the input is $O(n \log n)$ by Wang et al. [15]. Whether an $O(n)$ time algorithm exists for arbitrary k is still open.

2012 ACM Subject Classification Theory of computation → Facility location and clustering; Theory of computation → Network optimization

Keywords and phrases facility location, prune and search, parametric search, k -center problem, conditional k -center problem, trees

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.27

1 Introduction

In this paper, we study a popular facility location problem on graphs called the weighted k -center problem. The sites are the vertices of the graph and have positive weights associated with them. The edges of the graph have positive lengths and the facilities can be placed anywhere on them. The weighted distance between a facility and a site is the length of the shortest path between them times the weight of the site. Our objective is to place k facilities on the graph such that the maximum weighted distance of a site to its closest facility is minimized.

Kariv and Hakimi [9] in 1979 showed that the weighted k -center problem on general graphs is NP-hard. In fact, they proved a much stronger statement that finding the weighted k -center is NP-hard for planar graphs with maximum vertex degree 3. On the other hand in the same paper, they gave an $O(n^2 \log n)$ time algorithm for the problem if the underlying graph is a tree with n vertices. Later improvements in time complexity were done by Jeger and Kariv [8] to $O(kn \log n)$, and Megiddo and Tamir [12] to $O(n \log^2 n)$ using Cole's [6] optimization. Very recently in 2016 Banik et. al. [1] gave an $O(n \log n + k \log^2 n \log(n/k))$ time algorithm for the problem which was then improved to $O(n \log n)$ by Wang and Zhang [15] in 2018. Wang and Zang's solution is the current state of the art for the weighted k -center problem in trees for arbitrary value of k .

For $k = 1$ the problem had already been solved by Megiddo [11] in $O(n)$ time in 1983. In 2006 Ben-Moshe et al. [2] showed that the 2-center problem can also be computed in $O(n)$ time. In the same paper, they gave an $O(n \log n)$ time algorithm for the weighted k -center problem when k is 3 or 4. The problem has also been studied in the real line setting



© Binay Bhattacharya, Sandip Das, and Subhadeep Ranjan Dev;
licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 27; pp. 27:1–27:11

Leibniz International Proceedings in Informatics



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

where the sites and facilities are constrained to be placed on a given line. In this setting Bhattacharya et al. [5] gave an $O(n)$ time solution for the weighted k -center problem for any fixed k . For the unweighted case, the k -center problem on trees has been solved optimally by Frederickson [7] in $O(n)$ time. Karmakar et al. [10] studied some constrained version of this problem in \mathbb{R}^2 and gave algorithms which are near linear in n .

In this paper, we study the weighted k -center problem on trees for any constant k and present an $O(n)$ time solution for it. Our algorithm generalises the technique used in Bhattacharya et al. [4] where they give a linear time algorithm for finding the k -step fitting function of n points in \mathbb{R}^2 . A different (unpublished) linear time solution to the weighted k -center problem was also suggested in 2008 by Shi [14]. The algorithm presented in this paper is simpler and is based on the fact that a generalization of Megiddo's [11] approach applies suitably to our problem. This enables us to prune vertices from a subgraph of the tree which in turn produces a linear time algorithm.

The rest of the paper is organized as follows. In Section 2 we define the weighted k -center problem and the conditional weighted k -center problem. We also briefly mention the r -feasibility test. In Section 3 we introduce the notion of a *big-component* of a tree and propose a linear time algorithm to find it. In Section 4 we show how to prune the vertices of the big-component in order to reduce the size of the original tree. In Section 5 we present our linear time algorithm for the conditional weighted k -center problem which is a generalisation of the weighted k -center problem. We then conclude in Section 6.

2 Preliminaries

2.1 Problem Definition

Let T be a tree with vertex set $V(T)$ and edge set $E(T)$. Also, let the number of vertices in T , denoted by $|T|$, be n . Each vertex $v \in V(T)$ is associated with a positive weight $w(v)$ and each edge $e \in E(T)$ is associated with a positive length $l(e)$. To define the notion of points on an edge e , we assume e to be a line segment with length $l(e)$. The distance between two points x and y in e is proportional to the portion of e in between x and y with respect to $l(e)$. $A(T)$ denotes the set of all points on all edges of T .

The distance between any two points $x, y \in A(T)$, denoted by $l(x, y)$, is the sum of the lengths of the edges and the partial edges in the unique path between x and y in T . The weighted distance between a vertex $v \in V(T)$ and a point $x \in A(T)$ is defined as $d(x, v) = w(v) \cdot l(x, v)$. We extend this definition to include the weighted distance between a set of points $X \subseteq A(T)$ and a set of vertices $V' \subseteq V(T)$ which is given by the expression

$$d(X, V') = \max_{v \in V'} \{ \min_{x \in X} \{ d(x, v) \} \}$$

The objective of the weighted k -center problem on T is to find a set of points $X \subset A(T)$ with $|X| = k$ such that $d(X, V(T))$ is minimum. The points in X are called *centers*.

We solve the weighted k -center problem by solving a more general problem on trees which is called the *conditional* weighted (p, S) -center problem or the (p, S) -center problem in short. The problem was first introduced in Minięka [13]. Here, $S \subset A(T)$ is the set of centers already placed in T . We call S the set of *old centers*. The objective of the (p, S) -center problem is to find a set of points $X \subset A(T)$ with $|X| = p$ such that $d(X \cup S, V(T))$ is minimized. We call X the set of *new centers*. Observe that the solution to the (p, S) -center problem is also the solution to the weighted k -center problem, when $S = \emptyset$ and $p = k$.

2.2 The r -feasibility test

Let X be an optimal solution to the (p, S) -center problem in T . The optimal radius r^* is defined as the maximum weighted distance of a vertex to its closest center in $X \cup S$ i.e. $r^* = d(X \cup S, V(T))$. A point $x \in A(T)$ is said to *cover* a vertex $v \in V(T)$ with radius r if $d(x, v) \leq r$. Similarly, we say that a set of centers X' covers a set of vertices V' with radius r if $d(X', V') \leq r$. If no radius is mentioned, we assume r to be r^* .

The r -feasibility test for the (p, S) -center problem on T takes as input a radius r and returns (a) *feasible* if $r \geq r^*$ and, (b) *infeasible* if $r < r^*$.

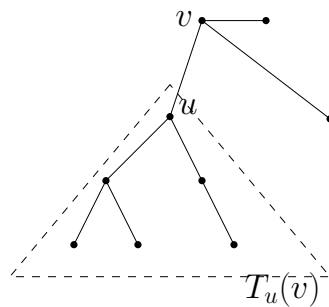
The algorithm was first presented by Kariv and Hakimi [9] in 1979 for the weighted k -center problem. The feasibility test for the (p, S) -center problem follows the same principle and is described in Shi [14]. The feasibility test takes $O(n)$ time.

2.3 Our Approach

The solution to the (p, S) -center problem presented here is recursive in nature. We first introduce the notion of a *big-component* of T . For a suitable constant c , a big-component is a subtree T' of T with at least $\frac{n}{c}$ vertices. Note that $T \setminus T'$ can be disconnected. It has the additional property that for some optimal solution X to the (p, S) -center problem the vertices of the big-component are “covered” by at most one new center from X . This property enables us to use the prune and search technique introduced by Megiddo [11], and generalized by Shi [14], to prune a constant fraction of the vertices of the big-component. The pruning step leads to a new tree T'' with a fraction of the number of vertices in T . We recursively perform the last two steps of finding a big-component and then pruning its vertices on these new tree T'' and the trees that follow until the size of the tree falls below a certain threshold. We then use any brute force technique to calculate the (p, S) -center in this final tree. The pruning step guarantees that the (p, S) -center solution of this reduced tree is also the (p, S) -center solution to the original tree T .

3 Big-Component

In this section we define a big-component and provide linear time algorithm to find it. Let $V \subseteq V(T)$ be such that the subgraph of T induced by V is connected. We call this induced subgraph an *induced subtree* of T . Let $V_u(v) = \{w \in V(T) \mid \text{the path from } v \text{ to } w \text{ passes through } u\}$; then $T_u(v)$ denotes the subtree induced by $V_u(v)$ and rooted at u (see Figure 1). Let $N(v)$ denote the neighbouring vertices of v in T , and let $N_{T'}(v)$ denotes the neighbouring vertices of v in the subtree T' of T . For $m \in \mathbb{Z}^+$, $[m]$ denotes the sequence $\{1, 2, \dots, m\}$.



■ **Figure 1** $T_u(v)$ is a subtree of T rooted at vertex u and not containing v .

We now introduce the notion of a big-component of T with respect to the (p, S) -center problem. Here, $k = p + |S|$.

► **Definition 1.** Let B be a subtree of T with at least $\frac{n}{2k}$ vertices and let $S_B \subseteq S$ be the old centers contained in it. B is a big-component of T with respect to the (p, S) -center problem if for some solution X to the problem all vertices of B are covered by either (a) S_B or (b) a single new center $x \in X$.

If B satisfies (a) we call B a type-a big-component and if it satisfies (b) we call it a type-b big-component.

In order to find a big-component, we first find a sequence of “candidate subtrees” of T . One of these candidate subtrees is a big-component of T . We search the subtrees in the candidate sequence sequentially until one of them identifies as a big-component.

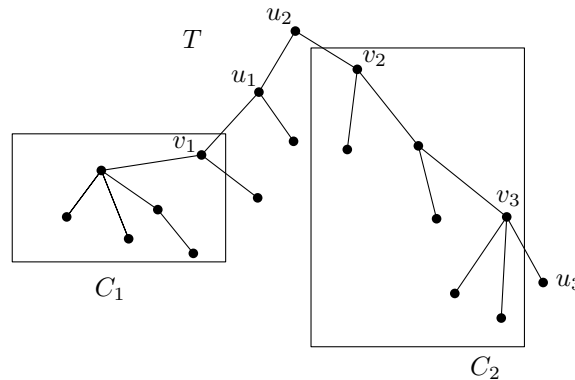
3.1 Candidate Sequence

► **Definition 2.** A subtree C of T is a candidate subtree of T if

1. $\frac{n}{2k} \leq |C| < \frac{n}{k}$,
2. C is connected to the rest of T i.e. $T \setminus C$ by a single vertex

The vertex through which C is connected to the rest of T is called the exit vertex of C and is denoted by $v(C)$.

Note that removing all vertices of C except $v(C)$ from T still keeps the rest of T i.e. $T \setminus C \cup \{v(C)\}$ connected.



■ **Figure 2** C_1 with exit vertex v_1 is a candidate subtree of T but C_2 is not. Here $n = 18$ and $k = 2$.

Let $T_1 = T$. For $i = 2, 3, \dots$ we define T_i to be the tree generated by removing all vertices of C_{i-1} except $v(C_{i-1})$ from T_{i-1} . Here, C_i is a candidate subtree of T_i .

► **Definition 3.** Let $\langle C_i \rangle_{i=1}^m = \langle C_1, C_2, \dots, C_m \rangle$ be a sequence of m subtrees of T such that C_i is a candidate subtree of T_i . We call this sequence a candidate sequence of T .

We now describe an algorithm to find a candidate sequence of T with respect to the (p, S) -center problem. Here we assume $k \ll n$.

► **Algorithm 1.**

Input: Tree T .

Output: A candidate sequence of T .

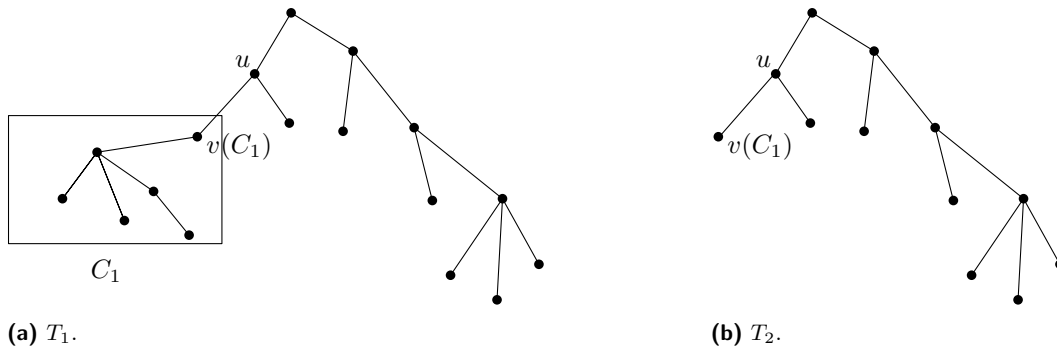


Figure 3 T_2 is formed by deleting all vertices of C_1 except $v(C_1)$.

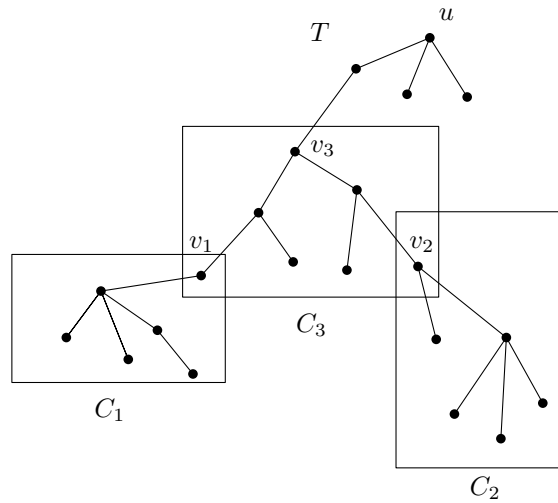


Figure 4 First C_1 is reported as a candidate subtree, then C_2 and then C_3 .

- Step 1:** We consider $T' = T$ and assume it to be rooted at an arbitrary vertex u . We traverse the vertices of T' by first visiting the leaf vertices of T' and then visiting any vertex whose all children have already been visited. For each vertex visited we perform Step 2 until $|T'| < \frac{n}{2k}$.
- Step 2:**
- a. Let v be the current vertex in our traversal. All vertices in $T'_v(u)$ have already been visited. If $|T'_v(u)| < \frac{n}{2k}$ we mark v as visited and continue to the next vertex in our traversal. Otherwise, if $|T'_v(u)| \geq \frac{n}{2k}$ we do the next step.
 - b. Let v_1, v_2, \dots, v_p be the children of v and let q be the smallest integer such that the subtree $C = \bigcup_{i=1}^q T'_{v_i}(v)$ has at least $\frac{n}{2k}$ vertices. Since for each child $v_i, |T'_{v_i}(v)| < \frac{n}{2k}$, we have that $|C| \leq \frac{n}{k}$. We report C as a candidate subtree with $v(C) = v$ and update T' by deleting all vertices of C except $v(C)$. We repeat Step 2b on v until $|T'_v(u)| < \frac{n}{2k}$.

The sequence of candidate subtrees found by Algorithm 1 is a candidate sequence of T . For an example see Figure 4.

- **Time Complexity.** Since Algorithm 1 performs a single traversal of T , it takes $O(n)$ time.
- **Observation 1.** The length of the candidate sequence generated by Algorithm 1 is at least k and at most $2k$.

Proof. The maximum size of a candidate subtree is $\lfloor \frac{n}{k} \rfloor$. Our algorithm stops only when the size of T_i falls below $\frac{n}{2k}$. Therefore, the length of the candidate sequence is at least k . Similarly, the minimum size of a candidate subtree is $\lceil \frac{n}{2k} \rceil$ and hence the length of the candidate sequence is at most $2k$. ◀

The next lemma justifies the need of finding a candidate sequence in order to find a big-component of T .

► **Lemma 4.** *Let $\langle C_i \rangle_{i=1}^m$ be a candidate sequence generated by Algorithm 1. Then there exists a $C_j \in \langle C_i \rangle_{i=1}^m$ which is a big-component of T .*

Proof. Let X be any optimal solution to the (p, S) -center problem. The covering of $X \cup S$ partitions T into k subtrees. These subtrees exclude exactly $k - 1$ edges of T from it. Since, $m \geq k$, there exists at least one candidate subtree C_j which contains none of these $k - 1$ edges. Then, by definition, this C_j is a big-component. ◀

3.2 The Big-Component Algorithm

We now present an algorithm to find a big-component B in T . The algorithm sequentially examines whether a subtree in the candidate sequence of T is a big-component. The algorithm is as follows.

► **Algorithm 2.**

Input: Tree T , integer p and old centers S .

Output: A big-component B of T with respect to the (p, S) -center problem.

Step 1: Compute a candidate sequence $\langle C_i \rangle_{i=1}^m$ of T using Algorithm 1. For each candidate subtree C_i , $i = 1, 2, \dots, k - 1$ ($k \leq m$) we do Step 2 and Step 3.

Step 2: Let C_j be the current candidate subtree. Let $\langle C_{i_p} \rangle_{p=1}^{\hat{j}}$ be the longest subsequence of $\langle C_i \rangle_{i=1}^{j-1}$ such that the subtree $\hat{C}_j = C_j \cup (\bigcup_{p=1}^{\hat{j}} C_{i_p})$ is connected. Note that \hat{C}_j can be computed using a standard tree traversal. See Figure 5 for an example. Also let \hat{S}_j be the old centers in \hat{C}_j .

Step 3: We define p_j to be the number of candidate subtrees in \hat{C}_j which do not contain any old centers. We do the following:

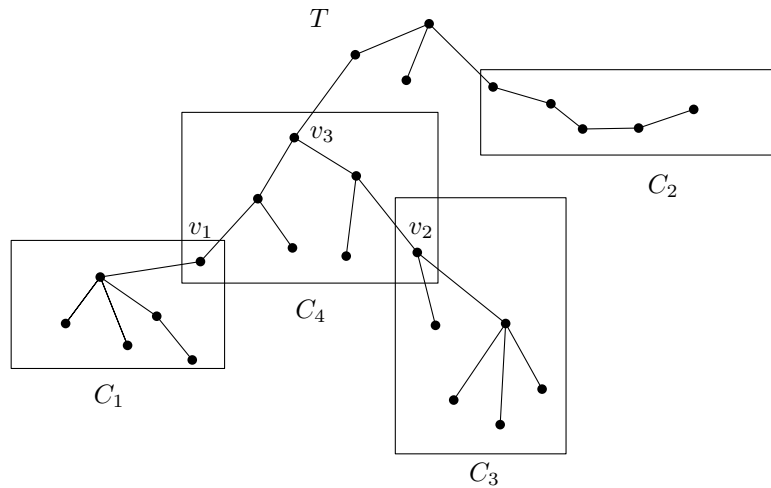
a. If $p_j < p$, we compute the (p_j, \hat{S}_j) -center on \hat{C}_j and set r_j to be its radius. We do an r_j -feasibility test on T with respect to the (p, S) -center problem. We declare C_j to be a big-component and stop if the feasibility test returns infeasible.

b. If $p_j \geq p$, we compute the $(0, S \setminus \hat{S}_j)$ -center in $(T \setminus \hat{C}_j) \cup \{v(C_j)\}$ and set r_j to be its radius. Do an r_j -feasibility test on T . We declare C_j to be a big-component and stop if the feasibility test returns feasible.

Step 4: If no subtree of $\langle C_i \rangle_{i=1}^{k-1}$ has been returned as a big-component in the above steps then we return C_k as a big-component of T . ◀

► **Lemma 5.** *The candidate subtree returned by Algorithm 2 is a big component of T .*

Proof. Consider a candidate subtree C_j and suppose it is not returned as a big-component. Then for the case $p_j < p$ the r_j -feasibility test returned feasible. Here, we assume $r_j > r^*$ since otherwise we have an optimal solution. This implies that in some optimal solution to the (p, S) -center problem on T , p_j new centers along with the old centers \hat{S}_j are not enough to cover \hat{C}_j . For the case $p_j \geq p$, the r_j -feasibility test returned infeasible. Which implies that in the optimal case the old centers $S \setminus \hat{S}_j$ have to cover more than the vertices



■ **Figure 5** $\langle C_i \rangle_{i=1}^4$ is a candidate sequence of T . $\hat{C}_4 = C_4 \cup (C_1 \cup C_3)$. Note that we have already tested C_1, C_2 and C_3 for big-component.

in $T \setminus \hat{C}_j \cup \{v(C_j)\}$. This in turn implies that p new centers along with \hat{S}_j are not sufficient to cover \hat{C}_j .

By a similar argument we can show that if C_j is returned as a big-component then at most $\min\{p, p_j\}$ centers are enough to cover the vertices in \hat{C}_j . And since all previous candidate subtrees $C_i, i \in [j - 1]$ were not returned as big-components, it implies that C_j is either covered by the old centers in C_j or exactly one new center (if C_j contains no old centers). Therefore C_j , by definition is a big-component.

Now, if C_k was returned as a big-component then, to cover all vertices in $\bigcup_{i=1}^{k-1} C_i$ at least $k - 1$ centers (old and new) are necessary. This in turn implies that C_k is completely covered by just 1 center, which again by definition is a big-component. ◀

► **Time Complexity.** Step 1 takes $O(n)$ time as shown in Section 3.1. Step 2 and step 3 iterates at most $k - 1$ times. On the j^{th} iteration, the time taken to execute these two steps is $T_{total}(p - 1, j \frac{n}{k}) + O(n)$. Here, $T_{total}(p, n)$ is the time required to compute the (p, S) -center problem on a tree with n vertices. Therefore the time taken by Algorithm 2 is given by the following recurrence.

$$T_{big}(p, n) \leq \begin{cases} \sum_{j=1}^{k-1} T_{total}(p - 1, j \frac{n}{k}) + O(kn) & , p \geq 1 \\ O(n) & , p = 0 \end{cases}$$

In this section we proposed an algorithm to find a big-component of T . In the rest of the paper, we present an algorithm to prune a constant fraction of vertices from the big-component without affecting solution of the (p, S) -center in T .

4 Vertex Pruning from a Big-Component

Pruning a vertex is analogous to deleting a vertex but with certain differences. While pruning a vertex v we take actions based on the degree of v .

If v is a degree 1 vertex and its incident edge does not contain an old center from S , we simply delete the vertex and its edge from T . Otherwise, v is not deleted. If v is a degree 2 vertex with neighbours v_1 and v_2 joined by edges e_1 and e_2 then, we delete v, e_1 and e_2 and

join v_1 and v_2 with a new edge e with $l(e) = l(e_1) + l(e_2)$. If e_1 and e_2 had old centers, we place them on e at appropriate locations. If v is a vertex of degree 3 or more then we do not immediately delete it; instead we flag it for later deletion. The flagged vertices do not contribute to the size of the candidate subtree or the big-component. But, they do contribute when considering the degree of other vertices to be pruned. When we delete a vertex v , we also recursively delete any adjacent flagged vertex whose current degree has fallen below 3.

Let T' be the tree generated after pruning some vertices from T . T' can contain vertices of T which have been pruned out but have not been deleted. We have the following lemma.

► **Observation 2.** *The number of vertices in T' is proportional to the number of unpruned degree 1 and degree 2 vertices of T in T' .*

Proof. Let n_1 be the number of degree 1 vertices, n_2 the number of degree 2 vertices and n_3 the number of degree 3 or more vertices of T' . If n is the total number of vertices in T' then $n = n_1 + n_2 + n_3$. Since, the number of leaf vertices in a tree is always greater than the number of degree 3 or more vertices, $n_1 \geq n_3$. Therefore $n = n_1 + n_2 + n_3 \leq 2n_1 + n_2$. Here, n_2 does not count any pruned vertex of T which are still present in T' . The number of degree 1 vertices of T which have been pruned and yet have not been deleted from T' and which are counted in n_1 is at most k , a constant. Therefore, the claim holds. ◀

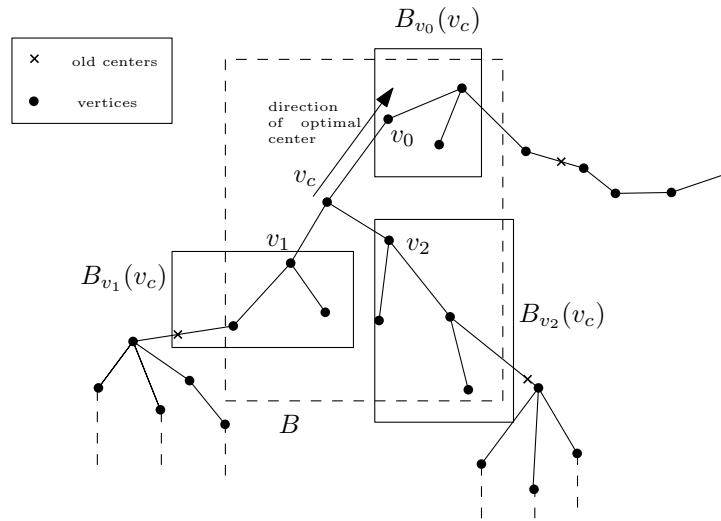
We are now ready to present an algorithm which prunes a constant fraction of the vertices of a big-component B without affecting the optimal solution to the (p, S) -center problem.

► **Algorithm 3.**

Input: Tree T , integer p , old centers S and a big-component B .

Output: Prune a constant fraction of vertices of B .

- Step 1:** If B has old centers present in it then B is a type-a big-component. Let $v_{far} \in V(B)$ be a vertex such that $d(S_B, v_{far}) = d(S_B, V(B))$. We prune all vertices in $V(B)$ except v_{far} and stop.
Else B is a type-b big-component and we proceed to the next step.
- Step 2:** We find a centroid v_c (a centroid is a vertex of tree whose removal splits the tree into forest such that all trees in the forest have size at most half the original tree. Such a vertex can be found out in time linear to the number of vertices) of B and compute the $(p-1, S \cup \{v_c\})$ -center in T . Let the set of new centers be X and let $r_{far} = d(X \cup S \cup \{v_c\}, V(T))$. Let V_{far} be the set of vertices of T at a weighted distance r_{far} from their closest center i.e. $V_{far} = \{v_i \in V(T) \mid d(X \cup S \cup \{v_c\}, v_i) = r_{far}\}$.
- Step 3:** If the vertices in V_{far} lie in more than one subtree $T_{v_i}(v_c)$, $v_i \in N(v_c)$ then moving the center placed at v_c away from the vertex in any direction will only increase the covering radius. Therefore, we can conclude that $r_{far} = r^*$ and $X \cup \{v_c\}$ is an optimal (p, S) -center solution of T . Let $v_{far} \in V_{far}$ be any arbitrary vertex. We prune all vertices in $V_{far} \setminus \{v_{far}\}$ and stop.
Else V_{far} is a subset of the vertices in a single subtree (say) $T_{v_0}(v_c)$ for some $v_0 \in N(v_c)$, we go to the next step.
- Step 4:** Since V_{far} is a subset of the vertices of $T_{v_0}(v_c)$, shifting the center from v_c in the direction of v_0 will reduce the covering radius. Therefore, the center which will cover the vertices of B lies in the direction of v_0 from v_c . Let V_{rest} be the set of vertices in the union of the subtrees $B_{v_i}(v_c)$, $\forall v_i \in N_B(v_c)$, $v_i \neq v_0$. All the vertices in V_{rest} are covered by a facility in $T_{v_0}(v_c)$, and therefore, are candidates for the pruning step.



■ **Figure 6** All vertices in V_{far} are inside $T_{v_0}(v_c)$. V_{rest} are the vertices in $B_{v_1}(v_c)$ and $B_{v_2}(v_c)$ together.

Step 5: Arbitrarily pair the vertices in V_{rest} . Leave out the last vertex if it cannot be paired. For each pair (a_i, b_i) with $w(a_i) \geq w(b_i)$, we compute the value

$$t_i = \frac{w(b_i) \times l(b_i, v_c) - w(a_i) \times l(a_i, v_c)}{w(a_i) - w(b_i)}$$

If t_i is negative then the weighted distance from the center (that covers both a_i and b_i) from a_i is always greater than that from b_i . Therefore we simply prune out b_i , as it cannot affect the value r^* . For the rest of the vertex pairs (a_i, b_i) we define $r_i = w(a_i) \times (l(a_i, v_c) + t_i)$. Let r_m be the median value of all these r_i 's. We do an r -feasibility test on T with $r = r_m$.

Step 6: Consider the case when the feasibility test returns feasible. Then for each pair (a_i, b_i) with $r_i \geq r_m$, we prune out the vertex a_i , since in any optimal solution, the distance of the center to a_i always dominates that to b_i . Similarly, we can prune out vertex b_j from each pair (a_j, b_j) with $r_j < r_m$ when the feasibility test returns infeasible. ◀

► **Time Complexity.** Finding centroid v_c takes $O(n)$ time, where n is the size of T . Finding the $(p - 1, S \cup \{v_c\})$ -center takes $T_{total}(p - 1, n)$ time. Rest of the steps takes $O(n)$ time. Therefore total time takes by the algorithm is

$$T_{prune}(p, n) = \begin{cases} T_{total}(p - 1, n) + O(n) & , p \geq 1 \\ O(n) & , p = 0 \end{cases}$$

► **Lemma 6.** Algorithm 3 does not change the solution to the (p, S) -center in T .

Proof. If B is a type-a big-component then in some optimal solution all vertices in $V(B)$ are covered by S_B . Therefore removing all vertices in $V(B)$ except v_{far} does not affect the (p, S) -center radius.

Again, if B is a type-b big-component then the vertices in V_{rest} are covered by a one new center say x . In Step 6 we prune from each pair only those vertices which can never be the farthest vertex from x . Therefore, deleting these vertices will not affect the position of x in anyway. ◀

27:10 The Weighted k -Center Problem in Trees for Fixed k

► **Lemma 7.** *Algorithm 3 prunes at least $\frac{n}{16k}$ vertices from B .*

Proof. Similar to the analysis in Megiddo [11] Algorithm 3 prunes $\frac{1}{8}$ -fraction of the vertices in B . Since B is a big-component, it has at least $\frac{n}{2k}$ vertices. Therefore, number of vertices pruned is at least $\frac{1}{8} \cdot \frac{n}{2k} = \frac{n}{16k}$. ◀

5 The (p, S) -center Algorithm

In this section, we present a recursive $O(n)$ time algorithm for the (p, S) -center problem in T where $k = p + |S|$ is a constant.

► **Algorithm 4.**

Input: Tree T , integer p and the set of centers S .

Output: An optimal solution X to the (p, S) -center problem in T .

Step 1: Let n_0 be some suitable constant. We set $T' = T$ and if $|T'| \geq n_0$ we do Step 2, otherwise we jump to Step 3.

Step 2: Find a big-component B in T' using Algorithm 2. Next we prune the vertices of B using Algorithm 3. The pruning step leads to a new tree T'' with lesser number of vertices than T' . We set $T' = T''$ and repeat this step if $|T'| \geq n_0$.

Step 3: We compute the (p, S) -center solution X in T' using standard brute force technique. From Lemma 6, X is also the solution to the (p, S) -center in the original tree T . ◀

► **Time Complexity.** The time taken by the Algorithm 4 is given by the following recursion

$$T_{total}(p, n) = \begin{cases} T_{big}(p, n) + T_{prune}(p, n) + T_{total}(p, n - \frac{n}{16k}) + O(n) & , n \geq n_0 \\ O(1) & , \text{otherwise} \end{cases}$$

Now,

$$\begin{aligned} T_{total}(p, n) &= T_{total}(p, n - \frac{n}{16k}) + T_{big}(p, n) + T_{prune}(p, n) + O(n) \\ &\leq T_{total}(p, \frac{16k-1}{16k}n) + T_{total}(p-1, n) + \sum_{j=1}^{k-1} T_{total}(p-1, j\frac{n}{k}) + O(kn) \\ &\leq T_{total}(p, \frac{16k-1}{16k}n) + T_{total}(p-1, n) + (k-1)T_{total}(p-1, n) + O(kn) \\ &\leq T_{total}(p, \frac{16k-1}{16k}n) + k \cdot T_{total}(p-1, n) + O(kn) \end{aligned}$$

To show that $T_{total}(p, n) \leq c2^{pk}n$ (c is a constant such that $O(kn) \leq ckn$) it suffices to show that

$$\begin{aligned} T_{total}(p, \frac{16k-1}{16k}n) + k \cdot T_{total}(p-1, n) + O(kn) &\leq c2^{pk}n \\ \iff c2^{pk} \cdot \frac{16k-1}{16k}n + ck2^{(p-1)k}n + ckn &\leq c2^{pk}n \\ \iff -\frac{1}{16k} + k2^{-k} + k2^{-pk} &\leq 0 \end{aligned}$$

By numerical computation we can show this to be true for $k \geq 13$ and $1 \leq p \leq k$. This implies that the (p, S) -center problem can be solved in $O(2^{pk}n)$ time. By similar computation we can show that for $1 \leq k \leq 12$, $T_{total}(p, n) \leq O(n)$. Therefore the weighted k -center problem can be solved in $O(2^{k^2}n)$ time. For fixed k this time is linear in n .

6 Conclusion

The most efficient algorithm for the weighted k -center problem in trees for arbitrary k takes $O(n \log n)$ time and is given by Wang et al. [15]. For $k = 1$ and 2, Megiddo [11] and Ben-Moshe et al. [2] give $O(n)$ time solutions, respectively. It is not known whether a linear time algorithm exists for this problem. We settle this problem partially by presenting a linear time algorithm for any constant k . However, the question of whether an optimal linear time algorithm exists for an arbitrary k still remains open.

For the weighted k -center problem in cactus Ben-Moshe [3] presented an $O(n^2)$ time algorithm. In the same paper, they also showed that for $k = 1$ and 2, the problem can be solved in $O(n \log n)$ and $O(n \log^3 n)$ time respectively. No subquadratic time algorithm is known for the weighted k -center problem for cactus when $k > 2$.

References

- 1 Aritra Banik, Binay Bhattacharya, Sandip Das, Tsunehiko Kameda, and Zhao Song. The p -Center Problem in Tree Networks Revisited. In *15th Scandinavian Symposium and Workshops on Algorithm Theory*, 2016.
- 2 Boaz Ben-Moshe, Binay Bhattacharya, and Qiaosheng Shi. An Optimal Algorithm for the Continuous/Discrete Weighted 2-Center Problem in Trees. In José R. Correa, Alejandro Hevia, and Marcos Kiwi, editors, *LATIN 2006: Theoretical Informatics*, pages 166–177, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
- 3 Boaz Ben-Moshe, Binay Bhattacharya, Qiaosheng Shi, and Arie Tamir. Efficient algorithms for center problems in cactus networks. *Theoretical Computer Science*, 378(3):237–252, 2007. Algorithms and Computation.
- 4 Binay Bhattacharya, Sandip Das, and Tsunehiko Kameda. Linear-time fitting of a k -step function. *Discrete Applied Mathematics*, 2017. doi:10.1016/j.dam.2017.11.005.
- 5 Binay Bhattacharya and Qiaosheng Shi. Optimal Algorithms for the Weighted p -Center Problems on the Real Line for Small p . In Frank Dehne, Jörg-Rüdiger Sack, and Norbert Zeh, editors, *Algorithms and Data Structures*, pages 529–540, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
- 6 Richard Cole. Slowing Down Sorting Networks To Obtain Faster Sorting Algorithm. In *Foundations of Computer Science, 1984. 25th Annual Symposium on*, pages 255–260. IEEE, 1984.
- 7 Greg N. Frederickson. Parametric search and locating supply centers in trees. In Frank Dehne, Jörg-Rüdiger Sack, and Nicola Santoro, editors, *Algorithms and Data Structures*, pages 299–319, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.
- 8 M Jeger and Oded Kariv. Algorithms for finding P -centers on a weighted tree (for relatively small P). *Networks*, 15(3):381–389, 1985.
- 9 Oded Kariv and S Louis Hakimi. An algorithmic approach to network location problems. I: The p -centers. *SIAM Journal on Applied Mathematics*, 37(3):513–538, 1979.
- 10 Arindam Karmakar, Sandip Das, Subhas C Nandy, and Binay K Bhattacharya. Some variations on constrained minimum enclosing circle problem. *Journal of Combinatorial Optimization*, 25(2):176–190, 2013.
- 11 Nimrod Megiddo. Linear-time algorithms for linear programming in \mathbb{R}^3 and related problems. *SIAM journal on computing*, 12(4):759–776, 1983.
- 12 Nimrod Megiddo and Arie Tamir. New results on the complexity of p -centre problems. *SIAM Journal on Computing*, 12(4):751–758, 1983.
- 13 Edward Minieka. Conditional centers and medians of a graph. *Networks*, 10(3):265–272, 1980.
- 14 Qiaosheng Shi. *Efficient algorithms for network center/covering location optimization problems*. PhD thesis, School of Computing Science-Simon Fraser University, 2008.
- 15 Haitao Wang and Jingru Zhang. An $O(n \log n)$ -Time Algorithm for the k -Center Problem in Trees. In Bettina Speckmann and Csaba D. Tóth, editors, *34th International Symposium on Computational Geometry (SoCG 2018)*, volume 99 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 72:1–72:15, Dagstuhl, Germany, 2018.

Online Knapsack Problems with a Resource Buffer

Xin Han

Dalian University of Technology, Dalian, China
hanxin@dlut.edu.cn

Yasushi Kawase

Tokyo Institute of Technology, Tokyo, Japan
kawase.y.ab@m.titech.ac.jp

Kazuhisa Makino

Kyoto University, Kyoto, Japan
makino@kurims.kyoto-u.ac.jp

Haruki Yokomaku

NTT DATA Mathematical Systems, Tokyo, Japan
dsm4up2c@gmail.com

Abstract

In this paper, we introduce online knapsack problems with a resource buffer. In the problems, we are given a knapsack with capacity 1, a buffer with capacity $R \geq 1$, and items that arrive one by one. Each arriving item has to be taken into the buffer or discarded on its arrival irrevocably. When every item has arrived, we transfer a subset of items in the current buffer into the knapsack. Our goal is to maximize the total value of the items in the knapsack. We consider four variants depending on whether items in the buffer are removable (i.e., we can remove items in the buffer) or non-removable, and proportional (i.e., the value of each item is proportional to its size) or general. For the general&non-removable case, we observe that no constant competitive algorithm exists for any $R \geq 1$. For the proportional&non-removable case, we show that a simple greedy algorithm is optimal for every $R \geq 1$. For the general&removable and the proportional&removable cases, we present optimal algorithms for small R and give asymptotically nearly optimal algorithms for general R .

2012 ACM Subject Classification Theory of computation → Online algorithms; Theory of computation → Discrete optimization

Keywords and phrases Online knapsack problem, Resource augmentation, Competitive analysis

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.28

Related Version A full version of the paper is available at [8], <https://arxiv.org/abs/1909.10016>.

Funding *Xin Han*: supported by RGC (HKU716412E) and NSFC (11571060).

Yasushi Kawase: supported by JSPS KAKENHI Grant Number 16K16005.

Kazuhisa Makino: supported by JSPS KAKENHI Grant Number JP24106002, JP25280004, JP26280001, and JST CREST Grant Number JPMJCR1402.

1 Introduction

Online knapsack problem is one of the most fundamental problems in online optimization [16, 18]. In the problem, we are given a knapsack with a fixed capacity, and items with sizes and values, which arrive one by one. Upon arrival, we must decide whether to accept the arrived item into the knapsack, and this decision is irrevocable.

In this paper, we introduce a variant of the online knapsack problem, which we call *online knapsack problems with a resource buffer*. Suppose that we have a buffer with fixed capacity in addition to a knapsack with fixed capacity, and items arrive online. Throughout this paper, we assume that the knapsack capacity is 1, and the buffer capacity is $R (\geq 1)$. In



© Xin Han, Yasushi Kawase, Kazuhisa Makino, and Haruki Yokomaku;
licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 28; pp. 28:1–28:14

Leibniz International Proceedings in Informatics



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

addition, assume that each item e has a size $s(e)$ and a value $v(e)$. When an item e has arrived, we must decide whether to take it into the buffer or not. The total size of the selected items must not exceed the capacity of the buffer R . Further, we cannot change the decisions that we made past, i.e., once an item is rejected, it will never be put into the buffer. We consider two settings: (i) *non-removable*, i.e., we cannot discard items in the buffer, and (ii) *removable*, i.e., we can discard some items in the buffer, and once an item is discarded, it will never be put into the buffer again. After the end of the item sequence, we transfer a subset of items from the buffer into the knapsack. Our goal is to maximize the total value of the items in the knapsack under the capacity constraint. It is worth mentioning that, if $R = 1$, our problem is equivalent to the standard online knapsack problem.

Our model can be regarded as a “partial” resource augmentation model. That is, in the resource augmentation model, the online algorithm can use the buffer for the final result. On the other hand, in our model, the online algorithm uses the buffer only to temporarily store items, and it must use the knapsack to output the final result. Moreover, our model can be viewed as a streaming setting: we process items in a streaming fashion, and we can keep only a small portion of the items in memory at any point.

To make things more clear, let us see an example of the online knapsack problem with a resource buffer. Let $R = 1.5$. Suppose that three items e_1, e_2, e_3 with $(s(e_1), v(e_1)) = (0.9, 4)$, $(s(e_2), v(e_2)) = (0.7, 3)$, $(s(e_3), v(e_3)) = (0.2, 2)$ are given in this order, but we do not know the items in advance. When e_1 has arrived, suppose that we take it into the buffer. Then, for the non-removable case, we need to reject e_2 because we cannot put it together with e_1 . In contrast, for the removable case, we have another option – take e_2 into the buffer by removing e_1 . If $\{e_1, e_3\}$ is selected in the buffer at the end, the resulting value is 4 by transferring $\{e_1\}$ to the knapsack. Note that, in the resource augmentation model, we can obtain a solution with value 6 by selecting $\{e_1, e_3\}$.

Related work

For the non-removable online knapsack problem (i.e., non-removable case with $R = 1$), Marchetti-Spaccamela and Vercellis [19] showed that no constant competitive algorithm exists. Iwama and Taketomi [9] showed that there is no constant competitive algorithm even for the proportional case (i.e., the value of each item is proportional to its size). The problem has also been studied under some restrictions on the input [1, 4, 17, 20].

The removable variant of the online knapsack problem (i.e., removable case with $R = 1$) is introduced by Iwama and Taketomi [9]. They proved that no constant competitive deterministic algorithm exists in general, but presented an optimal $(1 + \sqrt{5})/2$ -competitive algorithm for the proportional case. The competitive ratios can be improved by using randomization [5, 7]. In addition, the problem with removal cost has been studied under the name of the *buyback problem* [2, 3, 6, 11, 12].

An online knapsack problem with resource augmentation is studied by Iwama and Zhang [10]. In their setting, an online algorithm is allowed to use a knapsack with capacity $R \geq 1$, while the offline algorithm has a knapsack with capacity 1. They developed optimal $\max\{1, 1/(R - 1)\}$ -competitive algorithms for the general&removable and proportional&non-removable cases and an optimal $\max\{1, \min\{\frac{1+\sqrt{4R+1}}{2R}, \frac{2}{2R-1}\}\}$ -competitive algorithm for the proportional&removable case. All of their algorithms are based on simple greedy strategies. The competitive ratios except for the general&non-removable cases become exactly 1 when R is a sufficiently large real.

In addition, there exist several papers that apply online algorithms to approximately solve the constrained stable matching problems [13–15].

Our results

We consider four variants depending on whether removable or non-removable, and proportional or general. In this paper, we focus on deterministic algorithms. Our results are summarized in Table 1. To compare our model to the resource augmentation model, we list the competitive ratio for both models in the table. It should be noted that each competitive ratio in our model is at least the corresponding one in the resource augmentation model. Hence, lower bounds for the resource augmentation model are also valid to our model.

For the general&non-removable case, we show that there is no constant competitive algorithm. For the proportional&non-removable case, we show that a simple greedy is optimal and its competitive ratio is $\max\{2, 1/(R - 1)\}$. Interestingly, the competitive ratio is equal to the ratio in resource augmentation model for $1 < R \leq 3/2$. For the general&removable case, we present an optimal algorithm for $1 < R \leq 2$. Furthermore, for large R , we provide an algorithm that is optimal up to a logarithmic factor. The algorithm partitions the input items into groups according to sizes and values, and it applies a greedy strategy for each group that meets a dynamically adjusted threshold. We will see that the competitive ratio is larger than 1 for any R but it converges to 1 as R goes to infinity. For the proportional&removable case, we develop optimal algorithms for $1 \leq R \leq 3/2$. The basic idea of the algorithms is similar to that of the algorithm for $R = 1$ given by Iwama and Taketomi [9]. Our algorithms classify the items into three types – *small*, *medium*, and *large* – and the algorithms carefully treat medium items. We observe that, as R becomes large, we need to handle more patterns to obtain an optimal algorithm. In addition, for large R , we show that the algorithm for the general&removable case is also optimal up to a logarithmic factor.

■ **Table 1** Summary of the competitive ratios for our model and the resource augmentation model.

variants	Our model			Resource augmentation			
	R	lower bound	upper bound	R	lower bound	upper bound	
non-removable	prop.	1 (1, $\frac{3}{2}$] $[\frac{3}{2}, \infty)$	∞ [19] $\frac{1}{R-1}$ [10] 2 (Thm. 3)	– $\frac{1}{R-1}$ (Thm. 4) 2 (Cor. 5)	1 (1, 2] $[2, \infty)$	∞ [19] $\frac{1}{R-1}$ [10] 1 [10]	– $\frac{1}{R-1}$ [10] 1 [10]
	gen.	$[1, \infty)$	∞ [19]	–	$[1, \infty)$	∞ [19]	–
removable	prop.	1 $[1, \frac{1+\sqrt{2}}{2}]$ $[\frac{1+\sqrt{2}}{2}, 2 - \frac{\sqrt{2}}{2}]$ $[2 - \frac{\sqrt{2}}{2}, 17 - 9\sqrt{3}]$ $[17 - 9\sqrt{3}, 2\sqrt{3} - 2]$ $[2\sqrt{3} - 2, \frac{3}{2}]$ $[1, \infty)$	$\frac{1+\sqrt{2}}{2}$ [9] $\frac{1+\sqrt{4R+1}}{2R}$ (Thm. 13) $\sqrt{2}^\dagger$ $\frac{\sqrt{16R+1}-1}{2R}^\dagger$ $\frac{1+\sqrt{3}}{2}^\dagger$ $\frac{2}{R}^\dagger$ $1 + \frac{1}{\lfloor 2R \rfloor + 1}$ (Thm. 17)	$\frac{1+\sqrt{2}}{2}$ [9] $\frac{1+\sqrt{4R+1}}{2R}$ (Thms. 14, 15) $\sqrt{2}$ (Thm. 15) $\frac{\sqrt{16R+1}-1}{2R}^\dagger$ $\frac{1+\sqrt{3}}{2}^\dagger$ $\frac{2}{R}^\dagger$ $1 + O(\frac{\log R}{R})$ (Thm. 9)	1 $[1, \frac{1+\sqrt{2}}{2}]$ $[\frac{1+\sqrt{2}}{2}, \frac{3}{2}]$ $[\frac{3}{2}, \infty)$	$\frac{1+\sqrt{2}}{2}$ [9] $\frac{1+\sqrt{4R+1}}{2R}$ [10] $\frac{2}{2R-1}$ [10] 1 [10]	$\frac{1+\sqrt{2}}{2}$ [9] $\frac{1+\sqrt{4R+1}}{2R}$ [10] $\frac{2}{2R-1}$ [10] 1 [10]
	gen.	1 (1, $\frac{3}{2}$] $[\frac{3}{2}, 2)$ $[1, \infty)$	∞ [19] $\frac{1}{R-1}$ (Thm. 7) 2 (Thm. 8) $1 + \frac{1}{R+1}$ (Thm. 6)	– $\frac{1}{R-1}$ (Thm. 12) 2 (Thm. 12) $1 + O(\frac{\log R}{R})$ (Thm. 9)	1 (1, 2] $[2, \infty)$	∞ [19] $\frac{1}{R-1}$ [10] 1 [10]	– $\frac{1}{R-1}$ [10] 1 [10]

†) The corresponding theorems can be found in the full version [8].

2 Preliminaries

We denote the size and the value of an item e as $s(e)$ and $v(e)$, respectively. We assume that $1 \geq s(e) > 0$ and $v(e) \geq 0$ for any e . For a set of items B , we abuse notation, and let $s(B) = \sum_{e \in B} s(e)$ and $v(B) = \sum_{e \in B} v(e)$.

For an item e , the ratio $v(e)/s(e)$ is called the *density* of e . If all the given items have the same density, we call the problem *proportional*. Without loss of generality, we assume that $v(e) = s(e)$ for the proportional case. We sometimes represent an item e as the pair of its size and value $(s(e), v(e))$. Also, for the proportional case, we sometimes represent an item e as its size $s(e)$.

Let $I = (e_1, \dots, e_n)$ be the input sequence of the online knapsack problem with a resource buffer. For a deterministic online algorithm ALG, let B_i be the set of items in the buffer at the end of the round i . Note that $B_0 = \emptyset$. In the removable setting, they must satisfy $B_i \subseteq B_{i-1} \cup \{e_i\}$ and $s(B_i) \leq R$ ($i = 1, \dots, n$). In the non-removable setting, they additionally satisfy $B_{i-1} \subseteq B_i$ ($i = 1, \dots, n$). Without loss of generality, we assume that the algorithm transfers the optimal subset of items from the buffer into the knapsack since we do not require the online algorithm to run in polynomial time. We denote the outcome value of ALG by $\text{ALG}(I) := \max\{v(B) \mid B \subseteq B_n, s(B) \leq 1\}$ and the offline optimal value $\text{OPT}(I) := \max\{v(B) \mid B \subseteq \{e_1, \dots, e_n\}, s(B) \leq 1\}$. Then, the *competitive ratio* of ALG for I is defined as $\text{OPT}(I)/\text{ALG}(I) (\geq 1)$. In addition, the competitive ratio of a problem is defined as $\inf_{\text{ALG}} \sup_I \text{OPT}(I)/\text{ALG}(I)$, where the infimum is taken over all (deterministic) online algorithms and the supremum is taken over all input sequences.

3 General&Non-removable Case

To make the paper self-contained, we show that the general&non-removable case admits no constant competitive algorithm. To see this, we observe an input sequence given by Iwama and Zhang [10], which was used to prove the corresponding result for the resource augmentation setting.

► **Theorem 1.** *For any $R \geq 1$, there exists no constant competitive algorithm for the general&non-removable online knapsack problem with a buffer.*

Proof. Let ALG be an online algorithm and let $R \geq 1$ and c be positive reals. Consider the input sequence $I := ((1, c^1), (1, c^2), \dots, (1, c^k))$, where $(1, c^k)$ is the first item so that ALG does not take into the buffer. Note that $k \leq \lfloor R \rfloor + 1$ since the buffer size is R . If $k = 1$, ALG is not competitive, since $\text{ALG}(I) = 0$ and $\text{OPT}(I) = c$. If $k > 1$, since $\text{ALG}(I) = c^{k-1}$ and $\text{OPT}(I) = c^k$, the competitive ratio is c , which is unbounded as c goes to infinity. ◀

4 Proportional&Non-removable Case

In this section, we consider the proportional&non-removable case. We show that the competitive ratio is $\max\{\frac{1}{R-1}, 2\}$ for the case.

4.1 Lower bounds

For lower bounds, we consider two cases separately: $1 < R \leq 3/2$ and $R > 3/2$.

► **Theorem 2.** *For all R with $1 < R \leq 3/2$ and all $\epsilon > 0$, the competitive ratio of the proportional&non-removable online knapsack problem with a buffer is at least $1/(R-1) - \epsilon$.*

Proof. Let ϵ' be a positive real such that $\frac{1}{R-1+\epsilon'} \geq \frac{1}{R-1} - \epsilon$ and let ALG be an online algorithm. Consider the following input sequence I :

$$R - 1 + \epsilon', 1.$$

Then, ALG must pick the first item, otherwise ALG is not competitive, since $\text{ALG}(I) = 0$ and $\text{OPT}(I) = R - 1 + \epsilon'$. Recall that ALG cannot discard the item since we consider the non-removable setting. Also, ALG cannot take the second item since the buffer size is strictly smaller than the total size of the first and the second items. Thus, $\text{ALG}(I) = R - 1 + \epsilon'$ and $\text{OPT}(I) = 1$, and hence the competitive ratio is at least $\frac{1}{R-1+\epsilon'} \geq \frac{1}{R-1} - \epsilon$. ◀

It should be noted that the input sequence in the proof of Theorem 2 is the same as the one in [10], which is used to show a lower bound for the resource augmentation model.

► **Theorem 3.** *For all $R > 3/2$ and all $\epsilon > 0$, the competitive ratio of the proportional & non-removable online knapsack problem with a buffer is at least $2 - \epsilon$.*

Proof. Let ϵ' be a positive real such that $\frac{2}{1+2\epsilon'} \geq 2 - \epsilon$ and let ALG be an online algorithm. Consider the following input sequence I :

$$\frac{1}{2} + \epsilon', \frac{1}{2} + \frac{\epsilon'}{2}, \dots, \frac{1}{2} + \frac{\epsilon'}{k}, \frac{1}{2} - \frac{\epsilon'}{k},$$

where the k th item $(1/2 + \epsilon'/k)$ is the first item that ALG does not take it into the buffer. Note that I is uniquely determined by ALG and $k \leq 2R$. Since $\text{ALG}(I) = 1/2 + \epsilon'$ and $\text{OPT}(I) = 1/2 + \epsilon'/k + 1/2 - \epsilon'/k = 1$, the competitive ratio is at least $\frac{1}{1/2+\epsilon'} \geq 2 - \epsilon$. ◀

4.2 Upper bounds

For upper bounds, we consider an algorithm that greedily picks a given item if it is possible. The formal description of the algorithm is given in Algorithm 1. Recall that the resulting outcome of the algorithm is $\max\{s(B) \mid B \subseteq B_n, s(B) \leq 1\}$, where B_n is the items in the buffer at the final round n . We prove that the algorithm is optimal for any $R > 1$.

■ **Algorithm 1** $1/(R-1)$ -competitive algorithm.

```

1  $B_0 \leftarrow \emptyset;$ 
2 for  $i \leftarrow 1, 2, \dots$  do
3   if  $s(B_{i-1} \cup \{e_i\}) \leq R$  then  $B_i \leftarrow B_{i-1} \cup \{e_i\}$  else  $B_i \leftarrow B_{i-1};$ 

```

► **Theorem 4.** *Algorithm 1 is $1/(R-1)$ -competitive for the proportional & non-removable online knapsack problem with a buffer when $1 < R \leq 3/2$.*

Proof. Let ALG be an online algorithm induced by Algorithm 1 and I be an input sequence. Without loss of generality, we can assume $s(I) > R$ since otherwise $\text{ALG}(I) = \text{OPT}(I)$.

Suppose that I does not contain items with size at least $R-1$. Let k be the round such that $\sum_{i=1}^{k-1} s(e_i) < R-1 \leq \sum_{i=1}^k s(e_i)$. Then, we have $s(B_k) = \sum_{i=1}^k s(e_i) = s(e_k) + \sum_{i=1}^{k-1} s(e_i) < (R-1) + (R-1) \leq 1$ by $s(e_k) < R-1$ and $R \leq 3/2$. Therefore, in this case, the competitive ratio is at most $\frac{1}{R-1}$.

Next, suppose that I contains an item with size at least $R-1$. Let e_j be the first item in I such that $s(e_j) \geq R-1$. If $s(B_{j-1}) \geq R-1$, then the competitive ratio is at most $\frac{1}{R-1}$ by the same argument as above. Otherwise (i.e., $s(B_{j-1}) < R-1$), we have $s(B_{j-1} \cup \{e_j\}) \leq R$ and hence $e_j \in B_j \subseteq B_n$, i.e., e_j is selected in B_n .

Thus, $\text{ALG}(I) \geq s(e_j) = R-1$ and the competitive ratio is at most $\frac{1}{R-1}$. ◀

Since $1/(R-1) = 2$ when $R = 3/2$, we obtain the following corollary from Theorem 4.

► **Corollary 5.** *Algorithm 1 is 2-competitive for the proportional&non-removable online knapsack problem with a buffer when $R \geq 3/2$.*

5 General&Removable Case

In this section, we consider the general&removable case. We show that the competitive ratio is $\max\{\frac{1}{R-1}, 2\}$ for $R \leq 2$. In addition, for general R , we prove that the competitive ratio is at most $1 + O(\log R/R)$ and at least $1 + \frac{1}{R+1}$.

5.1 Lower bounds

Here, we give lower bounds of the competitive ratio in this case. We first present a general lower bound $1 + 1/(R+1)$. The proof can be found in the full version [8].

► **Theorem 6.** *For $R \geq 1$, the competitive ratio of the general&removable online knapsack problem with a buffer is at least $1 + \frac{1}{R+1}$.*

Next, we provide the tight lower bound for $R \leq 2$. We separately consider the following two cases: $1 < R \leq 3/2$ and $3/2 \leq R < 2$.

► **Theorem 7.** *For all R with $1 < R \leq 3/2$ and all $\epsilon > 0$, the competitive ratio of the general&removable online knapsack problem with a buffer is at least $1/(R-1) - \epsilon$.*

Proof. Let ALG be an online algorithm. Let $\hat{\epsilon}$ be a positive real such that $1/\hat{\epsilon}$ is an integer and $\min\left\{\frac{1}{(R-1+\hat{\epsilon})(1+\hat{\epsilon})}, \frac{1-\hat{\epsilon}^2}{R-1}\right\} \geq \frac{1}{R-1} - \epsilon$. In addition, let $m := 1/\hat{\epsilon}$ and $n := 1/\hat{\epsilon}^3$.

Suppose that ALG is requested the following sequence of items:

$$(1, 1), (\hat{\epsilon}, \hat{\epsilon}^3), (\hat{\epsilon}, 2\hat{\epsilon}^3), \dots, (\hat{\epsilon}, n\hat{\epsilon}^3),$$

until ALG discards the first item $(1, 1)$. Note that the first item has a large size and a medium density, and the following items have the same small sizes but different densities that slowly increase from small to large. In addition, ALG must take the first item at the beginning (otherwise the competitive ratio becomes infinite). Thus, ALG would keep the first item and the last $\lfloor \frac{R-1}{\hat{\epsilon}} \rfloor$ items in each round.

We have two cases to consider: ALG discards the first item $(1, 1)$ or not.

Case 1: Suppose that ALG discards the first item $(1, 1)$ when the item $(\hat{\epsilon}, i\hat{\epsilon}^3)$ comes. Note that the requested sequence is $I := ((1, 1), (\hat{\epsilon}, \hat{\epsilon}^3), (\hat{\epsilon}, 2\hat{\epsilon}^3), \dots, (\hat{\epsilon}, i\hat{\epsilon}^3))$. Then, we have $\text{ALG}(I) \leq (\lfloor \frac{R-1}{\hat{\epsilon}} \rfloor + 1)i\hat{\epsilon}^3$ (since ALG keeps at most $\lfloor \frac{R-1}{\hat{\epsilon}} \rfloor + 1$ small items at the end) and $\text{OPT}(I) \geq \max\{1, m \cdot (i - m)\hat{\epsilon}^3\}$ (the left term 1 comes from the first item and the right term $m \cdot (i - m)\hat{\epsilon}^3$ comes from the last $m (= 1/\hat{\epsilon})$ items). Hence, the competitive ratio is at least $\frac{\max\{1, m \cdot (i - m)\hat{\epsilon}^3\}}{(\lfloor \frac{R-1}{\hat{\epsilon}} \rfloor + 1)i\hat{\epsilon}^3} \geq \frac{1}{(R-1+\hat{\epsilon})(1+\hat{\epsilon})} \geq \frac{1}{R-1} - \epsilon$.

Case 2: Suppose that ALG does not reject the first item until the end. Then, the competitive ratio is at least $\frac{m \cdot (n-m)\hat{\epsilon}^3}{\lfloor \frac{R-1}{\hat{\epsilon}} \rfloor n\hat{\epsilon}^3} \geq \frac{1}{R-1} \cdot \frac{m(n-m)\hat{\epsilon}^3}{n\hat{\epsilon}^2} = \frac{1-\hat{\epsilon}^2}{R-1} \geq \frac{1}{R-1} - \epsilon$. ◀

► **Theorem 8.** *For all R with $3/2 \leq R < 2$ and all $\epsilon > 0$, the competitive ratio of the general&removable online knapsack problem with a buffer is at least $2 - \epsilon$.*

Proof. Let k be an integer such that $k > \max\{\frac{1}{2-R}, \frac{1}{\epsilon}\}$. Let ALG be an online algorithm.

Consider the item sequence $I := (e_1, \dots, e_k)$ where $(s(e_i), v(e_i)) = (1 - \frac{i}{2k^2}, 1 - \frac{i}{2k})$ for $i = 1, \dots, k$. Then, at the end of the sequence, ALG must keep exactly one item because it must select at least one item (otherwise the competitive ratio is unbounded) and every pair of items exceeds the capacity of the buffer (i.e., $s(e_i) + s(e_j) \geq 2(1 - \frac{k}{2k^2}) = 2 - \frac{1}{k} > R$ for any $i, j \in \{1, \dots, k\}$).

Suppose that $\{e_i\}$ is selected in the buffer at the end of the sequence I . If $i = k$, then the competitive ratio for I is $\frac{\text{OPT}(I)}{\text{ALG}(I)} = \frac{v(e_1)}{v(e_k)} = \frac{1 - \frac{1}{2k}}{1 - \frac{1}{2}} = 2 - \frac{1}{k} > 2 - \epsilon$. Otherwise (i.e., $i < k$), let us consider a sequence $I' := (e_1, \dots, e_k, e_{k+1})$ with $(s(e_{k+1}), v(e_{k+1})) = (\frac{i+1}{2k^2}, 1 - \frac{i}{2k})$. Then, the competitive ratio for I' is at least $\frac{\text{OPT}(I')}{\text{ALG}(I')} = \frac{v(e_{i+1}) + v(e_{k+1})}{v(e_i)} = \frac{(1 - \frac{i+1}{2k}) + (1 - \frac{i}{2k})}{1 - \frac{i}{2k}} = 2 - \frac{1}{2k-i} \geq 2 - \frac{1}{k} > 2 - \epsilon$. \blacktriangleleft

5.2 Upper bounds

Here, we provide an asymptotically nearly optimal algorithm for large R and an optimal algorithm for small R (< 2).

First, we provide a $(1 + O(\log R/R))$ -competitive algorithm for the asymptotic case. Suppose that R is sufficiently large. Let $m := \lfloor (R-3)/2 \rfloor$ and let ϵ (≤ 1) be a positive real such that $\log_{1+\epsilon}(1/\epsilon) = m$. Note that we have $m = \Theta(\frac{1}{\epsilon} \log \frac{1}{\epsilon})$ and $\epsilon = O(\log R/R)$ (see Lemma 18 in Appendix A).

We partition all the items as follows. Let S be the set of items with size at most ϵ . Let M be the set of items not in S and let M^j ($j \in \mathbb{Z}$) be the set of items $e \in M$ with $(1+\epsilon)^j \leq v(e) < (1+\epsilon)^{j+1}$ (note that j is not restricted to be positive). Let us consider Algorithm 2 for the problem. Intuitively, the algorithm selects items in greedy ways for S and each M^j with $\nu_i \leq j \leq \mu_i$. Note that for any $i \geq 1$, we have $\mu_i - \nu_i = 2m$. For each $i \geq 1$, since $s(B_i \cap S) \leq 2 + \epsilon$ and $s(B_i \cap M^j) \leq 1$ for any $\nu_i \leq j \leq \mu_i$, we have $s(B_i) \leq 2m + 2 + \epsilon \leq R$. Thus, the algorithm is applicable.

■ **Algorithm 2** $(1 + O(\log R/R))$ -competitive algorithm.

```

1  $B_0 \leftarrow \emptyset$ ;
2 for  $i \leftarrow 1, 2, \dots$  do
3    $B_i \leftarrow \emptyset$  and  $B'_i \leftarrow (B_{i-1} \cup \{e_i\})$ ;
4   foreach  $e \in B'_i \cap S$  in the non-increasing order of the density do
5      $B_i \leftarrow B_i \cup \{e\}$ ;
6     if  $s(B_i) > 2$  then break;
7   Let  $e_i^* \in \arg \max\{v(e) \mid e \in B'_i\}$ ;
8   Let  $\mu_i \leftarrow \lfloor \log_{1+\epsilon} v(e_i^*) \rfloor$  and  $\nu_i \leftarrow \lfloor \log_{1+\epsilon} \epsilon^2 v(e_i^*) \rfloor$ ; //  $e_i^* \in M^{\mu_i}$ 
9   for  $j \leftarrow \nu_i, \dots, \mu_i$  do
10    foreach  $e \in B'_i \cap M^j$  in the non-decreasing order of the size do
11    if  $s(B_i \cap M^j) + s(e) \leq 1$  then  $B_i \leftarrow B_i \cup \{e\}$ ;
```

► **Theorem 9.** *Algorithm 2 is $(1 + O(\log R/R))$ -competitive for the general removable online knapsack problem with a buffer when R is a sufficiently large real.*

Let $I := (e_1, \dots, e_n)$ be an input sequence, $B_{\text{OPT}} \in \arg \max\{v(X) \mid s(X) \leq 1, X \subseteq \{e_1, \dots, e_n\}\}$ be the offline optimal solution, and $B_{\text{ALG}} \in \arg \max\{v(X) \mid s(X) \leq 1, X \subseteq B_n\}$ be the outcome solution of ALG. We construct another feasible solution B^* from B_n by Algorithm 3. Note that $v(B_{\text{ALG}}) \geq v(B^*)$.

■ **Algorithm 3** Construct a feasible solution.

```

1  $B^* \leftarrow B_n \cap B_{\text{OPT}};$ 
2 for  $k \leftarrow \nu_n, \dots, \mu_n$  do
3   Let  $r_k \leftarrow |(B_{\text{OPT}} \setminus B^*) \cap M^k|;$ 
4   for  $j \leftarrow 1, \dots, r_k$  do
5     Let  $a \leftarrow \arg \min\{s(e) \mid e \in (B_n \setminus B^*) \cap M^k\}$  and  $B^* \leftarrow B^* \cup \{a\};$ 
6 while  $(B_n \setminus B^*) \cap S \neq \emptyset$  do
7   Let  $a \in \arg \max\{v(e)/s(e) \mid e \in (B_n \setminus B^*) \cap S\};$ 
8   if  $s(B^*) + s(a) \leq 1$  then  $B^* \leftarrow B^* \cup \{a\};$ 
9   else break;
10 return  $B^*;$ 

```

To prove the theorem, we show the following two claims.

▷ **Claim 10.** $v(B_{\text{OPT}} \cap M) \leq (1 + \epsilon)v(B^* \cap M) + \epsilon v(B_{\text{OPT}})$ and $s(B_{\text{OPT}} \cap M) \geq s(B^* \cap M)$.

▷ **Claim 11.** $v(B_{\text{OPT}} \cap S) \leq v(B^* \cap S) + \epsilon(1 + 2\epsilon)v(B_{\text{OPT}})$.

With these claims, B^* is feasible, and we have $v(B_{\text{OPT}}) = v(B_{\text{OPT}} \cap M) + v(B_{\text{OPT}} \cap S) \leq (1 + \epsilon)v(B^*) + (2\epsilon + 2\epsilon^2)v(B_{\text{OPT}})$. This implies $(1 - 2\epsilon - 2\epsilon^2)v(B_{\text{OPT}}) \leq (1 + \epsilon)v(B^*)$. Since $v(B^*) \leq v(B_{\text{ALG}})$, the competitive ratio of Algorithm 2 is at most $\frac{1+\epsilon}{1-2\epsilon-2\epsilon^2} \leq \frac{1+\epsilon}{1-3\epsilon} \leq 1 + 6\epsilon = 1 + O(\log R/R)$, when $\epsilon < 1/12$ (this inequality follows from the assumption that R is sufficiently large).

The proof is completed by proving Claims 10 and 11.

Proof of Claim 10. Note that $v(B_{\text{OPT}} \cap M) = \sum_{k < \nu_n} v(B_{\text{OPT}} \cap M^k) + \sum_{k \geq \nu_n} v(B_{\text{OPT}} \cap M^k)$. For $e \in M^k$ with $k < \nu_n$, we have $s(e) > \epsilon$ and $v(e) < (1 + \epsilon)^{\nu_n} \leq \epsilon^2 v(e_n^*)$, and hence $v(e)/s(e) \leq \epsilon^2 v(e_n^*)/\epsilon \leq \epsilon v(B_{\text{OPT}})$. Thus, we have $\sum_{k < \nu_n} v(B_{\text{OPT}} \cap M^k) \leq \epsilon v(B_{\text{OPT}})$. For k with $\mu_n \leq k \leq \nu_n$, the set $B_n \cap M^k$ is the greedy solution for M^k according to the non-decreasing order of their size. Hence, by the construction of B^* , the number of items in $B_{\text{OPT}} \cap M^k$ equals to the number of items in $B^* \cap M^k$, and we have $s(B_{\text{OPT}} \cap M^k) \geq s(B^* \cap M^k)$. Also, for each $e \in B_{\text{OPT}} \cap M^k$ and $f \in B^* \cap M^k$, $v(e)/v(f) < (1 + \epsilon)^{k+1}/(1 + \epsilon)^k = (1 + \epsilon)$. Hence, $\sum_{k \geq \nu_n} v(B_{\text{OPT}} \cap M^k) \leq (1 + \epsilon) \sum_{k \geq \nu_n} v(B^* \cap M^k)$. \triangleleft

Proof of Claim 11. It is sufficient to consider the case $B_{\text{OPT}} \cap S \not\subseteq B_n$, since otherwise $B_{\text{OPT}} \cap S \subseteq B^* \cap S$ and the claim clearly holds. Hence, we have $s(B_n \cap S) > 2$. Let $B_n \cap S = \{f_1, f_2, \dots, f_{|B_n \cap S|}\}$ be sorted in non-increasing order of their density. Let f_j be the item with the largest index in $(B_n \cap S) \setminus B_{\text{OPT}}$. Also let $\ell \geq 1$ be the index such that $\sum_{i=1}^{\ell} s(f_i) \leq 1 < \sum_{i=1}^{\ell+1} s(f_i)$. There are two cases to consider: $j \leq \ell$ and $j > \ell$.

Case 1: Suppose that $j \leq \ell$. Then, by the definition of f_j , we have $\{f_{\ell+1}, \dots, f_{|B_n \cap S|}\} \subseteq B_{\text{OPT}}$. Since $s(B_n \cap S) > 2$, we have $s(B_{\text{OPT}}) \geq s(B_{\text{OPT}} \cap S) \geq s(B_n \cap S) - \sum_{i=1}^{\ell} s(f_i) > 1$, which contradicts with $s(B_{\text{OPT}}) \leq 1$.

Case 2: Suppose that $j > \ell$. In this case, we prove that $v(f_j) \leq \epsilon(1 + 2\epsilon)v(B_{\text{OPT}})$.

Since $\sum_{i=1}^{\ell} s(f_i) \geq 1 - \epsilon$, we have $(1 - \epsilon) \cdot \frac{v(f_j)}{s(f_j)} \leq \sum_{i=1}^{\ell} s(f_i) \cdot \frac{v(f_j)}{s(f_j)} \leq \sum_{i=1}^{\ell} s(f_i) \cdot \frac{v(f_i)}{s(f_i)} = \sum_{i=1}^{\ell} v(f_i) \leq v(B_{\text{OPT}})$. Therefore, $v(f_j) \leq \frac{s(f_j)}{1 - \epsilon} v(B_{\text{OPT}}) \leq \frac{\epsilon}{1 - \epsilon} v(B_{\text{OPT}}) \leq \epsilon(1 + 2\epsilon)v(B_{\text{OPT}})$ when $\epsilon \leq 1/2$.

Since $s(B^* \cap M) \leq s(B_{\text{OPT}} \cap M)$ by construction of B^* , we have $s(B^* \cap S) + s(f_j) \geq s(B_{\text{OPT}} \cap S)$. By construction of $B_n \cap S$, we have $\min\{v(f)/s(f) \mid f \in (B_n \cap S) \setminus B_{\text{OPT}}\} \geq \max\{v(f)/s(f) \mid f \in (B_{\text{OPT}} \cap S) \setminus B_n\}$. Therefore, $v(B^* \cap S) + v(f_j) \geq v(B_{\text{OPT}} \cap S)$.

Moreover we have $v(f_j) \leq \epsilon(1 + 2\epsilon)v(B_{\text{OPT}})$, and the claim follows. \triangleleft

The proof of Theorem 9 is completed.

Next, let us consider an algorithm that selects items according to the non-increasing order of the density. The algorithm is formally described in Algorithm 4. We prove that it is optimal when $1 < R < 2$.

■ **Algorithm 4** $\max\{1/(R - 1), 2\}$ -competitive algorithm for $1 < R < 2$.

```

1  $B_0 \leftarrow \emptyset$ ;
2 for  $i \leftarrow 1, 2, \dots$  do
3    $B_i \leftarrow \emptyset$ ;
4   foreach  $e \in B_{i-1} \cup \{e_i\}$  in the non-increasing order of the density do
5     if  $s(B_i) + s(e) \leq R$  then  $B_i \leftarrow B_i \cup \{e\}$ ;

```

► **Theorem 12.** *Algorithm 4 is $\max\{1/(R - 1), 2\}$ -competitive for the general&removable online knapsack problem with a buffer when $1 < R < 2$.*

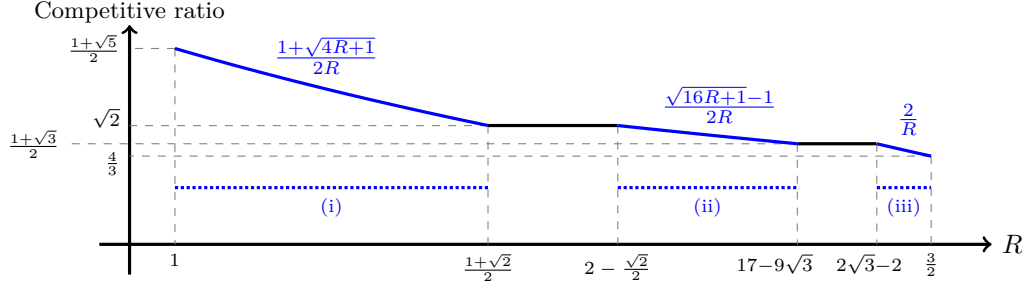
Proof. Let $I := (e_1, \dots, e_n)$ be an input sequence. Without loss of generality, we can assume that $\sum_{i=1}^n s(e_i) > R$ since otherwise $\text{ALG}(I) = \text{OPT}(I)$. Let f_1, \dots, f_n be the rearrangement of I according to the non-increasing order of the density, i.e., $\{f_1, \dots, f_n\} = \{e_1, \dots, e_n\}$ and $v(f_1)/s(f_1) \geq \dots \geq v(f_n)/s(f_n)$. Let k ($\leq n - 1$) be the index such that $\sum_{i=1}^k s(f_i) \leq 1 < \sum_{i=1}^{k+1} s(f_i)$. Then, by the definition of the algorithm, we have $\{f_1, \dots, f_k\} \subseteq B_n$. There are two cases to consider: $f_{k+1} \notin B_n$ and $f_{k+1} \in B_n$.

Case 1: Suppose that $f_{k+1} \notin B_n$. Then, we have $\sum_{i=1}^{k+1} s(f_i) > R$, and hence $\sum_{i=1}^k s(f_i) > R - s(f_{k+1}) \geq R - 1$ by $s(f_{k+1}) \leq 1$. Thus, $\text{OPT}(I)$ is at most $\text{ALG}(I)/(R - 1)$ and the competitive ratio is at most $1/(R - 1)$.

Case 2: Suppose that $f_{k+1} \in B_n$. By a similar analysis of the famous 2-approximation algorithm for the offline knapsack problem, we have $\text{OPT}(I) \leq \sum_{i=1}^k v(f_i) + v(f_{k+1}) \leq 2 \cdot \max\{\sum_{i=1}^k v(f_i), v(f_{k+1})\} \leq 2 \cdot \text{ALG}(I)$. Thus, the competitive ratio is at most 2. \blacktriangleleft

6 Proportional&Removable Case

In this section, we consider the proportional&removable case. We consider the following four cases separately: (i) $1 \leq R \leq \frac{1+\sqrt{2}}{2}$, (ii) $2 - \frac{\sqrt{2}}{2} \leq R \leq 17 - 9\sqrt{3}$, (iii) $2\sqrt{3} - 2 \leq R \leq 3/2$, and (iv) general R (see Figure 1). We remark that the competitive ratios for $\frac{1+\sqrt{2}}{2} \leq R \leq 2 - \frac{\sqrt{2}}{2}$ (and $17 - 9\sqrt{3} \leq R \leq 2\sqrt{3} - 2$) can be obtained by considering the upper bound for $R = \frac{1+\sqrt{2}}{2}$ in case (i) ($R = 17 - 9\sqrt{3}$ in case (ii)) and the lower bound for $R = 2 - \frac{\sqrt{2}}{2}$ in case (ii) ($R = 2\sqrt{3} - 2$ in case (iii)). Due to space limitation, we only analyze cases (i) and (iv). The analysis for (ii) and (iii) can be found in the full version [8].



■ **Figure 1** The competitive ratios for the proportional&removable case with $1 \leq R \leq \frac{3}{2}$.

6.1 $1 \leq R \leq \frac{1+\sqrt{2}}{2}$

We prove that the competitive ratio is $\frac{1+\sqrt{4R+1}}{2R}$ when $1 \leq R \leq \frac{1+\sqrt{2}}{2}$. Let $r > 0$ be a real such that $r + r^2 = R$, i.e., $r = \frac{\sqrt{1+4R}-1}{2}$.

6.1.1 Lower bound

We first prove the lower bound.

► **Theorem 13.** *For any $\epsilon > 0$, the competitive ratio of the proportional&removable online knapsack problem with a buffer is at least $\frac{1+\sqrt{4R+1}}{2R} - \epsilon$ when $1 \leq R < 2$.*

Proof. Let ALG be an online algorithm and let ϵ' be a positive real such that $\frac{r}{r^2+\epsilon'} \geq \frac{1}{r} - \epsilon$ and $\epsilon' < r - r^2$. Note that $r = \frac{\sqrt{1+4R}-1}{2} < 1$ and $\frac{1}{r} = \frac{1+\sqrt{4R+1}}{2R}$. Consider the input sequence $I := (e_1, e_2)$ where $s(e_1) = r$ and $s(e_2) = r^2 + \epsilon'$. Since $r + r^2 = R$, ALG must discard at least one of them. If ALG discards the item with size r , then the competitive ratio for the sequence is $\frac{r}{r^2+\epsilon'} \geq \frac{1}{r} - \epsilon = \frac{1+\sqrt{4R+1}}{2R} - \epsilon$. If ALG discards the item with size $r^2 + \epsilon'$, let $I' := (e_1, e_2, e_3)$ where $s(e_3) = 1 - r^2 - \epsilon'$. As $r \geq 1 - r^2$ and $r + (1 - r^2 - \epsilon') > 1$, we have $\text{OPT}(I') = 1$ and $\text{ALG}(I') \leq r$. Hence the competitive ratio is at least $\frac{1}{r} = \frac{1+\sqrt{4R+1}}{2R}$. ◀

6.1.2 Upper bound for $1 \leq R \leq 10/9$

Next, we give an optimal algorithm for $1 \leq R \leq 10/9$. In this subsection, an item e is called *small*, *medium*, and *large* if $s(e) \leq r^2$, $r^2 < s(e) < r$, and $r \leq s(e)$, respectively. Let S , M , and L respectively denote the sets of small, medium, and large items.

We consider Algorithm 5, which is a generalization of the $\frac{1+\sqrt{5}}{2}$ -competitive algorithm for $R = 1$ given by Iwama and Taketomi [9]. If the algorithm can select a set of items B' such that $r \leq s(B') \leq 1$, it keeps the set B' until the end since it is sufficient to achieve $1/r$ -competitive. Otherwise, it picks the smallest medium item (if exists) and greedily selects small items according to the non-increasing order of the sizes. We show that it is optimal when $1 \leq R \leq 10/9$.

► **Theorem 14.** *Algorithm 5 is $\frac{1+\sqrt{1+4R}}{2R}$ -competitive for the proportional&removable online knapsack problem with a buffer when $1 \leq R \leq 10/9$.*

Proof. Let $I := (e_1, \dots, e_n)$ be the input sequence. If there exists a large item e_i , the competitive ratio is at most $1/r = \frac{1+\sqrt{1+4R}}{2R}$ by $r \leq s(e_i) \leq 1$. If there exist two medium items e_i, e_j such that $s(e_i) + s(e_j) \leq 1$, the competitive ratio is at most $1/r = \frac{1+\sqrt{1+4R}}{2R}$ by $r < 2r^2 < s(e_i) + s(e_j) \leq 1$. In what follows, we assume that all the input items are

■ **Algorithm 5** $\frac{1+\sqrt{1+4R}}{2R}$ -competitive algorithm for $1 \leq R \leq \frac{10}{9}$.

```

1  $B_0 \leftarrow \emptyset$ ;
2 for  $i \leftarrow 1, 2, \dots$  do
3   if  $\exists B' \subseteq B_{i-1} \cup \{e_i\}$  such that  $r \leq s(B') \leq 1$  then  $B_i \leftarrow B'$ ;
4   else if  $e_i \in M$  and  $|B_{i-1} \cap M| = 1$  then
5     let  $\{e'_i\} = B_{i-1} \cap M$ ;
6     if  $s(e_i) < s(e'_i)$  then  $B_i \leftarrow B_{i-1} \cup \{e_i\} \setminus \{e'_i\}$ ;
7     else  $B_i \leftarrow B_{i-1}$ ;
8   else
9      $B_i \leftarrow \emptyset$ ;
10    foreach  $e \in B_{i-1} \cup \{e_i\}$  in non-increasing order of size do
11      if  $s(B_i) + s(e) \leq R$  then  $B_i \leftarrow B_i \cup \{e\}$ ;

```

not large and every pair of medium items cannot be packed into the knapsack together. In addition, suppose that $s(B_n) \notin [r, 1] > 1$ since otherwise the competitive ratio is at most $1/r = \frac{1+\sqrt{1+4R}}{2R}$. By the algorithm, this additional assumption means $s(B') \notin [1, r]$ for any $B' \subseteq B_{i-1} \cup \{e_i\}$ with $i \in \{1, \dots, n\}$.

If $\{e_1, \dots, e_n\} \cap S \subseteq B_n$, the competitive ratio is at most

$$\frac{r + s(\{e_1, \dots, e_n\} \cap S)}{r^2 + s(\{e_1, \dots, e_n\} \cap S)} \leq \frac{1}{r} = \frac{1 + \sqrt{1 + 4R}}{2R}.$$

Otherwise, i.e., $\{e_1, \dots, e_n\} \cap S \not\subseteq B_n$, let e_i be a small item that is not in B_n , and j be the smallest index such that $j \geq i$ and $e_i \notin B_j$. Note that $e_i \in B_{j-1} \cup \{e_j\}$. We have four cases to consider.

Case 1: Suppose that $s(e_i) \geq r/2$. In this case, there exists $e' \in B_j$ such that $r^2 \geq s(e') \geq s(e_i)$. Thus, we have $r \leq s(e_i) + s(e') \leq 1$, a contradiction.

Case 2: Suppose that there exists no medium item in B_j . Then, there exists $B' \subseteq B_{j-1} \cup \{e_j\}$ such that $r \leq s(B') \leq 1$, because $s(B_{j-1} \cup \{e_j\}) > R$ and all the items in $B_{j-1} \cup \{e_j\}$ are small. This is a contradiction.

Case 3: Suppose that $s(e) < r/2$ for any $e \in B_j \cap S$. Then, we have $r \leq s(B_j) \leq 1$, a contradiction.

Case 4: Let us consider the other case, i.e., $s(e_i) < r/2$, $\exists e \in B_j \cap M$, and $\exists e' \in B_j \cap S$ such that $s(e') \geq r/2$. Then, $s(B_j) - s(e) + s(e_i) \geq R - s(e) \geq R - r^2 = r$. Also, $s(B_j) - s(e') \leq R - s(e') \leq R - r/2 = r^2 + r/2 \leq 1$. By the additional assumption, we have $s(B_j) - s(e) + s(e_i) > 1$ and $s(B_j) - s(e') < r$. Thus, we have $s(B_j) > 1 + s(e) - s(e_i) > 1 + r^2 - r/2 = (1 - r)^2 + 3r/2 \geq r + r/2 \geq r + s(e') > s(B_j)$, which is a contradiction. ◀

6.1.3 Upper bound for $\frac{10}{9} \leq R \leq \frac{1+\sqrt{2}}{2}$

Recall that $r > 0$ is a real such that $r + r^2 = R$, i.e., $r = \frac{\sqrt{1+4R}-1}{2}$. For $\frac{10}{9} \leq R \leq \frac{1+\sqrt{2}}{2}$, we have $2/3 \leq r \leq 1/\sqrt{2}$ and $1 - r \leq r/2 \leq r^2 \leq 1/2 < r < 1$. In this subsection, an item e is called *small*, *medium*, and *large* if $s(e) \leq 1 - r$, $1 - r < s(e) < r$, and $r \leq s(e)$, respectively. Let S , M , and L respectively denote the sets of small, medium, and large items. In addition, M is further partitioned into three subsets M_i ($i = 1, 2, 3$), where M_1 , M_2 , M_3 respectively denote the set of the items e with size $1 - r < s(e) \leq r/2$, $r/2 < s(e) < r^2$, and $r^2 \leq s(e) < r$.

28:12 Online Knapsack Problems with a Resource Buffer

We consider Algorithm 6 for the problem. If the algorithm can select a set of items B' such that $r \leq s(B') \leq 1$, it keeps the set B' until the end. Otherwise, it partitions the buffer into two spaces with size r and r^2 . All the small items are taken into the first space. If the set of medium items is of size at least r^2 , then the smallest its subset B' with size at least r^2 is selected into the first space. If the set of medium items is of size at most r^2 , then all of them are selected into the first space. If there are remaining medium items, the smallest one is kept in the second space if its size is smaller than r^2 . We show that the algorithm is optimal when $\frac{10}{9} \leq R \leq \frac{1+\sqrt{2}}{2}$.

■ **Algorithm 6** $\frac{1+\sqrt{1+4R}}{2R}$ -competitive algorithm for $\frac{10}{9} \leq R \leq \frac{1+\sqrt{2}}{2}$.

```

1  $B_0 \leftarrow \emptyset, B_0^{(1)} \leftarrow \emptyset, B_0^{(2)} \leftarrow \emptyset;$ 
2 for  $i \leftarrow 1, 2, \dots$  do
3   if  $\exists B' \subseteq B_{i-1} \cup \{e_i\}$  such that  $r \leq s(B') \leq 1$  then  $B_i^{(1)} \leftarrow B'$  and  $B_i^{(2)} \leftarrow \emptyset$ ;
4   else if  $s((B_{i-1} \cup \{e_i\}) \cap M) \geq r^2$  then
5     let  $T_i \in \arg \min\{s(B') \mid B' \subseteq (B_{i-1} \cup \{e_i\}) \cap M, s(B') \geq r^2\};$ 
6      $B_i^{(1)} \leftarrow T_i \cup ((B_{i-1} \cup \{e_i\}) \cap S);$ 
7     if  $B_i^{(1)} \neq B_{i-1} \cup \{e_i\}$  then
8       let  $a \in \arg \min\{s(e) \mid e \in B_{i-1} \cup \{e_i\} \setminus B_i^{(1)}\};$ 
9       if  $a \in M_1 \cup M_2$  then  $B_i^{(2)} \leftarrow \{a\};$ 
10  else  $B_i^{(1)} \leftarrow B_{i-1} \cup \{e_i\}$  and  $B_i^{(2)} \leftarrow \emptyset$ ;
11   $B_i \leftarrow B_i^{(1)} \cup B_i^{(2)};$ 

```

► **Theorem 15.** *Algorithm 6 is $\frac{1+\sqrt{1+4R}}{2R}$ -competitive for the proportional removable online knapsack problem with a buffer when $10/9 \leq R \leq \frac{1+\sqrt{2}}{2}$.*

Let $I := (e_1, \dots, e_n)$ be the input sequence and let $I_k := \{e_1, \dots, e_k\}$ be the first k items of I .

► **Lemma 16.** *If $I_n \subseteq M$, then Algorithm 6 is $\frac{1+\sqrt{1+4R}}{2R}$ ($= 1/r$)-competitive when $10/9 \leq R \leq \frac{1+\sqrt{2}}{2}$.*

Proof. The proof can be found in the full version [8]. ◀

Now, we are ready to prove Theorem 15.

Proof of Theorem 15. Let $\text{OPT} \in \arg \max\{s(X) \mid X \subseteq I_n, s(X) \leq 1\}$ and $\text{OPT}_M \in \arg \max\{s(X) \mid X \subseteq I_n \cap M, s(X) \leq 1\}$. Without loss of generality, we can assume that $\sum_{i=1}^n s(e_i) > R$.

If $e_i \in L$ for some i , then $r \leq s(B_n^{(1)}) \leq 1$. Thus, we assume that all the items in the input sequence are not large, i.e., $I_n \cap L = \emptyset$.

Suppose that Algorithm 6 discards some small items, i.e., $I_n \cap S \neq B_n \cap S$. Let j be the round such that $I_{j-1} \cap S = B_{j-1} \cap S$ and $I_j \cap S \neq B_j \cap S$. Let $T_j \in \arg \min\{s(B') \mid B' \subseteq (B_{j-1} \cup \{e_j\}) \cap M, s(B') > r\}$. Since $I_{j-1} \cap S = B_{j-1} \cap S$ and $I_j \cap S \neq B_j \cap S$, we have $s(T_j \cup (I_j \cap S)) > 1$. Since $s(e) < 1 - r$ ($\forall e \in S$), there exists $S' \in I_j \cap S$ such that $r \leq s(T_j \cup S') \leq 1$. Therefore, if $I_n \cap S \neq B_n \cap S$, then $\text{ALG}(I) \geq r$.

Consequently, we assume $I_n \cap L = \emptyset$ and $I_n \cap S \subseteq B_n$. Then, the competitive ratio is at most

$$\frac{s(\text{OPT})}{s(B_n^{(1)})} \leq \frac{s(\text{OPT}_M) + s(I_n \cap S)}{s(B_n^{(1)} \cap M) + s(I_n \cap S)} \leq \frac{s(\text{OPT}_M)}{s(B_n^{(1)} \cap M)},$$

and hence we can assume, without loss of generality, that $I_n \subseteq M$.

Thus, by Lemma 16, the theorem is proved. \blacktriangleleft

6.2 General R

In this subsection, we consider proportional&removable case with general R . By Theorem 9, the upper bound of the competitive ratio is $1 + O(\log R/R)$. Hence, we only give a lower bound of the competitive ratio.

► **Theorem 17.** *For any positive real $\epsilon < 1$, the competitive ratio of the proportional&removable online knapsack problem with a buffer is at least $1 + \frac{1}{\lceil 2R \rceil + 1} - \epsilon$.*

Proof. Let $n := \lceil 2R \rceil + 1$ and let ALG be an online algorithm. Consider the item sequence $I := (e_1, \dots, e_{n-1}, e_n)$ where $s(e_i) = \frac{i}{n} + \frac{\epsilon}{n^2}$ for $i = 1, \dots, n-1$ (we will set $s(e_n)$ later depending on ALG). At the end of $(n-1)$ st round, ALG must discard at least one item because $\sum_{i=1}^{n-1} s(e_i) > \frac{n-1}{2} = \frac{\lceil 2R \rceil}{2} \geq R$. Suppose that ALG discards e_j , and let $s(e_n) = 1 - s(e_j)$. Then, we have $\text{OPT}(I) = s(e_j) + s(e_n) = 1$. We will prove that $\text{ALG}(I)$ is at most $1 - (1 - \epsilon)/n$, which implies that the competitive ratio of ALG is at least $\frac{1}{1 - (1 - \epsilon)/n} \geq 1 + (1 - \epsilon)/n \geq 1 + \frac{1}{\lceil 2R \rceil + 1} - \epsilon$.

Let B^* be the output of ALG, i.e., $s(B^*) = \text{ALG}(I)$. We have two cases to consider: $e_n \notin B^*$ and $e_n \in B^*$.

Case 1: If $e_n \notin B^*$, then we have $s(B^*) = \frac{\sum_{e_i \in B^*} i}{n} + \frac{|B^*| \cdot \epsilon}{n^2}$. We assume $B^* \neq \emptyset$ since otherwise $s(B^*) = 0$. Since $s(B^*) \leq 1$ and $\frac{|B^*| \cdot \epsilon}{n^2} > 0$, we have $\frac{\sum_{i|e_i \in B^*} i}{n} \leq \frac{n-1}{n}$. Hence, we obtain $s(B^*) \leq \frac{n-1}{n} + \frac{n \cdot \epsilon}{n^2} = 1 - \frac{1-\epsilon}{n}$.

Case 2: If $e_n \in B^*$, then we have $s(B^*) = \frac{(n-j) + \sum_{i|e_i \in B^* \setminus \{e_n\}} i}{n} + \frac{(|B^*| - 2) \cdot \epsilon}{n^2}$. We assume $|B^*| \geq 3$ since otherwise $s(B^*) \leq \frac{n-1}{n}$ by $e_j \notin B^*$. Since $s(B^*) \leq 1$ and $\frac{(|B^*| - 2) \cdot \epsilon}{n^2} > 0$, we have $\frac{(n-j) + \sum_{i|e_i \in B^* \setminus \{e_n\}} i}{n} \leq \frac{n-1}{n}$. Hence, we obtain $s(B^*) \leq \frac{n-1}{n} + \frac{n \cdot \epsilon}{n^2} = 1 - \frac{1-\epsilon}{n}$. \blacktriangleleft

References

- 1 Susanne Albers, Arindam Khan, and Leon Ladewig. Improved Online Algorithms for Knapsack and GAP in the Random Order Model. In *Proceedings of APPROX/RANDOM*, pages 22:1–22:23, 2019.
- 2 Badanidiyuru Ashwinkumar and Robert Kleinberg. Randomized Online Algorithms for the Buyback Problem. In *Internet and Network Economics*, pages 529–536, 2009.
- 3 Moshe Babaioff, Jason D. Hartline, and Robert D. Kleinberg. Selling Ad Campaigns: Online Algorithms with Cancellations. In *Proceedings of EC*, 2009.
- 4 Moshe Babaioff, Nicole Immorlica, David Kempe, and Robert Kleinberg. A knapsack secretary problem with applications. In *Proceedings of APPROX/RANDOM*, pages 16–28. Springer, 2007.
- 5 Marek Cygan, Łukasz Jeż, and Jiří Sgall. Online Knapsack Revisited. *Theory of Computing Systems*, 58(1):153–190, 2016.
- 6 Xin Han, Yasushi Kawase, and Kazuhisa Makino. Online Unweighted Knapsack Problem with Removal Cost. *Algorithmica*, 70(1):76–91, 2014.

- 7 Xin Han, Yasushi Kawase, and Kazuhisa Makino. Randomized algorithms for online knapsack problems. *Theoretical Computer Science*, 562:395–405, 2015.
- 8 Xin Han, Yasushi Kawase, Kazuhisa Makino, and Haruki Yokomaku. Online Knapsack Problems with a Resource Buffer. *CoRR*, abs/1909.10016, 2019. [arXiv:1909.10016](#).
- 9 Kazuo Iwama and Shiro Taketomi. Removable Online Knapsack Problems. In *Proceeding of ICALP*, pages 293–305, 2002.
- 10 Kazuo Iwama and Guochuan Zhang. Optimal Resource Augmentations for Online Knapsack. In *Proceedings of APPROX/RANDOM*, pages 180–188, 2007.
- 11 Yasushi Kawase, Xin Han, and Kazuhisa Makino. Proportional cost buyback problem with weight bounds. *Theoretical Computer Science*, 2016.
- 12 Yasushi Kawase, Xin Han, and Kazuhisa Makino. Unit Cost Buyback Problem. *Theory of Computing Systems*, 2018.
- 13 Yasushi Kawase and Atsushi Iwasaki. Near-Feasible Stable Matchings with Budget Constraints. In *Proceedings of IJCAI*, pages 242–248, 2017.
- 14 Yasushi Kawase and Atsushi Iwasaki. Approximately Stable Matchings with Budget Constraints. In *Proceedings of AAAI*, pages 242–248, 2018.
- 15 Yasushi Kawase and Atsushi Iwasaki. Approximately Stable Matchings with General Constraints. *CoRR*, abs/1907.04163, 2019. [arXiv:1907.04163](#).
- 16 Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack Problems*. Springer-Verlag Berlin Heidelberg, 2004.
- 17 Anton J. Kleywegt and Jason D. Papastavrou. The dynamic and stochastic knapsack problem. *Operations research*, 46(1):17–35, 1998.
- 18 Dennis Komm. *An Introduction to Online Computation*. Springer, 2016.
- 19 Alberto Marchetti-Spaccamela and Carlo Vercellis. Stochastic on-line knapsack problems. *Mathematical Programming*, 68(1):73–104, 1995.
- 20 Yunhong Zhou, Deeparnab Chakrabarty, and Rajan Lukose. Budget Constrained Bidding in Keyword Auctions and Online Knapsack Problems. In *Proceedings of WWW*, pages 1243–1244, 2008.

A Relationship Among m , ϵ and R in Algorithm 2

Here, we prove some relationships among m , ϵ and R in Algorithm 2.

► **Lemma 18.** *Let $R \geq 3$, $m := \lfloor (R-3)/2 \rfloor$ and let $\epsilon > 0$ be a real such that $\log_{1+\epsilon}(1/\epsilon) = m$. Then, $m = \Theta(\frac{1}{\epsilon} \log \frac{1}{\epsilon})$ and $\epsilon = O(\log R/R)$*

Proof. By the definition of the base of natural logarithm e and the monotonicity of $(1+1/x)^x$, we have $2 \leq (1+1/x)^x \leq e$ for any $x \geq 1$. As $\epsilon \leq 1$, we have

$$2^{\epsilon m} \leq (1+\epsilon)^{\frac{1}{\epsilon} \epsilon m} \leq e^{\epsilon m}.$$

By substituting $m = \log_{1+\epsilon}(1/\epsilon)$, we have $(1+\epsilon)^{\frac{1}{\epsilon} \epsilon m} = 1/\epsilon$. Hence, we get

$$\epsilon m \log 2 \leq \log \frac{1}{\epsilon} \leq \epsilon m. \tag{1}$$

This implies $m = \Theta(\frac{1}{\epsilon} \log \frac{1}{\epsilon})$.

Next, we show that $\epsilon = O(\log R/R)$. By the inequalities (1), we have

$$\epsilon \leq \frac{\log \frac{1}{\epsilon}}{m \log 2} \leq \frac{\log(\frac{1}{\epsilon} \log \frac{1}{\epsilon})}{m \log 2} \leq \frac{\log m}{m \log 2} = \frac{\log \lfloor (R-3)/2 \rfloor}{\lfloor (R-3)/2 \rfloor \log 2} = O\left(\frac{\log R}{R}\right). \quad \blacktriangleleft$$

Local Cliques in ER-Perturbed Random Geometric Graphs

Matthew Kahle

Department of Mathematics, The Ohio State University, USA
mkahle@math.osu.edu

Minghao Tian

Computer Science and Engineering Dept., The Ohio State University, USA
tian.394@osu.edu

Yusu Wang

Computer Science and Engineering Dept., The Ohio State University, USA
yusu@cse.ohio-state.edu

Abstract

We study a random graph model introduced in [20] where one adds Erdős–Rényi (ER) type perturbation to a random geometric graph. More precisely, assume $G_{\mathcal{X}}^*$ is a random geometric graph sampled from a nice measure on a metric space $\mathcal{X} = (X, d)$. An *ER-perturbed random geometric graph* $\widehat{G}(p, q)$ is generated by removing each existing edge from $G_{\mathcal{X}}^*$ with probability p , while inserting each non-existent edge to $G_{\mathcal{X}}^*$ with probability q . We consider a localized version of clique number for $\widehat{G}(p, q)$: Specifically, we study the *edge clique number* for each edge in a graph, defined as the size of the largest clique(s) in the graph containing that edge. We show that the edge clique number presents two fundamentally different types of behaviors in $\widehat{G}(p, q)$, depending on which “type” of randomness it is generated from.

As an application of the above results, we show that by a simple filtering process based on the edge clique number, we can recover the shortest-path metric of the random geometric graph $G_{\mathcal{X}}^*$ within a multiplicative factor of 3 from an ER-perturbed observed graph $\widehat{G}(p, q)$, for a significantly wider range of insertion probability q than what is required in [20].

2012 ACM Subject Classification Mathematics of computing → Random graphs; Theory of computation → Computational geometry

Keywords and phrases random graphs, random geometric graphs, edge clique number, the probabilistic method, metric recovery

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.29

Related Version A full version of the paper is available at <https://arxiv.org/abs/1810.08383>.

1 Introduction

Random graphs are mathematical models which have applications in a wide spectrum of domains. Erdős–Rényi graph $G(n, p)$ is one of the oldest and most-studied models for networks [19], constructed by adding edges between all pairs of n vertices with probability p independently. Many global properties of this model are well-studied by using the probabilistic method [1], such as the clique number and the phase transition behaviors of connected components w.r.t. parameter p .

Another classical type of random graphs is the random geometric graph $G(\mathbb{X}_n; r)$ introduced by Edgar Gilbert in 1961 [10]. This model starts with a set of n points \mathbb{X}_n randomly sampled over a metric space (typically a cube in \mathbb{R}^d) from some probability distribution, and edges are added between all pairs of points within distance r to each other. The Erdős–Rényi random graphs and random geometric graphs exhibit similar behavior for the Poisson degree distribution; however, other properties, such as the clique number and phase transition (w.r.t. to p or to r), could be very different [11, 16, 21, 22].



© Matthew Kahle, Minghao Tian, and Yusu Wang;
licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 29; pp. 29:1–29:22

Leibniz International Proceedings in Informatics



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

This model has many applications in real world where the physical locations of objects involved play an important role [7], for example wireless ad hoc networks [18] and transportation networks [2].

We are interested in mixed models that “combine” both types of randomness together. One way to achieve this is to add Erdős–Rényi type perturbation (percolation) to random geometric graphs. A natural question arises: what are the properties of this type of random graphs? Although these graphs are related to the continuum percolation theory [17], our understanding about them so far is still limited: In previous studies, the underlying spaces are typically plane (called the Gilbert disc model) [3], cubes [6] and tori [13]; the vertices are often chosen as the standard lattices of the space; and the results usually concern the connectivity [4, 23] or diameter [24].

Our work. In this paper, we consider a mixed model of Erdős–Rényi random graphs and random geometric graphs, and study the behavior of a local property called *edge clique number*. More precisely, we use the following *ER-perturbed random geometric graph model* previously introduced in [20]. Suppose there is a compact metric space $\mathcal{X} = (X, d)$ (as feature space) with a probability distribution induced by a “nice” measure μ supported on \mathcal{X} (e.g., the uniform measure supported on an embedded smooth low-dimensional Riemannian manifold). Assume we now randomly sample n points V i.i.d from this measure μ , and build the random geometric graph $G_{\mathcal{X}}^*(r)$, which is the r -neighborhood graph spanned by V (i.e, two points $u, v \in V$ are connected if their distance $d(u, v) \leq r$). Next, we add Erdős–Rényi (ER) type perturbation to $G_{\mathcal{X}}^*(r)$: each edge in $G_{\mathcal{X}}^*(r)$ is deleted with a uniform probability p , while each “short-cut” edge between two unconnected nodes u, v is inserted to $G_{\mathcal{X}}^*(r)$ with a uniform probability q . We denote the resulting generated graph by $\widehat{G}_{\mathcal{X}}^{p,q}(r)$.

Intuitively, one can imagine that a graph is generated first as a proximity graph (captured by the random geometric graph) in some feature space (\mathcal{X} in the above setting). The random insertion / deletion of edges then allows for noise or exceptions. For example, in a social network, nodes could be sampled from some feature space of people, and two people could be connected if they are nearby in the feature space. However, there are always some exceptions – friends could be established by chance even they are very different from each other (“far away”), and two similar (“close”) people (say, close geographically and in tastes) may not develop friendship. The ER-perturbation introduced above by [20] aims to account for such kind of exceptions.

We introduce a local property called the *edge clique number* of a graph G , to provide a more refined view than the global clique number. It is defined for each edge (u, v) in the graph, denoted as $\omega_{u,v}(G)$, as the size of the largest clique containing uv in graph G . Our main result is that $\omega_{u,v}(\widehat{G}_{\mathcal{X}}^{p,q}(r))$ presents two fundamentally different types of behaviors, depending on from which “type” of randomness the edge (u, v) is generated from: A “good” edge from the random geometric graph $G_{\mathcal{X}}^*(r)$ has an edge-clique number similar to edges from a certain random geometric graph; while a “bad” edge (u, v) introduced during the random-insertion process has an edge-clique number similar to edges in some random Erdős–Rényi graph. See Theorems 5, 10, 12, and 14 for the precise statements.

As an application of our theoretical analysis, in Theorem 15, we show that by using a filtering process based on our edge clique number, we can recover the shortest-path metric of the random geometric graph $G_{\mathcal{X}}^*(r)$ within a multiplicative factor of 3, from an ER-perturbed graph $\widehat{G}_{\mathcal{X}}^{p,q}(r)$, for a significantly wider range of insertion probability q than what’s required in [20], although we do need a stronger regularity condition on the measure μ . See more discussion at the end of Section 4.

2 Preliminaries

Suppose we are given a compact geodesic metric space $\mathcal{X} = (X, d)$ [5]¹. We will consider “nice” measures on \mathcal{X} . Specifically,

► **Definition 1** (Doubling measure [12]). *Given a metric space $\mathcal{X} = (X, d)$, let $B_r(x) \subset X$ denotes the closed metric ball $B_r(x) = \{y \in X \mid d(x, y) \leq r\}$. A measure $\mu : X \rightarrow \mathbb{R}$ on \mathcal{X} is said to be doubling if every metric ball (with positive radius) has finite and positive measure and there is a constant $L = L(\mu)$ s.t. for all $x \in X$ and every $r > 0$, we have $\mu(B_{2r}(x)) \leq L \cdot \mu(B_r(x))$. We call L the doubling constant and say μ is an L -doubling measure.*

Intuitively, the doubling measure generalizes a nice measure on the Euclidean space, but still behaves nicely in the sense that the growth of the mass within a metric ball is bounded as the radius of the ball increases. For our theoretical results later, we in fact need a stronger condition on the input measure, which we will specify later in **Assumption-A** at the beginning of Section 3.

ER-perturbed random geometric graph. Following [20], we consider the following random graph model: Given a compact metric space $\mathcal{X} = (X, d)$ and a L -doubling probability measure μ supported on X , let V be a set of n points sampled i.i.d. from μ . We build the r -neighborhood graph $G_{\mathcal{X}}^*(r) = (V, E^*)$ for some parameter $r > 0$ on V ; that is, $E^* = \{(u, v) \mid d(u, v) \leq r, u, v \in V\}$. We call $G_{\mathcal{X}}^*(r)$ a random geometric graph generated from (\mathcal{X}, μ, r) . Now we add the following two types of random perturbations:

p -deletion: For each existing edge $(u, v) \in E^*$, we delete edge (u, v) with probability p .

q -insertion: For each non-existent edge $(u, v) \notin E^*$, we insert edge (u, v) with probability q .

The order of applying the above two types of perturbations doesn't matter since they are applied to two disjoint sets respectively. The final graph $\widehat{G}_{\mathcal{X}}^{p,q}(r) = (V, \widehat{E})$ is called a (p, q) -perturbation of $G_{\mathcal{X}}^*(r)$, or simply an *ER-perturbed random geometric graph*.

We now introduce a local version of the standard clique number:

► **Definition 2** (Edge clique number). *Given a graph $G = (V, E)$, for any edge $(u, v) \in E$, its edge clique number $\omega_{u,v}(G)$ is defined as*

$$\omega_{u,v}(G) = \text{the size of the largest clique(s) in } G \text{ containing } (u, v).$$

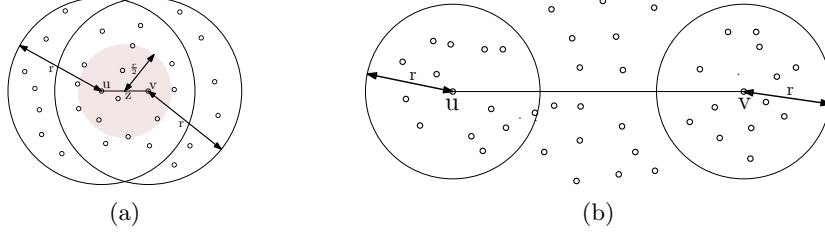
Setup for the remainder of the paper. For convenience of reference, we collect our standard notations. We assume throughout that we are given a fixed compact geodesic metric space $\mathcal{X} = (X, d)$ and a fixed L -doubling probability measure μ . We denote V as the set of n graph nodes sampled i.i.d. from μ . $\widehat{G} = \widehat{G}_{\mathcal{X}}^{p,q}(r) = (V, \widehat{E})$ is a (p, q) -perturbation of a random geometric graph $G^* = G_{\mathcal{X}}^*(r)$ spanned by V with radius parameter r . For an arbitrary graph G , let $V(G)$ and $E(G)$ refer to its vertex set and edge set, respectively, and let $N_G(u)$ denote the set of neighbors of u in G (i.e. nodes connected to $u \in V(G)$ by edges in $E(G)$).

We now define two types of edges in the perturbed graph \widehat{G} . Roughly speaking, we say an edge in \widehat{G} is a good-edge if it is generated by the random geometric graph G^* and later is not removed by (p, q) -perturbation. A bad-edge is typically some long-range edge inserted by the perturbation.

¹ A geodesic metric space is a metric space where any two points in it are connected by a path whose length equals the distance between them. Uniqueness of geodesics is not required. Riemannian manifolds or path-connected compact sets in the Euclidean space are all geodesic metric spaces.

► **Definition 3** (Good / bad-edges). An edge (u, v) in the perturbed graph \widehat{G} is a good-edge if $d(u, v) \leq r$. An edge (u, v) in the perturbed graph \widehat{G} is a bad-edge if for any $x \in N_{G^*}(u)$ and $y \in N_{G^*}(v)$, we have $d(x, y) > r$.

In other word, (u, v) is a bad-edge if and only if there are no edges between neighbors of u and neighbors of v in G^* . See figure 1 for some examples.



■ **Figure 1** (a) shows a good-edge (u, v) . It also shows that if $d(u, v) \leq r$, then there exists an $r/2$ -ball (shaded region) in the intersection of $B_r(u)$ and $B_r(v)$; (b) shows a bad-edge (u, v) .

Organization of paper. In Section 3, we study the behavior of edge clique number for good-edges and bad-edges. Our main result Theorem 14 roughly suggests, under certain conditions on the insertion probability q , for a good-edge (u, v) of $\widehat{G}_{\mathcal{X}}^{p,q}(r)$, with high probability, $\omega_{u,v}(\widehat{G})$ has order $\Omega(\ln \ln n)$; while for a bad-edge (u, v) , its edge-clique number $\omega_{u,v}(\widehat{G})$ has order $o(\ln \ln n)$ with high probability.

To illustrate the main ideas, we will first give results for when only edge-insertion type of perturbations is added to the random geometric graph in Section 3.1 – In fact, this case is of independent interest as well. An application of our result to recover the shortest-path metric of the hidden geometric graph is given in Section 4.

3 Two different behaviors of edge clique number

In Section 3.1, we study the edge clique numbers for the insertion-only perturbed random geometric graphs, both to illustrate the main ideas, and to show the different behaviors of the edge clique number more clearly. In Section 3.2, we study the case for deletion-only perturbed random geometric graphs, where we only delete each edge independently with probability p to obtain an input graph \widehat{G} . Finally, we discuss the combined case of an ER-perturbed random geometric graph in Section 3.3.

First, we need the following technical assumption on the parameter r (for the random geometric graph $G_{\mathcal{X}}^*(r)$) and the measure μ where graph nodes V are sampled from.

[Assumption-A]: The parameter r and the doubling measure μ satisfy the following condition:

There exist $s \geq \frac{13 \ln n}{n}$ ($= \Omega(\frac{\ln n}{n})$) and a constant ρ such that for any $x \in X$

(Density-cond) $\mu(B_{r/2}(x)) \geq s$.

(Regularity-cond) $\mu(B_{r/2}(x)) \leq \rho s$

Intuitively, these two conditions require that for the specific r value we choose, the mass contained inside all radius- r metric balls are similar (within a constant ρ factor). **Density-cond** is equivalent to the Assumption-R in [20]. It requires that r is large enough such that with high probability each vertex v in the random geometric graph $G_{\mathcal{X}}^*(r)$ has degree $\Omega(\ln n)$. Indeed, we have the following claim.

▷ Claim 4 ([20]). Under Density-cond, with probability at least $1 - n^{-5/3}$, each vertex in $G_{\mathcal{X}}^*(r)$ has at least $sn/4$ neighbors.

Proof. For a fixed vertex $v \in V$, let n_v be the number of points in $(V - \{v\}) \cap B_r(v)$. The expectation of n_v is $(n - 1) \cdot \mu(B_r(v)) \geq (n - 1) \cdot \mu(B_{\frac{r}{2}}(v)) \geq s(n - 1)$. By the Chernoff bound, we thus have that

$$\begin{aligned} \mathbb{P}\left[n_v < \frac{sn}{4}\right] &< \mathbb{P}\left[n_v < \frac{s(n-1)}{3}\right] \leq \mathbb{P}\left[n_v < \frac{1}{3}(n-1)\mu(B_r(v))\right] \\ &\leq e^{-\frac{(\frac{2}{3})^2}{2}(n-1)\mu(B_r(v))} \leq n^{-\frac{8}{3}} \end{aligned}$$

It then follows from the union bound that the probability that all n vertices in V have more than $sn/4$ neighbors is at least $1 - n \cdot n^{-\frac{8}{3}} = 1 - n^{-5/3}$. ◁

3.1 Insertion-only perturbation

Recall $G_{\mathcal{X}}^*(r) = (V, E)$ is a random geometric graph whose n vertices V sampled i.i.d. from a L -doubling probability measure μ supported on a compact metric space $\mathcal{X} = (X, d)$. In this section, we assume that the input graph \widehat{G} is generated from $G^* = G_{\mathcal{X}}^*(r)$ as follows: First, include all edges of G^* in \widehat{G} . Next, for any $u, v \in V$ with $(u, v) \notin E(G^*)$, we add edge (u, v) to $E(\widehat{G})$ with probability q . That is, we only insert edges to G^* to obtain \widehat{G} .

First, for good-edges, it is easy to obtain the following result.

▶ **Theorem 5.** *Assume Density-cond holds. Let G^* be an n -node random geometric graph generated from (X, d, μ) as described. Denote $\widehat{G} = \widehat{G}^q$ the final graph after inserting each edge not in G^* independently with probability q . Then, with high probability, for each good-edge (u, v) in \widehat{G} , its edge clique number satisfies that $\omega_{u,v}(\widehat{G}) \geq sn/4$.*

Proof. For each good-edge (u, v) , observe that $B_r(u) \cap B_r(v)$ contains at least one metric ball of radius $r/2$ (say $B_{r/2}(z)$ with z being the mid-point of a geodesic connecting u to v in X , see Figure 1 (a)). And all the points in an $r/2$ -ball span a clique in G^* (r -neighborhood graph). Then by an argument similar to the proof of Claim 4, we have that with probability at least $1 - n^{-\frac{2}{3}}$, the number of points in all of $O(n^2)$ number $r/2$ -balls centered at some mid-point of the geodesics between all pair of nodes $u, v \in V$ is at least $sn/4$. Hence with probability at least $1 - n^{-\frac{2}{3}}$, for all good-edge (u, v) in \widehat{G} , $\omega_{u,v}(\widehat{G}) \geq sn/4$. ◀

Bounding the edge clique number for bad-edges is much more challenging due to the interaction between local edges (from random geometric graph) and long-range edges (from random insertion). To handle this, we will create a finite specific collection of subgraphs for \widehat{G} in an appropriate manner, and bound the edge clique number of a bad-edge in each such subgraph. The property of this specific collection of subgraphs is that the union of these individual cliques provides an upper bound on the edge clique number for this edge in \widehat{G} . To construct this finite collection of subgraphs, we will use the so-called Besicovitch covering lemma which has a lot of applications in measure theory [8]. The finiteness here is crucial for later applying the union bound (i.e., Bonferroni inequality [9]).

First, we introduce some notations. We use a *packing* to refer to a countable collection \mathcal{B} of *pairwise disjoint* closed balls. Such a collection \mathcal{B} is a *packing w.r.t. a set P* if the centers of the balls in \mathcal{B} lie in the set $P \subset X$, and it is a δ -*packing* if all of the balls in \mathcal{B} have radius δ . A set $\{A_1, \dots, A_\ell\}, A_i \subseteq X$, covers P if $P \subseteq \bigcup_i A_i$.

► **Lemma 6** (Besicovitch Covering Lemma, doubling space version, [14]). *Let $\mathcal{X} = (X, d)$ be a doubling space. Then, there exists a constant $\beta = \beta(\mathcal{X}) \in \mathbb{N}$ such that for any $P \subset X$ and $\delta > 0$, there are β different δ -packings w.r.t. P , denoted by $\{\mathcal{B}_1, \dots, \mathcal{B}_\beta\}$, whose union also covers P .*

We call the constant $\beta(\mathcal{X})$ above the *Besicovitch constant*. Note that this constant only depends on the doubling space \mathcal{X} and thus is finite. Given a set A , we say that A is *partitioned into* A_1, A_2, \dots, A_k , if $A = A_1 \cup \dots \cup A_k$ and $A_i \cap A_j = \emptyset$ for any $i \neq j$.

► **Definition 7** (Well-separated clique-partitions family). *Consider the random geometric graph $G^* = G_{\mathcal{X}}^*(r)$. A family $\mathcal{P} = \{P_i\}_{i \in \Lambda}$, where $P_i \subseteq V$ and Λ is the index set of P_i s, forms a well-separated clique-partitions family of G^* if:*

1. $V = \cup_{i \in \Lambda} P_i$.
2. $\forall i \in \Lambda$, P_i can be partitioned as $P_i = C_1^{(i)} \sqcup C_2^{(i)} \sqcup \dots \sqcup C_{m_i}^{(i)}$ where
 - (2-a) $\forall j \in [1, m_i]$, there exist $\bar{v}_j^{(i)} \in V$ such that $C_j^{(i)} \subseteq B_{r/2}(\bar{v}_j^{(i)}) \cap V$.
 - (2-b) For any $j_1, j_2 \in [1, m_i]$ with $j_1 \neq j_2$, $d_H(C_{j_1}^{(i)}, C_{j_2}^{(i)}) > r$, where d_H is the Hausdorff distance between two sets in metric space (X, d) .

We also call $C_1^{(i)} \sqcup C_2^{(i)} \sqcup \dots \sqcup C_{m_i}^{(i)}$ a *clique-partition* of P_i (w.r.t. G^*), and its size (cardinality) is m_i . The size of the well-separated clique-partitions family \mathcal{P} is its cardinality $|\mathcal{P}| = |\Lambda|$.

In the above definition, (2-a) implies that each $C_j^{(i)}$ spans a clique in G^* ; thus we call $C_j^{(i)}$ a *clique* in P_i and $C_1^{(i)} \sqcup C_2^{(i)} \sqcup \dots \sqcup C_{m_i}^{(i)}$ a *clique-partition* of P_i . (2-b) means that there are no edges in G^* between any two cliques of P_i ; thus, any edge in \widehat{G} between such cliques must come from insertion. The following existence lemma can be derived by applying Lemma 6 several times.

► **Lemma 8.** *There is a well-separated clique-partitions family $\mathcal{P} = \{P_i\}_{i \in \Lambda}$ of $G_{\mathcal{X}}^*(r)$ with $|\Lambda| \leq \beta^2$, where $\beta = \beta(\mathcal{X})$ is the Besicovitch constant of \mathcal{X} .*

Proof. To prove the lemma, first we grow an $r/2$ -ball around each node in $V \subset X$ (the vertex set of G^*). By Besicovitch covering lemma (Lemma 6), we have a family of $(r/2)$ -packings w.r.t. V , $\mathcal{B} = \{\mathcal{B}_1, \dots, \mathcal{B}_{\alpha_1}\}$, whose union covers V . Here, the constant α_1 satisfies $\alpha_1 \leq \beta(\mathcal{X})$.

Each \mathcal{B}_i contains a collection of disjoint $r/2$ -balls centered at a subset of nodes in V , and let $V_i \subseteq V$ denote the centers of these balls. For any $u, v \in V_i$, we have $d(u, v) > r$ as otherwise, $B_{r/2}(u) \cap B_{r/2}(v) \neq \emptyset$ meaning that the $r/2$ -balls in \mathcal{B}_i are not all pairwise disjoint. Now consider the collection of r -balls centered at all nodes in V_i . Applying Besicovitch covering lemma to V_i again with $\delta = r$, we now obtain a family of r -packings w.r.t. V_i , denoted by $\mathcal{D}^{(i)} = \mathcal{D}_1^{(i)} \sqcup \dots \sqcup \mathcal{D}_{\alpha_2^{(i)}}^{(i)}$, whose union covers V_i . Here, the constant $\alpha_2^{(i)}$ satisfies $\alpha_2^{(i)} \leq \beta(\mathcal{X})$ for each $i \in [1, \alpha_1]$.

Now each $\mathcal{D}_j^{(i)}$ contains a set of disjoint r -balls centered at a subset of nodes $V_j^{(i)} \subseteq V_i$ of V_i . First, we claim that $\bigcup_j V_j^{(i)} = V_i$. This is because that \mathcal{B}_i is an $r/2$ -packing which implies that $d(u, v) > r$ for any two nodes $u, v \in V_i$. In other words, the r -ball around any node from V_i contains no other nodes in V_i . As the union of r -balls $\mathcal{D}_1^{(i)} \sqcup \dots \sqcup \mathcal{D}_{\alpha_2^{(i)}}^{(i)}$ covers V_i by construction, it is then necessary that each node V_i has to appear as the center in at least one $\mathcal{D}_j^{(i)}$ (i.e, in $V_j^{(i)}$). Hence $\bigcup_j V_j^{(i)} = V_i$.

Now for each vertex set $V_j^{(i)}$, let $P_j^{(i)} \subseteq V$ denote all points from V contained in the $r/2$ -balls centered at points in $V_j^{(i)}$. As $\cup_j V_j^{(i)} = V_i$, we have that $\cup_j P_j^{(i)} = \cup_{v \in V_i} (B_{r/2}(v) \cap V)$. It then follows that $\cup_{i \in [1, \alpha_1]} (\cup_{j \in [1, \alpha_2^{(i)}]} P_j^{(i)}) = V$ as the union of the family of $r/2$ -packings $\mathcal{B} = \{\mathcal{B}_1, \mathcal{B}_2, \dots, \mathcal{B}_{c_1}\}$ covers all points in V (recall that \mathcal{B}_i is just the set of $r/2$ -balls centered at points in V_i).

Clearly, each $P_j^{(i)}$ adapts a clique-partition: Indeed, for each $V_j^{(i)}$, any two nodes in $V_j^{(i)}$ are at least distance $2r$ apart (as the r -balls centered at nodes in $V_j^{(i)}$ are disjoint), meaning that the $r/2$ -balls around them are more than r (Hausdorff-)distance away. In other words, $\mathcal{P} = \{P_j^{(i)}, i \in [1, \alpha_1], j \in [1, \alpha_2^{(i)}]\}$ forms a well-separated clique-partitions family of G^* . Finally, since $\alpha_1, \alpha_2^{(i)} \leq \beta(\mathcal{X}) = \beta$, the cardinality of \mathcal{P} is thus bounded by β^2 . ◀

We also need the following lemma to upper-bound the number of points in every $r/2$ -ball centered at nodes of G^* .

► **Lemma 9.** *Suppose $G^* = (V, E^*)$ is an n -node random geometric graph sampled from (\mathcal{X}, μ, r) . If Assumption-A holds, then with probability at least $1 - n^{-5}$, for every $v \in V$, the ball $B_{r/2}(v) \cap V$ contains at most $3\rho sn$ points.*

Proof. For a fixed vertex $v \in V$, let $n_{v,r/2}$ be the number of points in $(V - \{v\}) \cap B_{r/2}(v)$. By the definition of random geometric graph, we know that $n_{v,r/2}$ is subject to binomial distribution $Bin(n - 1, \mu(B_{r/2}(v)))$. The expectation of $n_{v,r/2}$ is $(n - 1)\mu(B_{r/2}(v)) \leq \rho sn$. Also note that $(n - 1)\mu(B_{r/2}(v)) \geq (n - 1)s \geq 12 \ln n$. By applying the Chernoff bound, we thus have that

$$\mathbb{P} \left[n_{v,r/2} \geq \frac{5}{2} \rho sn \right] \leq \mathbb{P} \left[n_{v,r/2} \geq \frac{5}{2} (n - 1) \mu(B_{r/2}(v)) \right] \leq e^{-\frac{1}{3} \left(\frac{3}{2}\right) (n-1) \mu(B_{r/2}(v))} \leq n^{-6}$$

Finally, by applying the union bound, we know that with probability at least $1 - n \cdot n^{-6} = 1 - n^{-5}$, $\forall v \in G^*$, there are at most $\frac{5}{2} \rho sn + 1 < 3\rho sn$ points in the geodesic ball $B_{r/2}(v)$. ◀

We now state one of our main theorems, which relates the edge clique number for bad-edges with the insertion probability. To simplify notations, we call a clique containing an edge (u, v) a uv -clique.

► **Theorem 10.** *Assume Assumption-A holds. Let $\widehat{G} = \widehat{G}^q$ denote the graph obtained by inserting each edge not in $G_{\mathcal{X}}^*(r)$ independently with probability q . Then there exist constants $c_1, c_2, c_3 > 0$ which depend on the doubling constant L of μ , the Besicovitch constant $\beta(\mathcal{X})$, and the regularity constant ρ , such that for any $K = K(n)$ with $K \rightarrow \infty$ as $n \rightarrow \infty$, with high probability, $\omega_{u,v}(\widehat{G}) < K$ for any bad-edge (u, v) in \widehat{G} , as long as q satisfies*

$$q \leq \min \left\{ c_1, c_2 \cdot \left(\frac{1}{n} \right)^{c_3/K} \cdot \frac{K}{sn} \right\}. \tag{1}$$

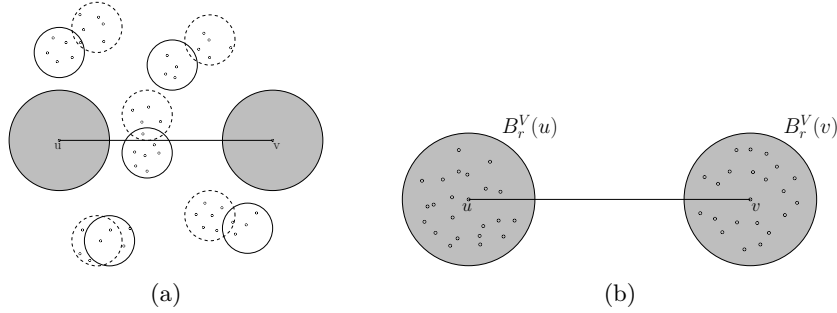
► **Remark.** To illustrate the above theorem, consider for example when $K = \Theta(sn)$. Then the theorem says that there exists constant c' such that if $q < c'$, then w.h.p. $\omega_{u,v} < K$ (thus $\omega_{u,v} = O(sn)$) for any bad-edge (u, v) . Now consider when $q = o(1)$. Then the theorem implies that w.h.p. the edge-clique number for any bad-edge is at most $K = o(sn)$. This is qualitatively different from the edge-clique number for a good-edge for the case $q = o(1)$, which is $\Omega(sn)$ as shown in Theorem 5. By reducing this insertion probability q , this gap can be made **larger and larger**.

Proof of Theorem 10. Given any node y , let $B_r^V(y) \subseteq V$ denote $B_r(y) \cap V$. Now consider a bad-edge (u, v) . Set $A_{uv} = \{w \in V \mid w \notin B_r(u) \cup B_r(v)\}$ and $B_{uv} = \{w \in V \mid w \in B_r^V(u) \cup B_r^V(v)\}$. Denote $\tilde{A}_{uv} = A_{uv} \cup \{u\} \cup \{v\}$; It is easy to check that $V = \tilde{A}_{uv} \cup B_{uv}$.

Let $G|_S$ denote the subgraph of G spanned by a subset S of its vertices. Given any set C , let $C|_S = C \cap S$ be the restriction of C to another set S . Now consider a subset of vertices $C \subseteq V$: obviously, $C = C|_{\tilde{A}_{uv}} \cup C|_{B_{uv}}$. Hence by the pigeonhole principle and the union bound, we have:

$$\begin{aligned} & \mathbb{P} \left[\widehat{G} \text{ has a } uv\text{-clique of size } \geq K \right] \\ & \leq \mathbb{P} \left[\widehat{G}|_{\tilde{A}_{uv}} \text{ has a } uv\text{-clique of size } \geq \frac{K}{2} \right] + \mathbb{P} \left[\widehat{G}|_{B_{uv}} \text{ has a } uv\text{-clique of size } \geq \frac{K}{2} \right] \quad (2) \end{aligned}$$

Next, we will bound the two terms on the right hand side of Eqn. (2) separately in Case (A) and Case (B) below.



■ **Figure 2** (a) A well-separated clique partition $\mathcal{P} = \{P_1, P_2\}$ of A_{uv} – points in the solid ball are P_1 , and those in dashed ball are P_2 . (b) Points in B_{uv} .

Case (A): bounding the first term in Eqn. (2). We apply Lemma 8 for points in A_{uv} . This gives us a well-separated clique-partitions family $\mathcal{P} = \{P_i\}_{i \in \Lambda}$ of A_{uv} with $|\Lambda|$ being a constant (see Figure 2 (a)). Augment each P_i to $\tilde{P}_i = P_i \cup \{u\} \cup \{v\}$. Suppose there is a clique C in $\widehat{G}|_{\tilde{A}_{uv}}$, then as $\bigcup_i \tilde{P}_i = \tilde{A}_{uv}$, we have $C = \bigcup_{i \in \Lambda} C|_{\tilde{P}_i}$, implying that $|C| \leq \sum_{i \in \Lambda} |C|_{\tilde{P}_i}$. Hence by pigeonhole principle and the union bound, we have:

$$\mathbb{P} \left[\widehat{G}|_{\tilde{A}_{uv}} \text{ has } uv\text{-clique of size } \geq \frac{K}{2} \right] \leq \sum_{i=1}^{|\Lambda|} \mathbb{P} \left[\widehat{G}|_{\tilde{P}_i} \text{ has } uv\text{-clique of size } \geq \frac{K}{2|\Lambda|} \right] \quad (3)$$

Now for arbitrary $i \in \Lambda$, consider $\widehat{G}|_{\tilde{P}_i}$, the induced subgraph of \widehat{G} spanned by vertices in \tilde{P}_i . Note, $\widehat{G}|_{\tilde{P}_i}$ can be viewed as generated by inserting each edge not in $G^*|_{\tilde{P}_i} \cup \{uv\}$ to it with probability q . Recall from Definition 7 that each P_i adapts a clique decomposition $C_1^{(i)} \sqcup \dots \sqcup C_{m_i}^{(i)}$, where every $C_j^{(i)}$ is contained in an $r/2$ -ball, and all such balls are r -separated (w.r.t Hausdorff distance).

Fix any $i \in \Lambda$. For simplicity of the argument below, set $m = m_i$, and let $N_j = |C_j^{(i)}|$ denote the number of points in the j -th cluster $C_j^{(i)}$. Note that obviously, $m \leq |P_i| \leq |V| = n$ for any $i \in \Lambda$. Set $N_{\max} = 3\rho sn$. By Lemma 9, we know that, with high probability (at least $1 - n^{-5}$), $N_j \leq N_{\max}$ for all j in $[1, m]$. Let F denote the event that “for every $v \in V$, the ball $B_{r/2}(v) \cap V$ contains at most N_{\max} points”; and F^c denotes the complement event of F .

Now set $k := \lfloor \frac{K}{2|\Lambda|} \rfloor - 2$. For every set S of $k+2$ vertices in this graph $\widehat{G}|_{\widehat{P}_i}$, let A_S be the event “ S is a uv -clique in $\widehat{G}|_{\widehat{P}_i}$ ” and X_S its indicator random variable. Set $X = \sum_{|S|=k+2} X_S$ and note that X is the number of uv -cliques of size $(k+2)$ in $\widehat{G}|_{\widehat{P}_i}$. It follows from Markov inequality that:

$$\begin{aligned} \mathbb{P}\left[\widehat{G}|_{\widehat{P}_i} \text{ has a } uv\text{-clique of size } \geq k+2\right] &= \mathbb{P}[X > 0] \leq \mathbb{P}[X > 0 \mid \mathbb{F}] + \mathbb{P}[\mathbb{F}^c] \\ &\leq \mathbb{E}[X \mid \mathbb{F}] + n^{-5}. \end{aligned} \quad (4)$$

On the other hand, using linearity of expectation, we have:

$$\begin{aligned} \mathbb{E}[X \mid \mathbb{F}] &= \sum_{|S|=k+2} \mathbb{E}[X_S \mid \mathbb{F}] = q^{2k} \sum_{\substack{x_1+x_2+\dots+x_m=k \\ 0 \leq x_i \leq N_i}} \binom{N_1}{x_1} \binom{N_2}{x_2} \dots \binom{N_m}{x_m} q^{(k^2 - \sum_{i=1}^m x_i^2)/2} \\ &\leq q^{2k} \sum_{\substack{x_1+x_2+\dots+x_m=k \\ 0 \leq x_i \leq N_{\max}}} \binom{N_{\max}}{x_1} \binom{N_{\max}}{x_2} \dots \binom{N_{\max}}{x_m} q^{(k^2 - \sum_{i=1}^m x_i^2)/2} \end{aligned} \quad (5)$$

To estimate this quantity, we have the following lemma:

► **Lemma 11.** *There exists a constant $c > 0$ depending on β and ρ such that for any constant $\epsilon > 0$, if $K \leq csn$ and*

$$q \leq \min \left\{ \left(\frac{k!}{n^\epsilon N_{\max}^k m} \right)^{1/2k}, \left(\frac{k!}{k^2 n^\epsilon N_{\max}^k m^2} \right)^{1/k}, \left(\frac{k!}{n^\epsilon m^k N_{\max}^k} \right)^{4/k^2} \right\} \quad (6)$$

then we have that $\mathbb{E}[X \mid \mathbb{F}] = O(n^{-\epsilon})$. Specifically, we can set $\epsilon = 3$ (this choice will be necessary later to apply union bound) and obtain $\mathbb{E}[X \mid \mathbb{F}] = O(n^{-3})$.

The proof of this lemma is rather technical, and can be found in Appendix A.1.

Furthermore, $|\Lambda| \leq \beta^2$ (which is a constant) and $m = |P_i| \leq |V| = n$. One can then verify that there exist constants c_2^a and c_3^a (which depend on the doubling constant L of μ , the Besicovitch constant β , and the regularity constant ρ), such that if

$$q \leq c_2^a \cdot \left(\frac{1}{n} \right)^{c_3^a/K} \cdot \frac{K}{sn}, \quad (7)$$

then the conditions in Eqn. (6) will hold (the simple proof of this can be found in Appendix A.2). Thus, by Lemma 11 and Eqn. (4), we know that if $K \leq csn$ and (7) holds, then

$$\forall i \in \Lambda, \mathbb{P}\left[\widehat{G}|_{\widehat{P}_i} \text{ has a } uv\text{-clique of size } \geq k+2\right] = O(n^{-3}). \quad (8)$$

On the other hand, note that

$$\mathbb{P}\left[\widehat{G}|_{\widehat{P}_i} \text{ has a } uv\text{-clique of size } \geq \frac{K}{2|\Lambda|}\right] = \mathbb{P}\left[\widehat{G}|_{\widehat{P}_i} \text{ has a } uv\text{-clique of size } \geq k+2\right]$$

As $|\Lambda|$ is a constant, by Eqn. (3), we obtain that

$$\begin{aligned} \text{If } \forall i \in \Lambda, \mathbb{P}\left[\widehat{G}|_{\widehat{P}_i} \text{ has a } uv\text{-clique of size } \geq \frac{K}{2|\Lambda|}\right] &= O(n^{-3}), \text{ then} \\ \mathbb{P}\left[\widehat{G}|_{\widehat{A}_{uv}} \text{ has a } uv\text{-clique of size } \geq \frac{K}{2}\right] &= O(n^{-3}) \end{aligned} \quad (9)$$

29:10 Local Cliques in ER-Perturbed Random Geometric Graphs

It then follows from Eqn. (8) and (9) that

$$\text{If } K \leq csn \text{ and (7) holds, then } \mathbb{P} \left[\widehat{G}|_{\tilde{A}_{uv}} \text{ has a } uv\text{-clique of size } \geq \frac{K}{2} \right] = O(n^{-3}). \quad (10)$$

Set $c_1^a = c \cdot c_2^a \cdot \left(\frac{1}{n}\right)^{c_3^a/(c \ln n)}$. Easy to see that:

$$\text{If } q \leq \min \left\{ c_1^a, c_2^a \cdot \left(\frac{1}{n}\right)^{c_3^a/K} \cdot \frac{K}{sn} \right\}, \text{ then}$$

$$\mathbb{P} \left[\widehat{G}|_{\tilde{A}_{uv}} \text{ has a } uv\text{-clique of size } \geq \frac{K}{2} \right] = O(n^{-3}). \quad (11)$$

Case (B): bounding the second term in Eqn. (2). Recall that $B_{uv} = \{w \in V \mid w \in B_r^V(u) \cup B_r^V(v)\}$ (see Figure 2 (b)). Imagine we now build the following random graph $\tilde{G}_{uv}^{local} = (\tilde{V}, \tilde{E})$: The vertex set \tilde{V} is simply B_{uv} . To construct the edge set \tilde{E} , first, add all edges in the clique spanned by nodes in $B_r^V(u)$ as well as edges in the clique spanned by nodes in $B_r^V(v)$ into \tilde{E} . Next, add edge uv to \tilde{E} . Finally, insert each crossing edge xy with $x \in B_r^V(u)$ and $y \in B_r^V(v)$ with probability q .

On the other hand, consider the graph $\widehat{G}|_{B_{uv}}$, the induced subgraph of \widehat{G} spanned by vertices in B_{uv} . We can imagine that the graph $\widehat{G}|_{B_{uv}}$ was produced by first taking the induced subgraph $G^*|_{B_{uv}}$, and then insert crossing edges xy each with probability q . Since uv is a bad-edge, by Definition 3, we know that there are no edges between nodes in $B_r^V(u)$ and $B_r^V(v)$ in the random geometric graph G^* . Hence we obtain:

$$\mathbb{P} \left[\widehat{G}|_{B_{uv}} \text{ has a } uv\text{-clique of size } \geq \frac{K}{2} \right] \leq \mathbb{P} \left[\tilde{G}_{uv}^{local} \text{ has a } uv\text{-clique of size } \geq \frac{K}{2} \right] \quad (12)$$

Using a similar argument as in case (A) (the missing details can be found in Appendix A.3), we have that there exist constants $c_1^b, c_2^b, c_3^b > 0$ which depend on the doubling constant L , the Besicovitch constant β and the regularity constant ρ such that

$$\text{If } q \leq \min \left\{ c_1^b, c_2^b \cdot \left(\frac{1}{n}\right)^{c_3^b/K} \cdot \frac{K}{sn} \right\}, \text{ then}$$

$$\mathbb{P} \left[\tilde{G}_{uv}^{local} \text{ has a } uv\text{-clique of size } \geq \frac{K}{2} \right] = O(n^{-3})$$

Combining this with Eqn. (12), (11) and (2), we know that there exist constants $c_1 = \min\{c_1^a, c_1^b\}$, $c_2 = \min\{c_2^a, c_2^b\}$ and $c_3 = \max\{c_3^a, c_3^b\}$ such that if q satisfies conditions in Eqn. (1), then

$$\mathbb{P} \left[\widehat{G} \text{ has a } uv\text{-clique of size } \geq K \right] = O(n^{-3})$$

Finally, by applying the union bound, this means:

$$\mathbb{P} \left[\text{for every bad-edge } (u, v), \widehat{G} \text{ has a } uv\text{-clique of size } \geq K \right] = O(n^{-1})$$

Thus with high probability, we have that for every bad-edge (u, v) , $\omega_{u,v}(\widehat{G}) < K$ as long as Eqn. (1) holds. \blacktriangleleft

3.2 Edge clique numbers for the deletion-only case

We now consider the deletion-only case, where we assume that the input graph $\widehat{G} = \widehat{G}^p$ is obtained by deleting each edge in the random geometric graph $G^* = G_{\mathcal{X}}^*(r)$ independently with probability p . For an edge (u, v) in \widehat{G} , below we will give a lower bound on the edge-clique number $\omega_{u,v}(\widehat{G})$. A simple observation is that for any edge (u, v) in G^* , as $d_X(u, v) \leq r$, we have that $B_r(u) \cap B_r(v)$ must contain a metric ball of radius $r/2$ (say $B_{r/2}(z)$) centered at midpoint z of a geodesic connecting u to v in X ; see Figure 1 (a)). Thus by a similar argument as the proof of Claim 4, the number of points in the $r/2$ -ball can be bounded from below w.h.p. Note that all points in a $r/2$ -ball span a clique in the random geometric graph G^* . Since we then remove each edge from G^* independently (to obtain \widehat{G}), to find a lower bound for $\omega_{u,v}(\widehat{G})$, it suffices to consider the “local” subgraph of \widehat{G} restricted within this $r/2$ -ball $B_{r/2}(z)$. This local graph has the same behavior as the standard Erdős–Rényi random graph $G(N_z, 1 - p)$, where N_z is the number of points from V within the ball $B_{r/2}(z)$. This eventually leads to the following result.

► **Theorem 12.** *Assume Density-cond holds. Let $\widehat{G} = \widehat{G}^p$ denote the final graph after deleting each edge in $G_{\mathcal{X}}^*(r)$ independently with probability p . Then, for any constant $p \in (0, 1)$, with high probability, we have $\omega_{u,v}(\widehat{G}) \geq \frac{2}{3} \log_{1/(1-p)} sn$ for all edges (u, v) in \widehat{G} .*

On the high level, we first prove the following technical lemma for an Erdős–Rényi random graph, via an application of Janson’s Inequality [1]. The detailed proof can be found in the full version of this paper [15] (see Appendix C in [15]).

► **Lemma 13.** *Suppose $G = G(N, \bar{p})$ is an Erdős–Rényi random graph with*

$$\bar{p} \in \left(\left(\frac{1}{N} \right)^{\frac{1}{10}}, \left(\frac{1}{N} \right)^{\frac{1}{64N}} \right).$$

Set $k := \lfloor \log_{1/\bar{p}} N \rfloor$. Then, we have

$$\mathbb{P}[\omega(G) < k] < e^{-N^{3/2}}$$

where $\omega(G)$ is the clique number of graph G .

► **Remark.** One can easily verify that $(\frac{1}{N})^{\frac{1}{10}}$ is very close to 0 and $(\frac{1}{N})^{\frac{1}{64N}}$ is very close to 1 as N goes to infinity. Hence the range $\bar{p} \in \left((\frac{1}{N})^{\frac{1}{10}}, (\frac{1}{N})^{\frac{1}{64N}} \right)$ is broader (significantly more relaxed) than requiring that \bar{p} is a constant between $(0, 1)$. Hence, while not pursued in the present paper, it is possible to show that Theorem 12 holds for a larger range of p .

Now we are ready to prove the main result in this section (Theorem 12).

Proof of Theorem 12. Using the argument in the proof of Claim 4, we know that for a fixed good-edge (u, v) (i.e. $d(u, v) \leq r$), with probability $1 - n^{-\frac{8}{3}}$, the geodesic ball $B_{r/2}(z)$ (z is the mid-point of a geodesic connecting u to v in X) contains at least $(sn/4)$ points. Note that all points in a $r/2$ -ball form a clique in r -neighborhood graph. Since we remove each edge independently, in order to estimate $\omega_{u,v}(\widehat{G})$ from below, it suffices to consider the “local” graph spanned by nodes in this $r/2$ -ball. Note that this “local” graph have the same behavior as the standard Erdős–Rényi random graph $G_{uv}^{loc} := G(N_z, 1 - p)$, where N_z denotes the number of points falling in the ball $B_{r/2}(z)$.

29:12 Local Cliques in ER-Perturbed Random Geometric Graphs

Furthermore, it is easy to see that, if $N_z \geq sn/4$, then for any constant $p \in (0, 1)$, one can always find a sufficiently large n such that $1 - p \in \left(\left(\frac{1}{N_z}\right)^{\frac{1}{10}}, \left(\frac{1}{N_z}\right)^{\frac{1}{6\sqrt{N_z}}} \right)$.

Now, we are ready to apply Lemma 13 to those “local” graphs: Note that \bar{p} in Lemma 13 will be set to be $1 - p$, and N will be set to be N_z .

$$\mathbb{P} \left[\omega(G_{uv}^{loc}) < k \mid N_z \geq \frac{sn}{4} \right] < e^{-\left(\frac{sn}{4}\right)^{3/2}} \leq e^{-(3 \ln n)^{3/2}} < n^{-(\ln n)^{1/2}}$$

where $k = \left\lfloor \log_{1/(1-p)} N_z \right\rfloor$.

Hence for $k' = \frac{2}{3} \log_{1/(1-p)} sn$, we have that

$$\mathbb{P} \left[\omega(G_{uv}^{loc}) < k' \mid N_z \geq \frac{sn}{4} \right] < \mathbb{P} \left[\omega(G_{uv}^{loc}) < k \mid N_z \geq \frac{sn}{4} \right] < n^{-(\ln n)^{1/2}}.$$

By the law of total probability, we know that

$$\begin{aligned} \mathbb{P} \left[\omega(G_{uv}^{loc}) < k' \mid d(u, v) \leq r \right] &< \mathbb{P} \left[\omega(G_{uv}^{loc}) < k' \mid N_z \geq \frac{sn}{4} \right] + \mathbb{P} \left[N_z < \frac{sn}{4} \right] \\ &< n^{-(\ln n)^{1/2}} + n^{-\frac{8}{3}} \end{aligned}$$

Applying the union bound, we have

$$\begin{aligned} \mathbb{P} \left[\bigwedge_{u, v \in V; d(u, v) \leq r} \omega(G_{uv}^{loc}) \geq k' \right] &= 1 - \mathbb{P} \left[\exists u, v \in V \text{ with } d(u, v) \leq r \text{ s.t. } \omega(G_{uv}^{loc}) < k' \right] \\ &\geq 1 - \frac{1}{2} n^2 \mathbb{P} \left[\omega(G_{uv}^{loc}) < k' \mid d(u, v) \leq r \right] \\ &\geq 1 - \frac{1}{2} n^{-(\ln n)^{1/2} + 2} - \frac{1}{2} n^{-\frac{2}{3}} \end{aligned}$$

Thus, with high probability, for each good-edge (u, v) , we have

$$\omega_{u, v}(\hat{G}) \geq k' = \frac{2}{3} \log_{1/(1-p)} sn. \quad \blacktriangleleft$$

3.3 Combined Case

In this section, we consider both the deletion and insertion. In other words, we consider the ER-perturbed random geometric graph \hat{G} generated via the model described in section 2 that includes both edge-deletion probability p and edge-insertion probability q . Our main results for the combined case are summarized in the following theorem. The (somewhat repetitive) details can be found in the full version of this paper [15] (see Appendix D in [15]).

► **Theorem 14.** *Assume Assumption-A holds. Let $\hat{G} = \hat{G}^{p, q}(r)$ denote the graph obtained by removing each edge in G^* ($= G_{\mathcal{X}}^*(r)$) independently with probability $p \in (0, 1)$ and inserting each edge not in G^* independently with probability q . There exist constants $c_1, c_2, c_3 > 0$ which depend on the doubling constant L of μ , the Besicovitch constant $\beta(\mathcal{X})$, and the regularity constant ρ , such that the following holds for any $K = K(n)$ with $K \rightarrow \infty$ as $n \rightarrow \infty$*

1. *W.h.p., for all good-edges $(u, v) \in \hat{G}$, $\omega_{u, v}(\hat{G}) \geq \frac{2}{3} \log_{1/(1-p)} sn$.*
2. *W.h.p., for all bad-edges $(u, v) \in \hat{G}$, $\omega_{u, v}(\hat{G}) < K$ as long as the insertion probability q satisfies*

$$q \leq \min \left\{ c_1, \quad c_2 \cdot \left(\frac{1}{n} \right)^{c_3/K} \cdot \frac{K}{sn\sqrt{1-p}} \right\} \quad (13)$$

► **Remark.** For example, assume $sn = \Theta(\ln n)$. Then for a constant deletion probability $p \in (0, 1)$, w.h.p. the edge clique number for any good-edge is **at least** $\Omega\left(\log_{1/(1-p)} sn\right) = \Omega(\ln \ln n)$. For any bad-edge uv , if the insertion probability $q = o\left(\left(\frac{1}{n}\right)^{\frac{c_3}{\ln \ln n}} \frac{\ln \ln n}{\ln n}\right)$, then its edge clique number is **at most** $K = o(\ln \ln n)$ w.h.p.. As q decreases, the gap between the edge clique number for good-edges and bad-edges can be made **larger and larger**.

Compared to the insertion-only case, it may seem that the condition on q is too restrictive (recall that for the insertion only case we only require $q = o(1)$ to have a gap between edge clique number for good-edges and bad-edges). Intuitively, this is because: even for an Erdős–Rényi graph $G(n, q)$ with $q = \left(\frac{1}{n}\right)^{\frac{c_3}{\ln \ln n}} \frac{\ln \ln n}{\ln n}$, its clique number is of order $\Theta(\ln \ln n)$ with high probability². This clique size is already at the same scale as the bound of edge clique number for a good-edge in the deletion-only case. Intuitively, this now gets into a regime where the good/bad-edges potentially have edge cliques of asymptotically similar sizes.

4 Recover the shortest-path metric of $G^*(r)$

In this section, we show an application in recovering the shortest-path metric structure of $G_{\mathcal{X}}^*(r)$ from an input observed graph $\widehat{G}_{\mathcal{X}}^{p,q}(r)$. This problem is previously introduced in [20]. Intuitively, assume that $G^* = G_{\mathcal{X}}^*(r)$ is the true graph of interests (which reflects the metric structure of (X, d)), but the observed graph is a (p, q) -perturbed version $\widehat{G} = \widehat{G}_{\mathcal{X}}^{p,q}(r)$ as described in Section 2. The goal is to recover the shortest-path metric of G^* from its noisy observation \widehat{G} with approximation guarantees. Note that due to the random insertion, two nodes could have significantly shorter path in \widehat{G} than in G^* .

Specifically, given two different metrics defined on the same space (Y, d_1) and (Y, d_2) , we say that $d_1 \leq \alpha \cdot d_2$ if for any two points $y_1, y_2 \in Y$, we have that $d_1(y_1, y_2) \leq \alpha \cdot d_2(y_1, y_2)$. The metric d_1 is an α -approximation of d_2 if $\frac{1}{\alpha} \cdot d_2 \leq d_1 \leq \alpha \cdot d_2$ for $\alpha \geq 1$ and $\alpha = 1$ means that $d_1 = d_2$.

Let d_G denote the shortest-path metric on graph G . It was observed in [20] that, roughly speaking, deletion (with p smaller than a certain constant) does not distort the shortest-path metric of G^* by more than a factor of 2. Insertion however could change shortest-path distances significantly. The authors of [20] then proposed a filtering process to remove some “bad” edges based on the so-called Jaccard index, and showed that after the Jaccard-filtering process, the shortest-path metric of the resulting graph \tilde{G} 2-approximates that of the true graph G^* when the insertion probability q is small.

We follow the same framework as [20], but change the filtering process to be one based on the edge clique number instead. This allows us to recover the shortest-path metric within constant factor for a much larger range of values of the insertion probability q , although we do need the extra Regularity-cond which is not needed in [20]. (Note that it does not seem that the bound of [20] can be improved even with this extra Regularity-cond).

We now introduce our edge-clique based filtering process.

τ -Clique filtering: Given graph \widehat{G} , we construct another graph \tilde{G}_τ on the same vertex set as follows: For each edge $(u, v) \in E(\widehat{G})$, we insert the edge (u, v) into $E(\tilde{G}_\tau)$ if and only if $\omega_{u,v}(\widehat{G}) \geq \tau$. That is, $V(\tilde{G}_\tau) = V(\widehat{G})$ and $E(\tilde{G}_\tau) := \{(u, v) \in E(\widehat{G}) \mid \omega_{u,v}(\widehat{G}) \geq \tau\}$.

The following result can be proved by almost the same argument as that for Theorem 12 of [20] with the help of Theorem 14.

² Indeed, the upper bound can be easily derived by computing the expectation; and Lemma 13 in the Appendix provides the lower bound.

► **Theorem 15.** *Assume Assumption-A holds. Suppose $\widehat{G} = \widehat{G}^{p,q}(r)$ is the graph as in Theorem 14. Let \widetilde{G}_τ denote the resulting graph after τ -Clique filtering. Then there exist constants $c_0, c_1, c_2, c_3 > 0$ which depend on the doubling constant L of μ , the Besicovitch constant $\beta(\mathcal{X})$, and the regularity constant ρ , such that if $p \in (0, c_0)$, $\tau \leq \frac{2}{3} \log_{1/(1-p)} sn$, and*

$$q \leq c_2 \cdot \left(\frac{1}{n}\right)^{c_3/\tau} \cdot \frac{\tau}{sn\sqrt{1-p}} \quad \left(= \min \left\{ c_1, \quad c_2 \cdot \left(\frac{1}{n}\right)^{c_3/\tau} \cdot \frac{\tau}{sn\sqrt{1-p}} \right\} \right),$$

then, with high probability, the shortest-path metric $d_{\widetilde{G}_\tau}$ is a 3-approximation of the shortest-path metric d_{G^} of G^* . However, if the deletion probability $p = 0$, then we have w.h.p. that $d_{\widetilde{G}_\tau}$ is a 3-approximation of d_{G^*} as long as $\tau < \frac{sn}{4}$, and $q \leq \min \left\{ c_1, \quad c_2 \cdot \left(\frac{1}{n}\right)^{c_3/\tau} \cdot \frac{\tau}{sn} \right\}$.*

Proof. A simple application of Theorem 14 (i) and (ii) gives the following two lemmas, respectively.

► **Lemma 16.** *Under the same setting as Theorem 14, if $p \in (0, 1)$ and the filtering parameter τ satisfies $\tau < \frac{2}{3} \log_{1/(1-p)} sn$, then, with high probability, our τ -Clique filtering process will not remove any good-edges.*

► **Lemma 17.** *Under the same setting as Theorem 14, there exist constants $c_1, c_2, c_3 > 0$ such that for constant $p \in (0, 1)$, with high probability, a τ -Clique filtering process deletes all bad-edges, as long as $q \leq \min \left\{ c_1, \quad c_2 \cdot \left(\frac{1}{n}\right)^{c_3/\tau} \cdot \frac{\tau}{sn\sqrt{1-p}} \right\}$.*

Our goal is to show that $\frac{1}{3}d_{\widetilde{G}_\tau} \leq d_{G^*} \leq 3d_{\widetilde{G}_\tau}$. Let \mathcal{E}_1 denote the event where $d_{\widehat{G} \cap G^*} \leq 2d_{G^*}$. By Lemma 17 of [20], event \mathcal{E}_1 happens with probability at least $1 - n^{-\Omega(1)}$.

Let \mathcal{E}_2 denote the event where all edges $\widehat{G} \cap G^*$ are also contained in the edge set of the filtered graph \widetilde{G}_τ ; that is, $\widehat{G} \cap G^* \subseteq \widetilde{G}_\tau$. By Lemma 16, event \mathcal{E}_2 happens with probability at least $1 - n^{-\frac{2}{3}}$ (this bound is derived in the proof of Theorem 12). It then follows that:

$$\text{If both events } \mathcal{E}_1 \text{ and } \mathcal{E}_2 \text{ happen, then } d_{\widetilde{G}_\tau} \leq d_{\widehat{G} \cap G^*} \leq 2d_{G^*} \leq 3d_{G^*}.$$

What remains is to show $d_{G^*} \leq 3d_{\widetilde{G}_\tau}$. To this end, we define \mathcal{E}_3 to be the event where for all bad-edges (u, v) in \widehat{G} , we have $\omega_{u,v}(\widehat{G}) < \tau$. If \mathcal{E}_3 happens, then it implies that for an arbitrary edge $(u, v) \in E(\widetilde{G}_\tau)$, either $(u, v) \in E(G^*)$ (thus $d_{G^*}(u, v) = 1$) or $d_{G^*}(u, v) \leq 3$ (since there is at least one edge connecting $N_{G^*}(u)$ and $N_{G^*}(v)$). By Lemma 17, event \mathcal{E}_3 happens with probability at least $1 - o(1)$ (the exact bound can be found in the proof of Theorem 14).

By applying the union bound, we know that $\mathcal{E}_1, \mathcal{E}_2$ and \mathcal{E}_3 happen simultaneously with high probability.

Using a similar argument as the proof of Theorem 11 in [20], it then follows that given any $u, v \in V$ connected in \widetilde{G}_τ , we can find a path in G^* of at most $3d_{\widetilde{G}_\tau}(u, v)$ number of edges to connect u and v . Furthermore, event \mathcal{E}_1 implies that if u and v are not connected in \widetilde{G}_τ , then they cannot be connected in G^* either. Putting everything together, we thus obtain $d_{G^*} \leq 3d_{\widetilde{G}_\tau}$. Theorem 15 then follows. ◀

► **Remark.** Consider the insertion-only case (i.e, the deletion probability $p = 0$), which is a case of independent interest. In this case, if we choose $\tau = \ln n$ and assume that $sn > 4\tau$, then w.h.p. we can recover the shortest-path metric within a factor of 3 as long as $q \leq c \frac{\ln n}{sn}$ for some constant $c > 0$. If $sn = \Theta(\ln n)$ (but $sn > 4\tau = 4 \ln n$), then q is only required to be smaller than a constant. If $sn = \ln^a n$ for some $a > 1$, then we require that $q \leq \frac{c}{\ln^{a-1} n}$. In contrast, [20] requires that $q = o(s)$, which is $q = o\left(\frac{\ln^c n}{n}\right)$ if $sn = \ln^a n$ with $a \geq 1$. The gap (ratio) between these two bounds is nearly a factor of n .

For a *constant* deletion probability $p \in (0, c_0)$, our clique filtering process still requires a much larger range of insertion probability q compared to what's required in [20]. For example, assume $sn = \Theta(\ln n)$. Then if we choose the filtering parameter to be $\tau = \sqrt{\ln \ln n}$, then we can recover d_{G^*} approximately as long as the insertion probability $q = o\left(\left(\frac{1}{n}\right)^{\frac{c_3}{\sqrt{\ln \ln n}}} \frac{\sqrt{\ln \ln n}}{\ln n}\right)$. This is still much larger than the q required in [20], which is $q = o(s) = o\left(\frac{\ln n}{n}\right)$. In fact, $\left(\frac{1}{n}\right)^{\frac{c_3}{\sqrt{\ln \ln n}}} \frac{\sqrt{\ln \ln n}}{\ln n}$ is asymptotically larger than $\frac{1}{n^\varepsilon}$ for any $\varepsilon > 0$. However, we do point out that the Jaccard-filtering process in [20] is algorithmically much simpler and faster, and can be done in $O(n^2)$ time, while the clique-filtering requires the computation of edge-clique numbers, which is computationally expensive.

References

- 1 Noga Alon and Joel Spencer. *The Probabilistic Method*. Wiley Publishing, 4th edition, 2016.
- 2 Philippe Blanchard and Dimitri Volchenkov. *Mathematical analysis of urban spatial networks*. Springer Science & Business Media, 2008.
- 3 Béla Bollobás and Oliver Riordan. *Percolation*. Cambridge University Press, 2006.
- 4 Lorna Booth, Jehoshua Bruck, Matthew Cook, and Massimo Franceschetti. Ad hoc wireless networks with noisy links. In *Proceedings of IEEE International Symposium on Information Theory*, pages 386–386. IEEE, 2003.
- 5 Martin R Bridson and André Haefliger. *Metric spaces of non-positive curvature*, volume 319. Springer Science & Business Media, 2011.
- 6 Don Coppersmith, David Gamarnik, and Maxim Sviridenko. *The Diameter of a Long-Range Percolation Graph*, pages 147–159. Birkhäuser Basel, Basel, 2002.
- 7 Carl Dettmann and Orestis Georgiou. Random geometric graphs with general connection functions. *Physical Review E*, 93(3):032313, 2016.
- 8 Herbert Federer. *Geometric measure theory*. Springer, 2014.
- 9 Janos Galambos. Bonferroni inequalities. *The Annals of Probability*, pages 577–581, 1977.
- 10 Edward Gilbert. Random plane networks. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):533–543, 1961.
- 11 Piyush Gupta and Panganamala Kumar. Critical power for asymptotic connectivity in wireless networks. In *Stochastic analysis, control, optimization and applications*, pages 547–566. Springer, 1999.
- 12 Juha Heinonen. *Lectures on analysis on metric spaces*. Springer Science & Business Media, 2012.
- 13 Svante Janson, Róbert Kozma, Miklós Ruszinkó, and Yury Sokolov. Bootstrap percolation on a random graph coupled with a lattice. *Electronic Journal of Combinatorics*, 2016.
- 14 Antti Kaenmaki, Tapio Rajala, and Ville Suomala. Local homogeneity and dimensions of measures. *Annali Della Scuola Normale Superiore Di Pisa-Classe Di Scienze*, 16(4):1315–1351, 2016.
- 15 Matthew Kahle, Minghao Tian, and Yusu Wang. Local cliques in ER-perturbed random geometric graphs. *arXiv preprint*, 2018. [arXiv:1810.08383](https://arxiv.org/abs/1810.08383).
- 16 Colin McDiarmid and Tobias Müller. On the chromatic number of random geometric graphs. *Combinatorica*, 31(4):423–488, 2011.
- 17 Ronald Meester and Rahul Roy. *Continuum percolation*, volume 119. Cambridge University Press, 1996.
- 18 Maziar Nekovee. Worm epidemics in wireless ad hoc networks. *New Journal of Physics*, 9(6):189, 2007.
- 19 Mark Newman. Random graphs as models of networks. *Handbook of Graphs and Networks: From the Genome to the Internet*, pages 35–68, 2002.

- 20 Srinivasan Parthasarathy, David Sivakoff, Minghao Tian, and Yusu Wang. A Quest to Unravel the Metric Structure Behind Perturbed Networks. In *33rd International Symposium on Computational Geometry, SoCG 2017, July 4-7, 2017, Brisbane, Australia*, pages 53:1–53:16, 2017. doi:10.4230/LIPIcs.SocG.2017.53.
- 21 Mathew Penrose. The longest edge of the random minimal spanning tree. *Ann. Appl. Probab.*, 7(2):340–361, May 1997.
- 22 Mathew Penrose. *Random geometric graphs*, volume 5. Oxford University Press, 2003.
- 23 Gareth Peters and Tomoko Matsui. *Theoretical Aspects of Spatial-Temporal Modeling*. Springer, 2015.
- 24 Xian Yuan Wu. Mixing Time of Random Walk on Poisson Geometry Small World. *Internet Mathematics*, 2017.

A The missing proofs in Section 3.1

A.1 The proof of Lemma 11

We first pick $c(\beta)$ to be a positive constant such that $\lfloor \frac{c(\beta)N_{\max}}{2|\Lambda|} \rfloor - 2 \leq N_{\max}$. Then, since $k = \lfloor \frac{K}{2|\Lambda|} \rfloor - 2$, it is easy to see that for any $K \leq c(\beta)N_{\max} = c(\beta)3\rho sn$, we have $k \leq N_{\max}$. Pick $c = c(\beta)3\rho$. Then, $K \leq csn$ implies $k \leq N_{\max}$.

To estimate the summation on the right hand side of Eqn. (5), we consider the quantity $x_{\max} := \max_i \{x_i\}$. We first enumerate all the possible cases of (x_1, x_2, \dots, x_m) when x_{\max} is fixed, and then vary the value of x_{\max} .

Set $h(y) = \max_{x_{\max}=y} \left\{ \sum_{i=1}^m x_i^2 \right\}$ for $y \geq \lceil \frac{k}{m} \rceil$. It is the maximum value of $\sum_{i=1}^m x_i^2$ under the constraint $x_{\max} = y$. Without loss of generality, we assume $x_1 = y$ and $y \geq x_2 \geq x_3 \geq \dots \geq x_m \geq 0$. We argue that $\arg \max_{x_{\max}=y} \left\{ \sum_{i=1}^m x_i^2 \right\} = \{y, y, \dots, y, k - ry, 0, \dots, 0\}$, that is $x_1 = x_2 = \dots = x_r = y, x_{r+1} = k - ry$ where $r = \lfloor \frac{k}{y} \rfloor$.

To show this, we first consider x_2 : if $x_2 = y$, then consider x_3 ; otherwise, $x_2 < y$, then we search for the largest index j such that $x_j > 0$. Note the fact that if $x \geq y > 0$, then $(x + 1)^2 + (y - 1)^2 = x^2 + y^2 + 2(x - y) + 2 > x^2 + y^2$. So if we increase x_2 by 1 and decrease x_j by 1, we will enlarge $\sum_{i=1}^m x_i^2$. After we update $x_2 = x_2 + 1, x_j = x_j - 1$, we still get a decreasing sequence $x_1 \geq x_2 \geq \dots \geq x_m \geq 0$. If we still have $x_2 < y$, then we repeat the same procedure above (by increasing x_2 and decreasing x_j where j is the largest index such that $x_j > 0$). We repeat this process until $x_2 = y$ or $x_1 + x_2 = k$. If it is the former case (i.e, $x_2 = y$), then we consider x_3 and so on. Finally, we will get the sequence $x_1 = \dots = x_r = y, x_{r+1} = k - ry$ where $r = \lfloor \frac{k}{y} \rfloor$ as claimed, and this maximizes $\sum_{i=1}^m x_i^2$.

Next we claim that $h(y + 1) > h(y)$. The reason is similar to the above. We update the sequence $x_1 = x_2 = \dots = x_r = y, x_{r+1} = k - ry$ (which corresponding to $h(y)$) from x_1 : we increase x_1 by 1; search the largest index s such that $x_s > 0$ and decrease x_s by 1. And then consider x_2 and so on and so forth. This process won't stop until $x_1 = x_2 = \dots = x_q = y + 1$ and $x_{q+1} = k - q(y + 1)$ with $q = \lfloor \frac{k}{y+1} \rfloor$. Thus $h(y + 1) > h(y)$.

By enumerating all the possible values of x_{\max} , we split Eqn. (5) into three parts as follows (corresponding to $x_{\max} = k, x_{\max} \in [\lceil \frac{k+1}{2} \rceil, k - 1]$ and $x_{\max} \in [\lceil \frac{k}{m} \rceil, \lceil \frac{k+1}{2} \rceil - 1]$) (see the remarks after this equation for how the inequality is derived);

$$\begin{aligned}
 & q^{2k} \sum_{\substack{x_1+x_2+\dots+x_m=k \\ x_i \geq 0}} \binom{N_{\max}}{x_1} \binom{N_{\max}}{x_2} \dots \binom{N_{\max}}{x_m} q^{(k^2 - \sum_{i=1}^m x_i^2)/2} \\
 & \leq q^{2k} \binom{N_{\max}}{k} m + q^{2k} \sum_{x_{\max} = \lceil \frac{k+1}{2} \rceil}^{k-1} \left(\binom{m}{1} \binom{N_{\max}}{x_{\max}} \sum_{\substack{y_1+\dots+y_{m-1}=k-x_{\max} \\ x_{\max} \geq y_i \geq 0}} \binom{N_{\max}}{y_1} \dots \right. \\
 & \quad \left. \binom{N_{\max}}{y_{m-1}} q^{x_{\max}(k-x_{\max})} \right) + \binom{mN_{\max}}{k} q^{\frac{(k-1)^2}{4} + 2k}.
 \end{aligned} \tag{14}$$

► **Remark.** The first term on the right hand side of Eqn. (14) comes from the fact that if $x_{\max} = k$, then there are m possible cases for (x_1, x_2, \dots, x_m) . For each case, the value of each term in the summation is $\binom{N_{\max}}{k}$, giving rise to the first term in Eqn. (14).

The third term on the right hand side of Eqn. (14) can be derived as follows. First, observe that

$$\begin{aligned}
 & \sum_{x_{\max} = \lceil \frac{k}{m} \rceil}^{\lceil \frac{k+1}{2} \rceil - 1} \left(\sum_{\substack{x_1+x_2+\dots+x_m=k \\ x_i \geq 0, \max_i \{x_i\} = x_{\max}}} \binom{N_{\max}}{x_1} \binom{N_{\max}}{x_2} \dots \binom{N_{\max}}{x_m} \right) \\
 & \leq \sum_{\substack{x_1+x_2+\dots+x_m=k \\ x_i \geq 0}} \binom{N_{\max}}{x_1} \binom{N_{\max}}{x_2} \dots \binom{N_{\max}}{x_m} = \binom{mN_{\max}}{k}.
 \end{aligned}$$

On the other hand, as $x_{\max} \leq \lceil \frac{k+1}{2} \rceil - 1 = \lceil \frac{k-1}{2} \rceil$, we have:

$$\frac{k^2 - \sum_{i=1}^m x_i^2}{2} \geq \frac{k^2 - h(x_{\max})}{2} \geq \frac{k^2 - h(\lceil \frac{k-1}{2} \rceil)}{2} \geq \frac{(k-1)^2}{4},$$

where the second inequality uses the fact that $h(y)$ is an increasing function, and the last inequality comes from that $h(\lceil \frac{k-1}{2} \rceil) \leq (\lceil \frac{k-1}{2} \rceil)^2 + (\lceil \frac{k-1}{2} \rceil)^2 + 1 \leq k^2/4 + k^2/4 + 1 = k^2/2 + 1$.

To this end, it suffices to estimate all three terms on the right hand side of Eqn. (14).

The first term of Eqn. (14): According to the assumptions in Eqn. (6), we know

$$q \leq \left(\frac{k!}{n^\epsilon N_{\max}^k m} \right)^{1/2k}.$$

Thus, for the first term of Eqn. (14), we have:

$$q^{2k} \binom{N_{\max}}{k} m \leq \left(\frac{k!}{n^\epsilon N_{\max}^k m} \right) \frac{N_{\max}^k m}{k!} = \frac{1}{n^\epsilon}. \tag{15}$$

The second term of Eqn. (14): For the second term of Eqn. (14), we relax the constraint $x_{\max} \geq y_i \geq 0$ to $y_i \geq 0$. Thus, we have:

$$\begin{aligned}
 \sum_{\substack{y_1+\dots+y_{m-1}=k-x_{\max} \\ x_{\max} \geq y_i \geq 0}} \binom{N_{\max}}{y_1} \dots \binom{N_{\max}}{y_{m-1}} & \leq \sum_{\substack{y_1+\dots+y_{m-1}=k-x_{\max} \\ y_i \geq 0}} \binom{N_{\max}}{y_1} \dots \binom{N_{\max}}{y_{m-1}} \\
 & = \binom{(m-1)N_{\max}}{k-x_{\max}} \leq \frac{(m-1)^{k-x_{\max}} N_{\max}^{k-x_{\max}}}{(k-x_{\max})!}.
 \end{aligned} \tag{16}$$

29:18 Local Cliques in ER-Perturbed Random Geometric Graphs

Now apply (16) to the second term of (14), we have (starting from the second line, we replace x_{max} to be j for simplicity):

$$\begin{aligned}
& q^{2k} \sum_{x_{max}=\lceil \frac{k+1}{2} \rceil}^{k-1} \left(\binom{m}{1} \binom{N_{max}}{x_{max}} \sum_{\substack{y_1+\dots+y_{m-1}=k-x_{max} \\ x_{max} \geq y_i \geq 0}} \binom{N_{max}}{y_1} \dots \binom{N_{max}}{y_{m-1}} q^{x_{max}(k-x_{max})} \right) \\
& \leq \sum_{j=\lceil \frac{k+1}{2} \rceil}^{k-1} \left(m \frac{(N_{max})^j}{j!} q^{2k+j(k-j)} \frac{(m-1)^{k-j} N_{max}^{k-j}}{(k-j)!} \right) \\
& < \sum_{j=\lceil \frac{k+1}{2} \rceil}^{k-1} \left(m^{k-j+1} N_{max}^k \binom{k}{k-j} \frac{1}{k!} q^{2k+j(k-j)} \right) \\
& < \sum_{j=\lceil \frac{k+1}{2} \rceil}^{k-1} \left(m^{k-j+1} N_{max}^k \frac{k^{k-j}}{(k-j)!} \frac{1}{k!} q^{2k+j(k-j)} \right). \tag{17}
\end{aligned}$$

Since $q \leq \left(\frac{k!}{k^2 n^\epsilon N_{max}^k m^2} \right)^{1/k}$ by Eqn. (6), for each j satisfying $\lceil \frac{k+1}{2} \rceil \leq j \leq k-1$, we have:

$$\begin{aligned}
& m^{k-j+1} N_{max}^k \frac{k^{k-j}}{(k-j)!} \frac{1}{k!} q^{2k+j(k-j)} \\
& \leq m^{k-j+1} N_{max}^k \frac{k^{k-j}}{(k-j)!} \frac{1}{k!} \frac{k!}{k^2 n^\epsilon N_{max}^k m^2} \left(\frac{k!}{k^2 n^\epsilon N_{max}^k m^2} \right)^{\frac{2k+j(k-j)}{k} - 1} \\
& \leq m^{k-j-1} k^{k-j-1} \left(\frac{k!}{k^2 n^\epsilon N_{max}^k m^2} \right)^{\frac{k+j(k-j)}{k}} \frac{1}{kn^\epsilon} \\
& \leq m^{k-j-1} k^{k-j-1} \left(\frac{k!}{k^2 n^\epsilon N_{max}^k m^2} \right)^{\frac{k-j-1}{2}} \frac{1}{kn^\epsilon} \tag{18} \\
& = \left(\frac{k!}{n^\epsilon N_{max}^k} \right)^{\frac{k-j-1}{2}} \frac{1}{kn^\epsilon} \\
& \leq \frac{1}{kn^\epsilon}. \tag{19}
\end{aligned}$$

Eqn. (18) comes from two facts: 1) $k \leq N_{max}$ (and thus the term $\frac{k!}{k^2 n^\epsilon N_{max}^k m^2} < 1$) and 2) by tedious by elementary calculation, we can show that $\frac{k+j(k-j)}{k} \geq \frac{k-j-1}{2}$ when $\lceil \frac{k+1}{2} \rceil \leq j \leq k-1$. Eqn. (19) holds since $k \leq N_{max}$ (and thus $\frac{k!}{n^\epsilon N_{max}^k} < 1$).

The third term of Eqn. (14): For the third term of (14), plugging in the condition

$$q \leq \left(\frac{k!}{n^\epsilon m^k N_{max}^k} \right)^{\frac{4}{k^2}},$$

we thus have

$$\begin{aligned} \binom{mN_{\max}}{k} q^{\frac{(k-1)^2}{4}+2k} &\leq \frac{(mN_{\max})^k}{k!} q^{\frac{(k-1)^2}{4}+2k} \\ &\leq \frac{(mN_{\max})^k}{k!} \frac{k!}{n^\epsilon m^k N_{\max}^k} \left(\frac{k!}{n^\epsilon m^k N_{\max}^k} \right)^{\frac{4}{k^2} \left(\frac{(k-1)^2}{4}+2k \right) - 1} \leq \frac{1}{n^\epsilon} \end{aligned} \quad (20)$$

where the last inequality holds as $\frac{k!}{n^\epsilon m^k N_{\max}^k} < 1$.

Finally, combining (15), (19) and (20), we have:

$$E[X | \mathbb{F}] \leq \frac{1}{n^\epsilon} + \frac{k}{2} \cdot \frac{1}{kn^\epsilon} + \frac{1}{n^\epsilon} = \frac{5}{2n^\epsilon}.$$

This proves Lemma 11.

A.2 Existences of constants c_2^a and c_3^a

We claim that there exist constants c_2^a and c_3^a (which depend on the doubling constant L of μ , the Besicovitch constant β , and the regularity constant ρ), such that if

$$q \leq c_2^a \cdot \left(\frac{1}{n} \right)^{c_3^a/K} \cdot \frac{K}{sn},$$

then the conditions in Eqn. (6) will hold. We prove this by elementary calculation below, where we will use the Stirling's approximation $k! > \sqrt{2\pi} k^k e^{-k}$ and the fact that $k \leq N_{\max} = 3\rho sn$ ($K \leq csn$ implies this due to the choice of c in the proof of Lemma 11) and $m \leq n$ (where recall that m is the size of the number of clusters in the clique-decomposition of P_i).

$$\begin{aligned} \left(\frac{k!}{n^\epsilon N_{\max}^k m} \right)^{\frac{1}{2k}} &> \left(\frac{\sqrt{2\pi} k^k e^{-k}}{n^{1+\epsilon}} \right)^{\frac{1}{2k}} \left(\frac{1}{3\rho sn} \right)^{\frac{1}{2}} = \left(e^{-\frac{1}{2}} (2\pi)^{\frac{1}{4k}} \right) \left(\frac{1}{n} \right)^{\frac{1+\epsilon}{2k}} \left(\frac{k}{3\rho sn} \right)^{\frac{1}{2}} \\ \left(\frac{k!}{k^2 n^\epsilon N_{\max}^k m^2} \right)^{\frac{1}{k}} &> \left(e^{-1} (2\pi)^{\frac{1}{4k}} k^{-\frac{2}{k}} \right) \left(\frac{1}{n} \right)^{\frac{2+\epsilon}{k}} \left(\frac{k}{3\rho sn} \right) \\ \left(\frac{k!}{n^\epsilon m^k N_{\max}^k} \right)^{\frac{4}{k^2}} &> \left((2\pi)^{\frac{2}{k^2}} e^{-\frac{4}{k}} \right) \left(\frac{1}{n} \right)^{\frac{4(k+\epsilon)}{k^2}} \left(\frac{k}{3\rho sn} \right)^{\frac{4}{k}}. \end{aligned}$$

Thus, by comparing the exponents of each term, we know that if $q \leq \frac{1}{3\rho e} \left(\frac{1}{n} \right)^{\frac{4+\epsilon}{k}} \left(\frac{k}{sn} \right)$, then the condition on q holds. Finally, since $k = \lfloor \frac{K}{2|\Lambda|} \rfloor - 2$, easy to see there exists constants c_2^a, c_3^a such that the constraint $q \leq c_2^a \left(\frac{1}{n} \right)^{c_3^a/K} \frac{K}{sn}$ implies the condition.

A.3 The missing details in case (B) of Theorem 10

Denote by \mathbb{H} the event that “for every $v \in V$, the ball $B_r(v) \cap V$ contains at most $3L\rho sn$ points”, and \mathbb{H}^c is its complement. By an argument similar to that of Claim 9, we have that $\mathbb{P}[\mathbb{H}^c] \leq n^{-5}$. Set $N_u := |B_r^V(u)|$ and $N_v := |B_r^V(v)|$. Let $\tilde{k} := \lfloor \frac{K}{2} \rfloor - 2$. For every set S of $(\tilde{k} + 2)$ vertices in \tilde{G}_{uv}^{local} , let A_S be the event “ S is a uv -clique in \tilde{G}_{uv}^{local} ” and Y_S its indicator random variable. Set

$$Y = \sum_{|S|=\tilde{k}+2} Y_S.$$

29:20 Local Cliques in ER-Perturbed Random Geometric Graphs

Then Y is the number of uv -cliques of size $(\tilde{k} + 2)$ in \tilde{G}_{uv}^{local} . Linearity of expectation gives:

$$\mathbb{E}[Y \mid \mathbf{H}] = \sum_{|S|=\tilde{k}+2} \mathbb{E}[Y_S \mid \mathbf{H}] = \sum_{\substack{x_1+x_2=\tilde{k} \\ 0 \leq x_1 \leq N_u-1 \\ 0 \leq x_2 \leq N_v-1}} \binom{N_u-1}{x_1} \binom{N_v-1}{x_2} q^{(x_1+1)(x_2+1)-1}. \quad (21)$$

To estimate this quantity, we first prove the following result:

► **Lemma 18.** *For any constant $\epsilon > 0$, we have that $\mathbb{E}[Y \mid \mathbf{H}] = O(n^{-\epsilon})$ as long as the following condition on q holds:*

$$q \leq \min \left\{ \left(\frac{\tilde{k}!}{\tilde{k}^2 n^\epsilon (N_u + N_v)^{\tilde{k}}} \right)^{1/\tilde{k}}, \left(\frac{\tilde{k}!}{n^\epsilon (N_u + N_v)^{\tilde{k}}} \right)^{16/\tilde{k}^2} \right\}. \quad (22)$$

Specifically, setting $\epsilon = 3$ (a case which we will use later), we have $\mathbb{E}[Y \mid \mathbf{H}] = O(n^{-3})$.

The proof of this technical result can be found in Appendix A.4.

Note that if event \mathbf{H} is true, then $N_u + N_v \leq 6L\rho sn$. In this case, there exist two constants c_2^b and c_3^b which depend on the doubling constant L of μ , the Besicovitch constant β , and the regularity constant ρ , such that if $K \leq 12L\rho sn$ and

$$q \leq c_2^b \cdot \left(\frac{1}{n} \right)^{c_3^b/K} \cdot \frac{K}{sn},$$

then the conditions in Eqn. (22) will hold (the simple proof of this can be found in Appendix A.5).

On the other hand, we have

$$\begin{aligned} \mathbb{P} \left[\tilde{G}_{uv}^{local} \text{ has a } uv\text{-clique of size } \geq \frac{K}{2} \right] &= \mathbb{P}[Y > 0] \\ &= \mathbb{P}[Y > 0 \mid \mathbf{H}] \cdot \mathbb{P}[\mathbf{H}] + \mathbb{P}[Y > 0 \mid \mathbf{H}^c] \cdot \mathbb{P}[\mathbf{H}^c] \\ &\leq \mathbb{P}[Y > 0 \mid \mathbf{H}] + \mathbb{P}[\mathbf{H}^c] \\ &\leq \mathbb{E}[Y \mid \mathbf{H}] + n^{-5}. \end{aligned}$$

Thus, by Lemma (18), we know that

$$\begin{aligned} \text{If } K \leq 6L\rho sn \text{ and } q \leq c_2^b \cdot \left(\frac{1}{n} \right)^{c_3^b/K} \cdot \frac{K}{sn}, \text{ then} \\ \mathbb{P} \left[\tilde{G}_{uv}^{local} \text{ has a } uv\text{-clique of size } \geq \frac{K}{2} \right] = O(n^{-3}). \end{aligned} \quad (23)$$

Finally, suppose $K > K_1 = 12L\rho sn$. Set

$$c_1^b = c_2^b \cdot \left(\frac{1}{n} \right)^{c_3^b/(12L\rho \ln n)} \cdot \frac{K_1}{sn} \leq c_2^b \cdot \left(\frac{1}{n} \right)^{c_3^b/K_1} \cdot \frac{K_1}{sn}$$

where the inequality holds as by Assumption-A $sn > \ln n$. Plugging in $K_1 = 12L\rho sn$ to the definition of c_1^b , it is then easy to see that c_1^b is a positive constant. Using Eqn. (23), we know that if $q \leq c_1^b$ and $K > K_1 = 12L\rho sn$, then

$$\begin{aligned} \mathbb{P} \left[\tilde{G}_{uv}^{local} \text{ has a } uv\text{-clique of size } \geq \frac{K}{2} \right] &\leq \mathbb{P} \left[\tilde{G}_{uv}^{local} \text{ has a } uv\text{-clique of size } \geq \frac{K_1}{2} \right] \\ &= O(n^{-3}). \end{aligned}$$

Combining this with the discussion above and applying Lemma 18, we have

$$\text{If } q \leq \min \left\{ c_1^b, c_2^b \cdot \left(\frac{1}{n} \right)^{c_3^b/K} \cdot \frac{K}{sn} \right\}, \text{ then}$$

$$\mathbb{P} \left[\tilde{G}_{uv}^{local} \text{ has a } uv\text{-clique of size } \geq \frac{K}{2} \right] = O(n^{-3}).$$

A.4 The proof of Lemma 18

Proof. It is easy to see that if $K > 2(N_u + N_v)$, then $\tilde{k} > N_u + N_v - 2$ which implies that the summation on the right hand side of Eqn. (21) is 0. Now let's focus on the case when $K \leq 2(N_u + N_v)$. In this case, we have $\tilde{k} \leq (N_u - 1) + (N_v - 1) < N_u + N_v$. Note that the right hand side of (21) can be bounded from above by:

$$\begin{aligned} & \sum_{\substack{x_1+x_2=\tilde{k} \\ 0 \leq x_1 \leq N_u-1 \\ 0 \leq x_2 \leq N_v-1}} \binom{N_u-1}{x_1} \binom{N_v-1}{x_2} q^{(x_1+1)(x_2+1)-1} \leq q^{\tilde{k}} \sum_{i=0}^{\tilde{k}} \binom{N_u}{i} \binom{N_v}{\tilde{k}-i} q^{i(\tilde{k}-i)} \\ & \leq q^{\tilde{k}} \left(\sum_{i=0}^{\lfloor \frac{\tilde{k}}{4} \rfloor} \left[\binom{N_u}{i} \binom{N_v}{\tilde{k}-i} + \binom{N_u}{\tilde{k}-i} \binom{N_v}{i} \right] q^{i(\tilde{k}-i)} \right) + \binom{N_u+N_v}{\tilde{k}} q^{\tilde{k} + \frac{\tilde{k}^2}{16}}. \end{aligned} \quad (24)$$

Eqn. (24) is due to the fact that when $\lfloor \frac{\tilde{k}}{4} \rfloor + 1 \leq i \leq \tilde{k} - \lfloor \frac{\tilde{k}}{4} \rfloor - 1$, we have

$$i(\tilde{k}-i) \geq \left(\left\lfloor \frac{\tilde{k}}{4} \right\rfloor + 1 \right) \left(\left\lfloor \frac{\tilde{k}}{4} \right\rfloor + 1 \right) \geq \frac{\tilde{k}^2}{16}.$$

Now it suffices to estimate the two terms on the right hand side of Eqn. (24).

The first term of Eqn. (24): For the first term of (24), we have the following estimate:

$$\begin{aligned} \left[\binom{N_u}{i} \binom{N_v}{\tilde{k}-i} + \binom{N_u}{\tilde{k}-i} \binom{N_v}{i} \right] q^{\tilde{k}+i(\tilde{k}-i)} & \leq \frac{N_u^i N_v^{\tilde{k}-i} + N_u^{\tilde{k}-i} N_v^i}{i!(\tilde{k}-i)!} q^{\tilde{k}} q^{i(\tilde{k}-i)} \\ & \leq \frac{(N_u + N_v)^{\tilde{k}}}{i!(\tilde{k}-i)!} q^{\tilde{k}} q^{i(\tilde{k}-i)}. \end{aligned}$$

By plugging in the condition $q \leq \left(\frac{\tilde{k}!}{\tilde{k}^2 n^\epsilon (N_u + N_v)^{\tilde{k}}} \right)^{1/\tilde{k}}$, we have:

$$\frac{(N_u + N_v)^{\tilde{k}}}{i!(\tilde{k}-i)!} q^{\tilde{k}} q^{i(\tilde{k}-i)} \leq \frac{(N_u + N_v)^{\tilde{k}}}{i!(\tilde{k}-i)!} \frac{\tilde{k}!}{\tilde{k}^2 n^\epsilon (N_u + N_v)^{\tilde{k}}} q^{i(\tilde{k}-i)} = \frac{\tilde{k}!}{i!(\tilde{k}-i)!} q^{i(\tilde{k}-i)} \frac{1}{\tilde{k}^2 n^\epsilon}.$$

For $i = 0$, we have $\frac{\tilde{k}!}{i!(\tilde{k}-i)!} q^{i(\tilde{k}-i)} \frac{1}{\tilde{k}^2 n^\epsilon} = \frac{1}{\tilde{k}^2 n^\epsilon}$. For $i \geq 1$, note that $1 \leq i \leq \lfloor \frac{\tilde{k}}{4} \rfloor$ implies

29:22 Local Cliques in ER-Perturbed Random Geometric Graphs

$\frac{i(\tilde{k}-i)}{\tilde{k}} \geq \frac{i}{2}$. Thus, we have:

$$\begin{aligned} \frac{\tilde{k}!}{i!(\tilde{k}-i)!} q^{i(\tilde{k}-i)} \frac{1}{\tilde{k}^2 n^\epsilon} &\leq \frac{\tilde{k}^i}{i!} \left(\frac{\tilde{k}!}{\tilde{k}^2 n^\epsilon (N_u + N_v)^{\tilde{k}}} \right)^{\frac{i(\tilde{k}-i)}{\tilde{k}}} \frac{1}{\tilde{k}^2 n^\epsilon} \\ &\leq \frac{\tilde{k}^i}{i!} \left(\frac{\tilde{k}!}{\tilde{k}^2 n^\epsilon (N_u + N_v)^{\tilde{k}}} \right)^{\frac{i}{2}} \frac{1}{\tilde{k}^2 n^\epsilon} \\ &\leq \left(\frac{\tilde{k}!}{n^\epsilon (N_u + N_v)^{\tilde{k}}} \right)^{\frac{i}{2}} \frac{1}{\tilde{k}^2 n^\epsilon} \\ &\leq \frac{1}{\tilde{k}^2 n^\epsilon}. \end{aligned}$$

The last two inequalities hold since $\tilde{k} \leq N_u + N_v$. Therefore,

$$q^{\tilde{k}} \sum_{i=0}^{\lfloor \frac{\tilde{k}}{4} \rfloor} \left[\binom{N_u}{i} \binom{N_v}{\tilde{k}-i} + \binom{N_u}{\tilde{k}-i} \binom{N_v}{i} \right] q^{i(\tilde{k}-i)} \leq \frac{\tilde{k}}{4} \frac{1}{\tilde{k}^2 n^\epsilon} = \frac{1}{4\tilde{k}n^\epsilon}. \quad (25)$$

The second term of Eqn. (24): For the second term of (24), directly plugging in the condition $q \leq \left(\frac{\tilde{k}!}{(N_u + N_v)^{\tilde{k}n^\epsilon} \right)^{16/\tilde{k}^2}$, we have:

$$\binom{N_u + N_v}{\tilde{k}} q^{\tilde{k} + \frac{\tilde{k}^2}{16}} \leq \frac{(N_u + N_v)^{\tilde{k}}}{\tilde{k}!} \frac{\tilde{k}!}{(N_u + N_v)^{\tilde{k}n^\epsilon}} \left(\frac{\tilde{k}!}{(N_u + N_v)^{\tilde{k}n^\epsilon}} \right)^{\frac{16}{\tilde{k}^2} \left(\tilde{k} + \frac{\tilde{k}^2}{16} \right) - 1} \leq \frac{1}{n^\epsilon}. \quad (26)$$

Finally, combining (25) and (26), we have:

$$\mathbb{E}[Y \mid \mathbf{H}] \leq \frac{1}{4\tilde{k}n^\epsilon} + \frac{1}{n^\epsilon} < \frac{2}{n^\epsilon}.$$

This finishes the proof of Lemma 18. ◀

A.5 Existences of constants c_2^b and c_3^b

Note that as event H holds, we have $N_u + N_v \leq 6L\rho sn$. Also note that if $K \leq 12L\rho sn$, then $\tilde{k} \leq 6L\rho sn$. Hence

$$\begin{aligned} \left(\frac{\tilde{k}!}{\tilde{k}^2 n^\epsilon (N_u + N_v)^{\tilde{k}}} \right)^{\frac{1}{\tilde{k}}} &> \left(\frac{\sqrt{2\pi} \tilde{k}^{\tilde{k}} e^{-\tilde{k}}}{\tilde{k}^2 n^\epsilon} \right)^{\frac{1}{\tilde{k}}} \frac{1}{6L\rho sn} = \left((2\pi)^{\frac{1}{2\tilde{k}}} \tilde{k}^{-\frac{2}{\tilde{k}}} e^{-1} \right) \left(\frac{1}{n} \right)^{\frac{\epsilon}{\tilde{k}}} \left(\frac{\tilde{k}}{6L\rho sn} \right) \\ \left(\frac{\tilde{k}!}{n^\epsilon (N_u + N_v)^{\tilde{k}}} \right)^{\frac{16}{\tilde{k}^2}} &> (2\pi)^{\frac{8}{\tilde{k}^2}} e^{-\frac{16}{\tilde{k}}} \left(\frac{1}{n} \right)^{\frac{16\epsilon}{\tilde{k}^2}} \left(\frac{\tilde{k}}{6L\rho sn} \right)^{\frac{16}{\tilde{k}}}. \end{aligned}$$

Finally, observe that $(2\pi)^{\frac{8}{\tilde{k}^2}} > 1$, $e^{-\frac{16}{\tilde{k}}} > \frac{1}{e}$. It then follows that there exists constants c_2^b, c_3^b such that $c_2^b \left(\frac{1}{n} \right)^{c_3^b/K} \frac{K}{sn}$ is smaller than the last term in the right hand side of each equation above. Hence $\tilde{k} \leq 6L\rho sn$ and $q \leq c_2^b \left(\frac{1}{n} \right)^{c_3^b/K} \frac{K}{sn}$ implies the condition on q as in Eqn. (22).

Local Routing in Sparse and Lightweight Geometric Graphs

Vikrant Ashvinkumar

University of Sydney, Australia
vash7242@uni.sydney.edu.au

Joachim Gudmundsson

University of Sydney, Australia
joachim.gudmundsson@sydney.edu.au

Christos Levcopoulos

Lund University, Sweden
christos.levcopoulos@cs.lth.se

Bengt J. Nilsson

Malmö University, Sweden
bengt.nilsson.ts@mau.se

André van Renssen

University of Sydney, Australia
andre.vanrenssen@sydney.edu.au

Abstract

Online routing in a planar embedded graph is central to a number of fields and has been studied extensively in the literature. For most planar graphs no $O(1)$ -competitive online routing algorithm exists. A notable exception is the Delaunay triangulation for which Bose and Morin [6] showed that there exists an online routing algorithm that is $O(1)$ -competitive. However, a Delaunay triangulation can have $\Omega(n)$ vertex degree and a total weight that is a linear factor greater than the weight of a minimum spanning tree.

We show a simple construction, given a set V of n points in the Euclidean plane, of a planar geometric graph on V that has small weight (within a constant factor of the weight of a minimum spanning tree on V), constant degree, and that admits a local routing strategy that is $O(1)$ -competitive. Moreover, the technique used to bound the weight works generally for any planar geometric graph whilst preserving the admission of an $O(1)$ -competitive routing strategy.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Computational geometry, Spanners, Routing

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.30

Related Version A full version of the paper is available at <https://arxiv.org/abs/1909.10215>.

Funding *Joachim Gudmundsson*: Funded by the Australian Government through the Australian Research Council DP150101134 and DP180102870.

Christos Levcopoulos: Swedish Research Council grants 2017-03750 and 2018-04001.

1 Introduction

The aim of this paper is to design a graph on V (a finite set of points in the Euclidean plane) that is cheap to build and easy to route on. Consider the problem of finding a route in a geometric graph from a given source vertex s to a given target vertex t . Routing in a geometric graph is a fundamental problem that has received considerable attention in the literature. In the offline setting, when we have full knowledge of the graph, the problem is well-studied and numerous algorithms exist for finding shortest paths (for example, the classic



© Vikrant Ashvinkumar, Joachim Gudmundsson, Christos Levcopoulos, Bengt J. Nilsson, and André van Renssen;

licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 30; pp. 30:1–30:13



Leibniz International Proceedings in Informatics

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

Dijkstra’s Algorithm [9]). In an online setting the problem becomes much more complex. The route is constructed incrementally and at each vertex a local decision has to be taken to decide which vertex to forward the message to. Without knowledge of the full graph, an online routing algorithm cannot identify a shortest path in general; the goal is to follow a path whose length approximates that of the shortest path.

Given a source vertex s , a target vertex t , and a message m , the aim is for an online routing algorithm to send m together with a header h from s to t in a graph G . Initially the algorithm only has knowledge of s , t and the neighbors of s , denoted $\mathcal{N}(s)$. Note that it is commonly assumed that for a vertex v , the set $\mathcal{N}(v)$ also includes information about the coordinates of the vertices in $\mathcal{N}(v)$. Upon receiving a message m and its header h , a vertex v must select one of its neighbours to forward the message to as a function of h and $\mathcal{N}(v)$. This procedure is repeated until the message reaches the target vertex t . Different routing algorithms are possible depending on the size of h and the part of G that is known to each vertex. Usually, there is a trade-off between the amount of information that is stored in the header and the amount of information that is stored in the vertices.

Bose and Morin [6] showed that greedy routing always reaches the intended destination on Delaunay triangulations. Dhandapani [8] proved that every triangulation can be embedded in such a way that it allows greedy routing and Angelini et al. [1] provided a constructive proof.

However, the above papers only prove that a greedy routing algorithm will succeed on the specific graphs therein. No attention is paid to the quality or *competitiveness* of the resulting path relative to the shortest path. Bose and Morin [6] showed that many local routing strategies are not competitive but also show how to route competitively in a Delaunay triangulation. Bonichon et al. [3, 4] provided different local routing algorithms for the Delaunay triangulation, decreasing the competitive ratio, and Bonichon et al. [2] designed a competitive routing algorithm for Gabriel triangulations.

To the best of our knowledge most of the existing routing algorithms consider well-known graph classes such as triangulations and Θ -graphs. However, these graphs are generally very expensive to build. Typically, they have high degree ($\Omega(n)$) and the total length of their edges can be as bad as $\Omega(n) \cdot wt(MST(V))$.

On the other hand, there is a large amount of research on constructing geometric planar graphs with “good” properties. However, none of these have been shown to have all of bounded degree, weight, planarity, and the admission of competitive local routing. Bose et al. [5] come tantalisingly close by providing a local routing algorithm for a plane bounded-degree spanner.

In this paper we consider the problem of constructing a geometric graph of small weight and small degree that guarantees a local routing strategy that is $O(1)$ -competitive. More specifically we show:

Given a set V of n points in the plane, together with two parameters $0 < \theta < \pi/2$ and $r > 0$, we show how to construct in $O(n \log n)$ time a planar $((1 + 1/r) \cdot \tau)$ -spanner with degree at most $5\lceil 2\pi/\theta \rceil$, and weight at most $((2r + 1) \cdot \tau)$ times the weight of a minimum spanning tree of V , where $\tau = 1.998 \cdot \max(\pi/2, \pi \sin(\theta/2) + 1)$. This construction admits an $O(1)$ -memory deterministic 1-local routing algorithm with a routing ratio of no more than $5.90 \cdot (1 + 1/r) \cdot \max(\pi/2, \pi \sin(\theta/2) + 1)$.

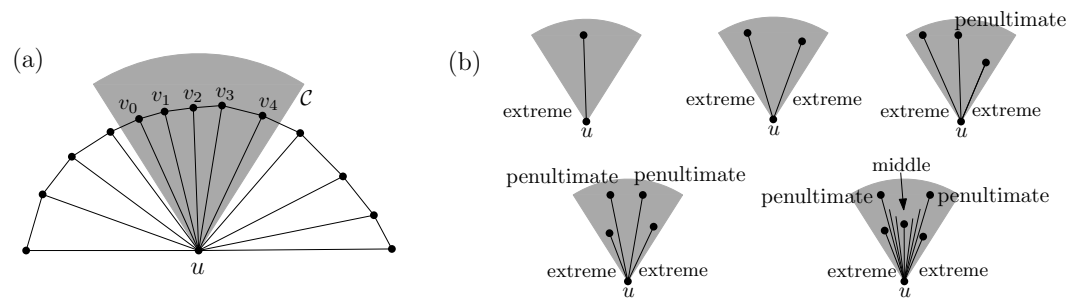
While we focus on our construction, we note that the techniques used to bound the weight of the graph apply generally to any planar geometric graph. In particular, using techniques similar to the ones we use, it may be possible to extend the results by Bose et al. [5] to obtain other routing algorithms for bounded-degree light spanners.

2 Building the Network

Given a Delaunay triangulation $\mathcal{DT}(V)$ of a point set V we will show that one can remove edges from $\mathcal{DT}(V)$ such that the resulting graph $\mathcal{BDG}(V)$ has constant degree and constant stretch-factor. We will also show that the resulting graph has the useful property that for every Delaunay edge (u, v) in the Delaunay triangulation there exists a spanning path along the boundary of the face in $\mathcal{BDG}(V)$ containing u and v . This property will be critical to develop the routing algorithm in Section 3. In Section 4 we will show how to prune $\mathcal{BDG}(V)$ further to guarantee the lightness property.

2.1 Building a Bounded Degree Spanner

The idea behind the construction is slightly reminiscent to that of the Θ -graph: For a given parameter $0 < \theta < \pi/2$, let $\kappa = \lceil 2\pi/\theta \rceil$ and let $\mathcal{C}_{u,\kappa}$ be a set of κ disjoint cones partitioning the plane, with each cone having angle measure at most θ at apex u . Let v_0, \dots, v_m be the clockwise-ordered Delaunay neighbours of u within some cone $C \in \mathcal{C}_{u,\kappa}$ (see Figure 1a).



■ **Figure 1** (a) An example of the vertices in some cone C with apex u . (b) Extreme, penultimate, and middle are mutually exclusive properties taking precedence in that order.

Call edges uv_0 and uv_m *extreme* at u . Call edges uv_1 and uv_{m-1} *penultimate* at u if there are two distinct extreme edges at u induced by C . If there are two distinct edges that are extreme at u induced by C and two distinct edges that are penultimate at u induced by C , then, of the remaining edges incident to u and contained in C , the shortest one is called the *middle* edge at u (see Figure 1b). We emphasise that: (1) If there are fewer than three neighbours of u in the cone C , then there are no penultimate edges induced by C ; and (2) If there are fewer than five neighbours of u in the cone C , then there is no middle edge induced by C .

The construction removes every edge except the extreme, penultimate, and middle ones in every $C \in \mathcal{C}_{u,\kappa}$, for every point u , in any order. The edges present in the final construction are thus the ones which are either extreme, penultimate, or middle at both of their endpoints (not necessarily the same at each endpoint).

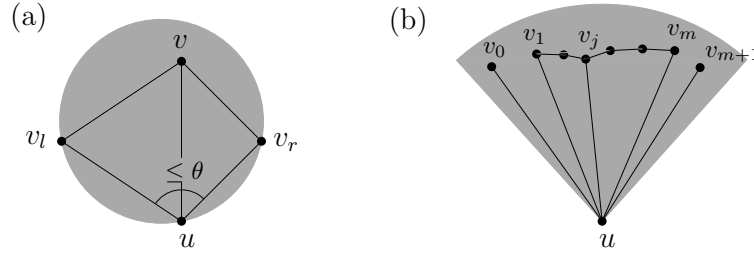
The resulting graph is denoted by $\mathcal{BDG}(V)$. The construction time of this graph is dominated by constructing the Delaunay triangulation, which requires $O(n \log n)$ time. Given the Delaunay triangulation, determining which edges to remove takes linear time. The degree of $\mathcal{BDG}(V)$ is bounded by 5κ , since each of the κ cones $C \in \mathcal{C}_{u,\kappa}$ can induce at most five edges. It remains to bound the spanning ratio.

2.2 Spanning Ratio

Before proving that the network is a spanner (Corollary 7) we will need to prove some basic properties regarding the edges in $\mathcal{BDG}(V)$. We start with a simple but crucial observation about consecutive Delaunay neighbours of a vertex u .

► **Lemma 1.** *Let C be a cone with apex u and angle measure $0 < \theta < \pi/2$. Let v_l, v, v_r be consecutive clockwise-ordered Delaunay neighbours of u contained in C . The interior angle $\angle(v_l, v, v_r)$ must be at least $\pi - \theta$.*

Proof. In the case when $\angle(v_l, v, v_r)$ is reflex in the quadrilateral $\Delta u, v_l, v, v_r$ the lemma trivially holds. Let us thus examine the case when $\angle(v_l, v, v_r)$ is not, in which case the quadrilateral $\Delta u, v_l, v, v_r$ is convex. Since v_l and v_r lie in a cone of angle measure θ , $\angle(v_l, u, v_r)$ is at most θ . Consequently, $\angle(v_l, u, v_r) + \angle(v_l, v, v_r)$ must be at least π (see Figure 2a). Hence, $\angle(v_l, v, v_r)$ is at least $\pi - \theta$. ◀



■ **Figure 2** (a) Example placement of u, v_l, v_r and v in the circle $\circ(v_l, u, v_r)$ (b) The path from v_1 to v_m along the hull of u must be in $\mathcal{BDG}(V)$. Furthermore, uv_0 and uv_{m+1} are extreme, uv_1 and uv_m are penultimate, and uv_j is a middle edge.

This essentially means that $\angle(v_l, v, v_r)$ is wide, and will help us to argue when $v_l v$ and $v v_r$ must be in $\mathcal{BDG}(V)$ (Lemma 5). Next, we define *protected* edges.

► **Definition 2.** *An edge uv is protected at u (with respect to some fixed $\mathcal{C}_{u,\kappa}$) if it is extreme, penultimate, or middle at u . An edge uv is fully protected if it is protected at both u and v .*

Hence, an edge is contained in $\mathcal{BDG}(V)$ if and only if it is fully protected. We continue with an observation that allows us to argue which edges are fully protected.

► **Observation 3.** *If an edge uv_i is not extreme at u , then u must have consecutive clockwise-ordered Delaunay neighbours v_{i-1}, v_i, v_{i+1} , all in the same cone $C \in \mathcal{C}_{u,\kappa}$. Similarly, if uv_i is neither extreme nor penultimate at u , then u must have consecutive clockwise-ordered Delaunay neighbours $v_{i-2}, v_{i-1}, v_i, v_{i+1}, v_{i+2}$, all in the same cone $C \in \mathcal{C}_{u,\kappa}$.*

► **Lemma 4.** *Every edge that is penultimate or middle at one of its endpoints is fully protected.*

Proof. Consider an edge uv that is penultimate or middle at u . Since it is protected at u , we need to show that it is protected at v . Since uv is not extreme at u , u must have consecutive clockwise-ordered Delaunay neighbours v_l, v, v_r in the same cone by Observation 3.

We show that uv must be extreme at v . Suppose for a contradiction that uv is not extreme at v . Then, by Observation 3, $v_l v$ and $v v_r$ are contained in the same cone with apex v and angle at most $\theta < \pi/2$. However, by Lemma 1, $\angle(v_l, v, v_r) \geq \pi - \theta > \theta$, which is impossible. Thus, uv is extreme at v and protected at v . Hence, the edge is fully protected. ◀

Now we can argue about the Delaunay edges along the hull of a vertex (see Figure 2b for an illustration of the lemma).

► **Lemma 5.** *Let v_0, \dots, v_{m+1} be the clockwise-ordered Delaunay neighbours of u contained in some cone $C \in \mathcal{C}_{u, \kappa}$. The edges in the path v_1, \dots, v_m are all fully protected.*

Proof. Let $v_i v_{i+1}$ be an edge along this path. Suppose for a contradiction that $v_i v_{i+1}$ is not protected at v_i . It is thus neither extreme, penultimate, nor middle at v_i . Then, by Observation 3, $v_i u$ and $v_i v_{i-1}$ must be contained in the same cone with apex v_i as $v_i v_{i+1}$. By Lemma 1, $\angle(v_{i-1}, v_i, v_{i+1}) \geq \pi - \theta > \theta$, contradicting that v_{i-1} , v_i , and v_{i+1} lie in the same cone. Such an edge must therefore be either extreme or penultimate, and thus protected, at $v_{i \geq 1}$. An analogous argument shows that the edge is either extreme or penultimate at $v_{i+1 \leq m}$. It is thus fully protected. ◀

Since these paths v_1, \dots, v_m are included in $\mathcal{BDG}(V)$, we can modify the proof of Theorem 3 by Li and Wang [11] to suit our construction to prove that $\mathcal{BDG}(V)$ is a spanner.

► **Theorem 6.** *$\mathcal{BDG}(V)$ is a $\max(\pi/2, \pi \sin(\theta/2) + 1)$ -spanner of the Delaunay triangulation $\mathcal{DT}(V)$ for an adjustable parameter $0 < \theta < \pi/2$.*

Putting the results from this section together, using that the Delaunay triangulation is a 1.998-spanner [12], and observing that $\mathcal{BDG}(V)$ is trivially planar since it is a subgraph of the Delaunay triangulation, we obtain:

► **Corollary 7.** *Given a set V of n points in the plane and a parameter $0 < \theta < \pi/2$, one can in $O(n \log n)$ time compute a graph $\mathcal{BDG}(V)$ that is a planar τ -spanner having degree at most $5 \lceil 2\pi/\theta \rceil$, where $\tau = 1.998 \cdot \max(\pi/2, \pi \sin(\theta/2) + 1)$.*

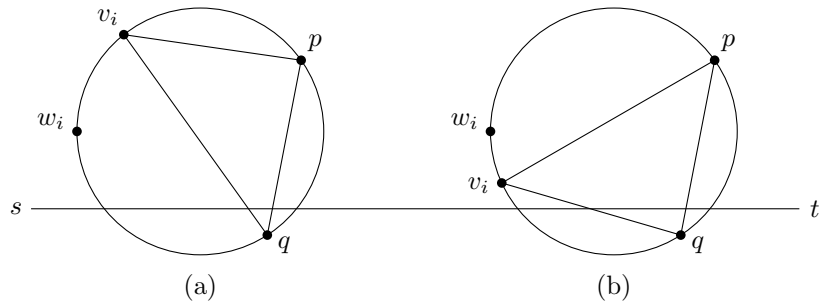
From the proof of Theorem 6 it follows that for every Delaunay edge (u, v) that is *not* in $\mathcal{BDG}(V)$, there is a path from u to v along the face containing u and v realising a path of length at most $\tau \cdot |uv|$. This is a key observation that will be used in Section 3.

3 Routing

In order to route efficiently on $\mathcal{BDG}(V)$, we modify the local routing algorithm by Bonichon et al. [4]. Given a source s and a destination t on the Delaunay triangulation $\mathcal{DT}(V)$, we assume without loss of generality that the line segment $[st]$ is horizontal with s to the left of t . This routing algorithm then works as follows: when we are at a vertex v_i ($v_0 = s$), set v_{i+1} to t and terminate if $v_i t$ is an edge in $\mathcal{DT}(V)$. Otherwise, consider the rightmost Delaunay triangle $T_i = \Delta v_i, p, q$ at v_i that has a non-empty intersection with $[st]$. Denote the circumcircle $\circ(v_i, p, q)$ with C_i , denote the leftmost point of C_i with w_i and the rightmost intersection of C_i and $[st]$ with r_i .

- If v_i is encountered in the clockwise walk along C_i from w_i to r_i , set v_{i+1} to p , the first vertex among $\{p, q\}$ encountered on this walk starting from v_i (see Figure 3a).
- Otherwise, set v_{i+1} to q , the first vertex among $\{p, q\}$ to be encountered in the counter-clockwise walk along C_i starting from v_i (see Figure 3b).

We modify this algorithm in such a way that it no longer necessarily uses the rightmost intersected triangle: At $v_{i > 0}$, we will find a Delaunay triangle A_i based on the Delaunay triangle $A_{i-1} = \Delta v_{i-1}, v_i, f$ used in the routing decision at v_{i-1} ($A_0 = T_0$).



■ **Figure 3** The routing choice: (a) At v_i we follow the edge to p . (b) At v_i we follow the edge to q .

Let $A_i = \Delta v_i, p, q$ be a Delaunay triangle with a non-empty intersection with $[st]$ to the right of the intersection of A_{i-1} with $[st]$. Moreover, if v_i is above $[st]$, then, when making a counterclockwise sweep centred at v_i starting from $v_i v_{i-1}$, we encounter $v_i q$ before $v_i p$, with $v_i q$ intersecting $[st]$ and $v_i p$ not intersecting $[st]$. An analogous statement holds when v_i lies below $[st]$, sweeping in clockwise direction.

We note that these triangles A_i always exist, since the rightmost Delaunay triangle is a candidate. Furthermore, the triangles occur in order along $[st]$ by definition. This implies that the routing algorithm terminates.

► **Theorem 8.** *The modified routing algorithm on the Delaunay triangulation is 1-local and has a routing ratio of at most $(1.185043874 + 3\pi/2) \approx 5.90$.*

Proof. The 1-locality follows by construction. The proof for the routing ratio of Bonichon et al.'s routing algorithm [4] holds for our routing algorithm, since the only parts of their proof using the property that T_i is rightmost are:

1. The termination of the algorithm (which we argued above).
2. The categorisation of the Worst Case Circles of Delaunay triangles T_i into three mutually exclusive cases (which we discuss next).

Thus, the modified routing algorithm on the Delaunay triangulation has a routing ratio of at most $(1.185043874 + 3\pi/2) \approx 5.90$. ◀

3.1 Worst Case Circles

In the analysis of the routing ratio of Bonichon et al.'s routing algorithm [4], the notion of Worst Case Circles is introduced whereby the length of the path yielded by the algorithm is bounded above by some path consisting of arcs along these Worst Case Circles; this arc-path is then shown to have a routing ratio of 5.90.

Suppose we have a candidate path, and are given a Delaunay triangle $\Delta v_i, v_{i+1}, u$ intersecting $[st]$; we denote its circumcircle by C_i with centre O_i . The Worst Case Circle C'_i is a circle that goes through v_i and v_{i+1} , whose centre O'_i is obtained by starting at O_i and moving it along the perpendicular bisector of $[v_i v_{i+1}]$ until either st is tangent to C'_i or v_i is the leftmost point of C'_i , whichever occurs first. The direction O'_i is moved in depends on the routing decision at v_i : if v_i is encountered on the clockwise walk from w_i to r_i , then O'_i is moved towards this arc, and otherwise, O'_i is moved in the opposite direction. Letting w'_i be the leftmost point of C'_i , we can categorise the Worst Case Circles into the following three mutually exclusive types.

1. Type X_1 : $v_i \neq w'_i$, and $[v_i v_{i+1}]$ does not cross $[st]$, and st is tangent to C'_i .
2. Type X_2 : $v_i = w'_i$ and $[v_i v_{i+1}]$ does not cross $[st]$.
3. Type Y : $v_i = w'_i$ and $[v_i v_{i+1}]$ crosses $[st]$.

Next, we show that the Worst Case Circles of Delaunay triangles A_i fall into the same categories. Let C_i be the circumcircle of A_i centred at O_i , let w_i be the leftmost point of C_i , and let r_i be the right intersection of C_i with $[st]$. We begin with the following observation which follows from how the criteria forces A_i to intersect $[st]$:

► **Observation 9.** Let $A_i = \Delta v_i, p, q$. Taking a clockwise walk along C_i from v_i to r_i , exactly one of p or q is encountered. An analogous statement holds for the counterclockwise walk.

This observation captures the necessary property that allows the categorisation to go through. We denote the Worst Case Circle of A_i by C'_i with centre O'_i , and leftmost point w'_i .

► **Lemma 10.** C'_i can be categorised into the following three mutually exclusive types:

1. Type X_1 : $v_i \neq w'_i$, and $[v_i v_{i+1}]$ does not cross $[st]$, and st is tangent to C'_i .
2. Type X_2 : $v_i = w'_i$ and $[v_i v_{i+1}]$ does not cross $[st]$.
3. Type Y : $v_i = w'_i$ and $[v_i v_{i+1}]$ crosses $[st]$.

Proof. If $[v_i v_{i+1}]$ does not cross $[st]$, C'_i is clearly of type X_1 or X_2 .

Consider when $[v_i v_{i+1}]$ crosses $[st]$. Without loss of generality, let v_i be above $[st]$ and v_{i+1} be below $[st]$. By Observation 9, v_i occurs on the counterclockwise walk around C_i from w_i to r_i , for if not, neither vertex of A_i occurs on the clockwise walk around C_i from v_i to r_i . Since v_i is above $[st]$, it lies above the leftmost intersection of C_i with $[st]$ and below w_i .

Since O'_i is moved along the perpendicular bisector of $[v_i v_{i+1}]$ towards the counterclockwise arc of v_i to v_{i+1} , it must be that w'_i (which starts at w_i when O'_i starts at O_i) moves onto v_i eventually. Thus, C'_i is Type Y . ◀

3.2 Routing on $\mathcal{BDG}(V)$

In order to route on $\mathcal{BDG}(V)$, we simulate the algorithm from the previous section. We first prove a property that allows us to distribute edge information over their endpoints.

► **Lemma 11.** Every edge $uv \in \mathcal{DT}(V)$ is protected by at least one of its endpoints u or v .

Proof. Suppose that uv is not protected at u . Then uv is not extreme at u and thus by Observation 3, u must have consecutive clockwise-ordered Delaunay neighbours v_l, v, v_r . By Lemma 1, $\angle(v_l, v, v_r) \geq \pi - \theta > \theta$ since $0 < \theta < \pi/2$, and thus v_l and v_r cannot both belong to the same cone with apex v and angle at most θ . Since v_r, u, v_l are consecutive clockwise-ordered Delaunay neighbours of v , and vv_l and vv_r cannot be in the same cone, it follows that vu is extreme at v . Hence, uv is protected at v when it is not protected at u . ◀

This lemma allows us to store all edges of the Delaunay triangulation by distributing them over their endpoints. At each vertex u , we store:

1. Fully protected edges uv , with two additional bits to denote whether it is extreme, penultimate, or middle at u .
2. Semi-protected edges uv (only protected at u), with one additional bit denoting whether the clockwise or counterclockwise face path is a spanning path to v .

We denote this augmented version of $\mathcal{BDG}(V)$ as a Marked Bounded Degree Graph or $\mathcal{MBDG}(V)$ for short. There is only a constant overhead to its construction.

► **Theorem 12.** $\mathcal{MBDG}(V)$ stores $O(1)$ words of information at each of its vertices.

Proof. There are at most 5κ edges, each with two additional bits, and 2κ semi-protected edges, each with one additional bit, stored at each vertex, where κ is a fixed constant. ◀

The routing algorithm now works as follows: At a high level, the simulation searches for a suitable candidate triangle A_i at v_i . This is done by taking a walk from v_i along a face to be defined later, ending at some vertex of A_i . Once at a vertex of A_i , we know the locations of the other vertices of this triangle and thus we can make the routing decision and we route to that vertex. Next, we describe how to route on the non-triangular faces of $\mathcal{MBDG}(V)$.

3.2.1 Unguided Face Walks

Suppose vu_1 and vu_m are a middle edge and a penultimate edge in cone C and suppose that vu_1 is the shorter of the two. For any vertex p on this face, we refer to the spanning face path from v to p starting with vu_1 as an Unguided Face Walk from v to p .

In the simulation, we use Unguided Face Walks in a way that p is undetermined until it is reached; we will take an Unguided Face Walk from v and test at each vertex along this walk if it satisfies some property, ending the walk if it does. Routing in this manner from v to p can easily be done locally: Suppose vu_1 was counterclockwise to vu_m . Then, at any intermediate vertex u_i , we take the edge immediately counterclockwise to u_iu_{i-1} ($v = u_0$). The procedure when vu_1 is clockwise to vu_m is analogous.

► **Observation 13.** An Unguided Face Walk needs $O(1)$ memory since at u_i , the previous vertex along the walk u_{i-1} must be stored in order to determine u_{i+1} .

► **Observation 14.** An Unguided Face Walk from v to p has a stretch factor of at most $\max(\pi/2, \pi \sin(\theta/2) + 1)$ by the proof of Theorem 6.

3.2.2 Guided Face Walks

Suppose vp is extreme at v but not protected at p (i.e., it is a semi-protected edge stored at v). Then, vp is a chord of some face determined by pu_1 and pu_m where the former is a middle edge and the latter a penultimate edge. Moreover, recall that we stored a bit with the semi-protected edge vp at v indicating whether to take the edge clockwise or counterclockwise to reach p . We refer to the face path from v to p following the direction pointed to by these bits as the Guided Face Walk from v to p . Routing from v to p can now be done as follows:

1. At v , store p in memory.
2. Until p is reached, if there is an edge to p , take it. Otherwise, take the edge pointed to by the bit of the semi-protected edge to p .

► **Observation 15.** A Guided Face Walk needs $O(1)$ memory since p needs to be stored in memory for the duration of the walk.

► **Observation 16.** A Guided Face Walk from v to p has a stretch factor of at most $\max(\pi/2, \pi \sin(\theta/2) + 1)$ by the proof of Theorem 6.

3.2.3 Simulating the Routing Algorithm

We are now ready to describe the routing algorithm in more detail. First, we consider finding the first vertex after s . If st is an edge, take it and terminate. Otherwise, at $s = v_0$, we consider all edges protected at s , and let su_1 and su_m be the first such edge encountered in a counterclockwise and clockwise sweep starting from $[st]$ centred at s . There are two subcases.

(I) If both su_1 and su_m are not middle edges at s , $\Delta s, u_1, u_m$ is a Delaunay triangle A_0 . Determine whether to route to u_1 or u_m , using the method described at the start of Section 3. If the picked edge is fully protected, we follow it. Otherwise, we take the Guided Face Walk from s to this vertex.

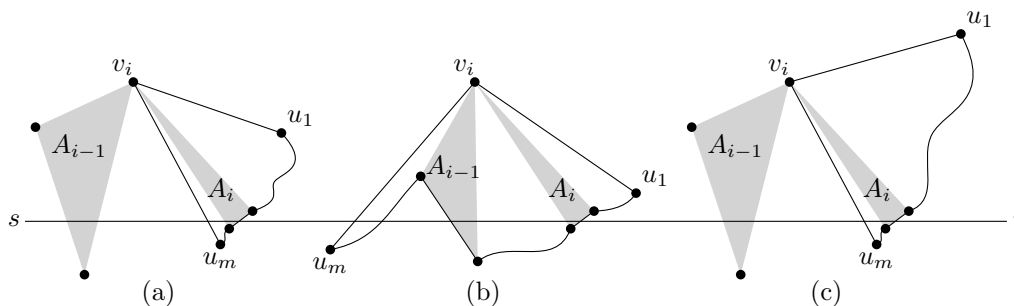
(II) If one of su_1 and su_m is a middle edge at s , the other edge must then be a penultimate edge. Then, $A_0 = \Delta s, p, q$ must be contained in the cone with apex s sweeping clockwise from su_1 to su_m . We assume that su_1 is shorter than su_m . Take the Unguided Face Walk from s until some u_i such that $u_i = p$ is above $[st]$ and $u_{i+1} = q$ is below $[st]$. We have now found $A_0 = \Delta s, p, q$ and we determine whether to route to p or q , using the method described at the start of Section 3.

In both cases, the memory used for the Face Walks is cleared and $A_0 = \Delta s, u_1, u_m$ is stored as the last triangle used.

Next, we focus on how to simulate a routing step from an arbitrary vertex v_i . Suppose v_i is above $[st]$, and that A_{i-1} is stored in memory. If $v_i t$ is an edge, take it and terminate. Otherwise, let $v_i f$ be rightmost edge of A_{i-1} that intersects $[st]$, and $\overline{v_i f}$ be its extension to a line. Make a counterclockwise sweep, centred at v_i and starting at $v_i f$, through all edges that are protected at v_i that lie in the halfplane defined by $\overline{v_i f}$ that contains t . Note that this region must have at least one such edge, since otherwise $v_i f$ is a convex hull edge, which cannot be the case since s and t are on opposite sides.

(I) If there is some edge that does not intersect $[st]$ in this sweep, let $v_i u_1$ be the first such edge encountered in the sweep and let $v_i u_m$ be the protected edge immediately clockwise to $v_i u_1$ at v_i . There are two cases to consider.

(I.I) If A_{i-1} is not contained in the cone with apex v_i sweeping clockwise from $v_i u_1$ to $v_i u_m$ (see Figure 4a), simulating the Delaunay routing algorithm is analogous to the method used for the first step: determine if $v_i u_1$ or $v_i u_m$ is a middle edge and use a Guided or Unguided Face Walk to reach the proper vertex of A_i .



■ **Figure 4** The three cases when simulating a step of the routing algorithm: (a) Case I.I, (b) case I.II, and (c) case II.

(I.II) If A_{i-1} is contained in the cone with apex v_i sweeping clockwise from $v_i u_1$ to $v_i u_m$ (see Figure 4b), then one of $v_i u_1$ and $v_i u_m$ must be a middle edge and the other a penultimate edge, since the edge $v_i f$ is contained in the interior of this cone. Then, $A_i = \Delta v_i, p, q$ must be contained in the cone with apex v_i sweeping clockwise from $v_i u_1$ to $v_i f$.

We take the Unguided Face Walk, starting from the shorter of $v_i u_1$ and $v_i u_m$, stopping when we find some u_i such that $u_i = p$ is above $[st]$ and $u_{i+1} = q$ is below $[st]$, and make the decision to complete the Unguided Face Walk to q or not. Note that when starting from $v_i u_m$ we need to pass f to ensure that A_i lies to the right of A_{i-1} .

(II) If all of the edges in the sweep intersect $[st]$ (see Figure 4c), let $v_i u_m$ be the last edge encountered in the sweep, and $v_i u_1$ be the protected edge immediately counterclockwise to it. Note that A_{i-1} cannot be contained in this cone, as that would imply that $\angle(u_1, v_i, u_m) \geq \pi$, making $v_i u_m$ a convex hull edge. Simulating the Delaunay routing algorithm is analogous to the method used for the first step: determine if $v_i u_1$ or $v_i u_m$ is a middle edge and use a Guided or Unguided Face Walk to reach the proper vertex of A_i .

In all cases, we clear the memory and store $A_i = \Delta v_i, p, q$ as the previous triangle. The case where v_i lies below $[st]$ is analogous. We obtain the following theorem.

► **Theorem 17.** *The routing algorithm on $\mathcal{MBDG}(V)$ is 1-local, has a routing ratio of at most $5.90 \cdot \max(\pi/2, \pi \sin(\theta/2) + 1)$ and uses $O(1)$ memory.*

4 Lightness

In the previous sections we have presented a bounded degree network $\mathcal{MBDG}(V)$ with small spanning ratio that allows for local routing. It remains to show how we can prune this graph even further to guarantee that the resulting network $\mathcal{LMBDG}(V)$ also has low weight.

We will describe a pruning algorithm that takes $\mathcal{MBDG}(V)$ and returns a graph (Light Marked Bounded Degree Graph) $\mathcal{LMBDG}(V) \subseteq \mathcal{MBDG}(V)$, allowing a trade-off between the weight (within a constant times that of the minimum spanning tree of V) and the (still constant) stretch factor. Then, we show how to route on $\mathcal{LMBDG}(V)$ with a constant routing ratio and constant memory.

4.1 The Levcopoulos and Lingas Protocol

To bound the weight of $\mathcal{MBDG}(V)$, we use the algorithm by Levcopoulos and Lingas [10] with two slight modifications: (1) allow any planar graph as input instead of only Delaunay triangulations, and (2) marking the endpoints of pruned edges to facilitate routing.

At a high level, the algorithm works as follows: Given $\mathcal{MBDG}(V)$, we compute its minimum spanning tree and add these edges to $\mathcal{LMBDG}(V)$. We then take an Euler Tour around the minimum spanning tree, treating it as a degenerate polygon P enclosing V . Finally, we start expanding P towards the convex hull $CH(V)$. As edges of $\mathcal{MBDG}(V)$ enter the interior of P , we determine whether to add them to $\mathcal{LMBDG}(V)$. This decision depends on a given parameter $r > 0$. If an edge is excluded from $\mathcal{LMBDG}(V)$, we augment its endpoints with information to facilitate routing should that edge be used in the path found on $\mathcal{MBDG}(V)$. Once P has expanded into $CH(V)$, we return $\mathcal{LMBDG}(V)$.

Before we bound the weight of $\mathcal{LMBDG}(V)$, we need some new notations. An edge in $\mathcal{LMBDG}(V)$ that belongs to the polygon P , or lies in the interior of P , is called an *included settled* edge. If it does not belong to $\mathcal{LMBDG}(V)$, then we say it is an *excluded settled* edge. An edge that has not been processed yet by the algorithm is said to be an *unsettled* edge.

Given an unsettled edge uv , let $\partial P(u, v)$ be the path along P from u to v such that $\partial P(u, v)$ concatenated with uv forms a closed curve that does not intersect the interior of P . When processing an edge uv , it is added to $\mathcal{LMBDG}(V)$ when the summed weight of the edges of $\partial P(u, v)$ is greater than $(1 + 1/r) \cdot |uv|$. This implies that $\mathcal{LMBDG}(V)$ is a spanner.

► **Theorem 18.** *$\mathcal{LMBDG}(V)$ is a $(1 + 1/r)$ -spanner of $\mathcal{MBDG}(V)$ for an adjustable parameter $r > 0$.*

► **Theorem 19.** *$\mathcal{LMBDG}(V)$ has weight at most $(2r + 1)$ times the weight of the minimum spanning tree of $\mathcal{MBDG}(V)$ for an adjustable parameter $r > 0$.*

Proof. Let P be the polygon that encloses V in the above algorithm. Initially P is the degenerate polygon described by the Euler tour of the minimum spanning tree of V in $\mathcal{LMBDG}(V)$. Give each edge e of P , a starting credit of $r|e|$. Denote the sum of credits of edges in P with $\text{credit}(P)$. The sum of $\text{credit}(P)$ and the weight of the initially included settled edges is then $(2r + 1)$ times the weight of the minimum spanning tree of $\mathcal{MBDG}(V)$.

As P is expanded and edges are settled, we adjust the credits in the following manner:

- If an unsettled edge uv is added into $\mathcal{LMBDG}(V)$ when settled, we set the credit of the newly added edge uv of P to $\text{credit}(\partial P(u, v)) - |uv|$, and, by removing $\partial P(u, v)$ from P , we effectively set the credit of edges along $\partial P(u, v)$ to 0.
- If an unsettled edge is excluded from $\mathcal{LMBDG}(V)$ when settled, we set the credit of the newly added edge uv of P to $\text{credit}(\partial P(u, v))$, and, by removing $\partial P(u, v)$, we effectively set the credit of edges along $\partial P(u, v)$ to 0.

We can see that the sum of $\text{credit}(P)$ and the weights of included settled edges, at any time, is at most $2r + 1$ times the weight of the minimum spanning tree of $\mathcal{MBDG}(V)$.

It now suffices to show that $\text{credit}(P)$ is never negative, which we do by showing that for every edge uv of P , at any time, $\text{credit}(uv) \geq r \cdot \text{weight}(uv) \geq 0$. Initially, when P is the Euler Tour around the minimum spanning tree of $\mathcal{MBDG}(V)$, we have that $\text{credit}(uv) = r \cdot \text{weight}(uv)$. We now consider two cases.

(I) If uv is added to $\mathcal{LMBDG}(V)$, then $\text{credit}(uv)$ equals

$$\text{credit}(\partial P(u, v)) - |uv| \geq r \cdot \text{weight}(\partial P(u, v)) - |uv| \geq r(1 + 1/r) |uv| - |uv| = r \cdot \text{weight}(uv).$$

The first inequality holds from the induction hypothesis, and the second inequality and last equality hold since uv is added to $\mathcal{LMBDG}(V)$.

(II) If uv is not added to $\mathcal{LMBDG}(V)$, then $\text{credit}(uv)$ equals

$$\text{credit}(\partial P(u, v)) \geq r \cdot \text{weight}(\partial P(u, v)) = r \cdot \text{weight}(uv).$$

The first inequality holds by induction, and the equality holds since uv was not added.

Since $\text{credit}(P)$ is never negative, and the sum of $\text{credit}(P)$ and the weights of included settled edges is at most $2r + 1$ times the weight of the minimum spanning tree of $\mathcal{MBDG}(V)$, the theorem follows. ◀

Putting together all the results so far, we get:

► **Theorem 20.** *Given a set V of n points in the plane together with two parameters $0 < \theta < \pi/2$ and $r > 0$, one can compute in $O(n \log n)$ time a planar graph $\mathcal{LMBDG}(V)$ that has degree at most $5 \lceil 2\pi/\theta \rceil$, weight of at most $((2r + 1) \cdot \tau)$ times that of a minimum spanning tree of V , and is a $((1 + 1/r) \cdot \tau)$ -spanner of V , where $\tau = 1.998 \cdot \max(\pi/2, \pi \sin(\theta/2) + 1)$.*

Proof. Let us start with the running time. The algorithm by Levcopoulos and Lingas (Lemma 3.3 in [10]) can be implemented in linear time and, according to Corollary 7, $\mathcal{BDG}(V)$ can be constructed in $O(n \log n)$ time, hence, $O(n \log n)$ in total.

The degree bound and planarity follow immediately from the fact that $\mathcal{LMBDG}(V)$ is a subgraph of $\mathcal{MBDG}(V)$, and the bound on the stretch factor follows from Theorem 18.

It only remains to bound the weight. Callahan and Kosaraju [7] showed that the weight of a minimum spanning tree of a Euclidean graph $G(V)$ is at most t times that of the weight of $MST(V)$ whenever G is a t -spanner on V . Since $\mathcal{MBDG}(V)$ is a τ -spanner on V by Corollary 7, $\mathcal{LMBDG}(V)$ has weight of at most $((2r + 1) \cdot \tau)$ times that of the minimum spanning tree of V . This concludes the proof of the theorem. ◀

Finally, we prove that $\mathcal{LMBDG}(V)$ has short paths between the ends of pruned edges.

► **Theorem 21.** *Let uv be an excluded settled edge. There is a face path in $\mathcal{LMBDG}(V)$ from u to v of length at most $(1 + 1/r) \cdot |uv|$.*

Proof. If uv is the first excluded settled edge processed by the Levcopoulos-Lingas algorithm, then all edges of $\partial P(u, v)$ must be included in $\mathcal{LMBDG}(V)$. By planarity, no edge will be added into the interior of the cycle consisting of uv and $\partial P(u, v)$ once uv is settled, and thus uv will be a chord on the face in $\mathcal{LMBDG}(V)$ that coincides with $\partial P(u, v)$. Thus, $\partial P(u, v)$ is a face path in $\mathcal{LMBDG}(V)$ from u to v with a length of at most $weight(uv) \leq (1 + 1/r) \cdot |uv|$.

Otherwise, if uv is an arbitrary excluded edge, then some edges of $\partial P(u, v)$ may be excluded settled edges. If none are excluded, then $\partial P(u, v)$ is again a face path with length at most $weight(uv)$. However, if some edges are excluded, then, by induction, for each excluded edge pq along $\partial P(u, v)$, there is a face path in $\mathcal{LMBDG}(V)$ from p to q with a length of $weight(pq) \leq (1 + 1/r) \cdot |pq|$. Replacing all such pq in $\partial P(u, v)$ by their face paths, and since no edge will be added into the interior of the cycle consisting of uv and $\partial P(u, v)$ once uv is settled, $\partial P(u, v)$ with its excluded edges replaced by their face paths is a face path in $\mathcal{LMBDG}(V)$ from u to v with a length of $weight(uv) \leq (1 + 1/r) \cdot |uv|$. ◀

5 Routing on the Light Graph

In order to route on $\mathcal{LMBDG}(V)$, we store an edge at each of its endpoints when it is excluded. Specifically, let uv be some excluded edge, at u (and v) we store uv , along with one bit to indicate whether the starting edge of the $(1 + 1/r)$ -path is the edge clockwise or counterclockwise to uv .

► **Observation 22.** $\mathcal{LMBDG}(V)$ stores $O(1)$ words of information at each vertex.

To route on $\mathcal{LMBDG}(V)$, we simulate the routing algorithm on $\mathcal{MBDG}(V)$. When this algorithm would follow an excluded edge uv at u , we store v and the orientation of the face path from uv at u in memory. Then, until v is reached, take the edge that is clockwise or counterclockwise to the edge arrived from, in accordance with the orientation stored. Once v is reached, we proceed with the next step of the routing algorithm on $\mathcal{MBDG}(V)$.

Note that bounding the weight in this manner only requires the input graph to be planar. It transforms the pruned edges into $O(d)$ information, where d is the degree of the input graph. This information is then distributed across the vertices of the face, such that each vertex stores $O(1)$ information. The scheme of simulating a particular routing algorithm and switching to a face routing mode when needed can then be applied to the resulting graph.

► **Lemma 23.** *The routing algorithm on $\mathcal{LMBDG}(V)$ is 1-local, has a routing ratio of $5.90(1 + 1/r) \max(\pi/2, \pi \sin(\theta/2) + 1)$ and uses $O(1)$ memory.*

Proof. The 1-locality follows by construction. The routing ratio follows from Theorem 17. Finally, the memory bound follows from the fact that while routing along a face path to get across a pruned edge, no such subpaths can be encountered. Thus, the only additional memory needed at any point in time is a constant amount to navigate a single face path. ◀

6 Conclusion

We showed how to construct and route locally on a bounded-degree lightweight spanner. In order to do this, we simulate the Delaunay routing algorithm by Bonichon et al. [4]. A natural question is whether our routing algorithm can be improved by using the improved Delaunay routing algorithm by Bonichon et al. [3]. Unfortunately, this is not obvious: when applying the improved algorithm on our graph, we noticed that the algorithm can revisit vertices. While this may not be a problem, it implies that the routing ratio proof from [3] needs to be modified in a non-trivial way and thus we leave this as future work.

References

- 1 Patrizio Angelini, Fabrizio Frati, and Luca Grilli. An Algorithm to Construct Greedy Drawings of Triangulations. *Journal of Graph Algorithms and Applications*, 14(1):19–51, 2010.
- 2 Nicolas Bonichon, Prosenjit Bose, Paz Carmi, Irina Kostitsyna, Anna Lubiw, and Sander Verdonschot. Gabriel triangulations and angle-monotone graphs: Local routing and recognition. In *International Symposium on Graph Drawing and Network Visualization (GD)*, pages 519–531, 2016.
- 3 Nicolas Bonichon, Prosenjit Bose, Jean-Lou De Carufel, Vincent Despré, Darryl Hill, and Michiel Smid. Improved routing on the Delaunay triangulation. In *Proceedings of the 26th Annual European Symposium on Algorithms (ESA)*, 2018.
- 4 Nicolas Bonichon, Prosenjit Bose, Jean-Lou De Carufel, Ljubomir Perković, and André Van Renssen. Upper and lower bounds for online routing on Delaunay triangulations. *Discrete & Computational Geometry*, 58(2):482–504, 2017.
- 5 Prosenjit Bose, Rolf Fagerberg, André van Renssen, and Sander Verdonschot. Optimal Local Routing on Delaunay Triangulations Defined by Empty Equilateral Triangles. *SIAM Journal on Computing*, 44(6):1626–1649, 2015.
- 6 Prosenjit Bose and Pat Morin. Online routing in triangulations. *SIAM Journal on Computing*, 33(4):937–951, 2004.
- 7 Paul B. Callahan and S. Rao Kosaraju. Faster algorithms for some geometric graph problems in higher dimensions. In *Proceedings of the 4th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 291–300, 1993.
- 8 Raghavan Dhandapani. Greedy Drawings of Triangulations. *Discrete & Computational Geometry*, 43(2):375–392, 2010.
- 9 Edgar W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1:269–271, 1959.
- 10 Christos Levcopoulos and Andrzej Lingas. There are planar graphs almost as good as the complete graphs and almost as cheap as minimum spanning trees. *Algorithmica*, 8(1-6):251–256, 1992.
- 11 Xiang-Yang Li and Yu Wang. Efficient construction of low weight bounded degree planar spanner. In *Proceedings of the 9th Annual International Computing and Combinatorics Conference (COCOON)*, pages 374–384. Springer, 2003.
- 12 Ge Xia. The stretch factor of the Delaunay triangulation is less than 1.998. *SIAM Journal on Computing*, 42(4):1620–1659, 2013.

Searching for Cryptogenography Upper Bounds via Sum of Square Programming

Dominik Scheder

Shanghai Jiao Tong University, China
dominik@cs.sjtu.edu.cn

Shuyang Tang

Shanghai Jiao Tong University, China
htftsyt@sjtu.edu.cn

Jiaheng Zhang

UC Berkeley, USA
jiaheng_zhang@berkeley.edu

Abstract

Cryptogenography is a secret-leaking game in which one of n players is holding a secret to be leaked. The n players engage in communication as to (1) reveal the secret while (2) keeping the identity of the secret holder as obscure as possible. All communication is public, and no computational hardness assumptions are made, i.e., the setting is purely information theoretic. Brody, Jakobsen, Scheder, and Winkler [2] formally defined this problem, showed that it has an equivalent geometric characterization, and gave upper and lower bounds for the case in which the n players want to leak a single bit. Surprisingly, even the easiest case, where two players want to leak a secret consisting of a single bit, is not completely understood. Doerr and Künnemann [4] showed how to automatically search for good protocols using a computer, thus finding an improved protocol for the 1-bit two-player case. In this work, we show how the search for upper bounds (impossibility results) can be formulated as a Sum of Squares program. We implement this idea for the 1-bit two-player case and significantly improve the previous upper bound from $47/128 = 0.3671875$ to 0.35183 .

2012 ACM Subject Classification Theory of computation → Communication complexity; Theory of computation → Semidefinite programming

Keywords and phrases Communication Complexity, Secret Leaking, Sum of Squares Programming

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.31

Supplement Material <http://basics.sjtu.edu.cn/~dominik/sos-cryptogenography/>

Funding *Dominik Scheder*: This research has been supported by the National Natural Science Foundation of China under grant 61502300.

1 Introduction

Cryptogenography is a whistleblowing problem involving n players. One player $J \in [n]$ is holding a secret $X \in \mathcal{X}$. The value of X is unknown to everybody else. The value of J is unknown, too (the other players just know that they do not have the secret). The goal of the players is to publish the secret while keeping the identity of J as obscure as possible. To achieve this goal, the players engage in public communication according to some publicly known protocol. The protocol ends once the value of the secret has been determined. At this point, the authorities step in and arrest the prime suspect, i.e., the player with the highest a-posteriori probability to be J .

In an (imaginary) application, the secret X could be a compromising document of some government agency or company, J a whistleblower, and $[n]$ a community of transparency activists of which J is a member. A more mundane application sees X as a pirated movie, J



© Dominik Scheder, Shuyang Tang, and Jiaheng Zhang;
licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 31; pp. 31:1–31:12

Leibniz International Proceedings in Informatics



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

as the person sharing the video, and $[n]$ as the users of a file sharing platform. In both cases, the participants want to keep the leaker J anonymous but make the secret X public. This distinguishes it from most other problems in cryptography.

Towards a rigorous formulation, suppose (J, X) are drawn from some *prior distribution* D on $[n] \times \mathcal{X}$. Typically, this will be the uniform distribution, representing a complete lack of knowledge about X and J . A communication protocol is a finite binary tree T , in which every inner node u is labeled with a speaker $P_u \in [n]$ and a vector $(p_i)_{i \in \mathcal{X} \cup \{*\}}$. The semantics of this tree is as follows: we start at the root of the tree. At an inner node u , player P_u sends the next bit of communication. If player P_u is the secret holder, she sends **right** with probability p_X (and **left** with probability $1 - p_X$). If she does not hold the secret, her message cannot depend on X , and thus she sends **right** with probability p_* . The protocol then proceeds to the left or right child of u , depending on the message sent by P_u .

With every node u of the tree we associate a distribution D_u on $[n] \cdot \mathcal{X}$. Namely, $D_u(i, x)$ is the probability that player i holds the secret and the secret is x , conditioned on the protocol arriving at node u . Note that D_{root} is the prior distribution. A protocol is said to be *valid* if X is determined once the protocol ends. In other words, for every leaf l of the protocol tree there is some $x \in \mathcal{X}$ such that $\sum_i D_l(i, x) = 1$. We call x the *output* of leaf l . If the protocol reaches leaf l , an outside observer can be sure that the secret is x . At this point, the authorities arrest the prime suspect, which is the player i maximizing $D_l(i, x)$. The players win if the arrested player is not the secret holder, which happens with probability $1 - \max_{i \in [n]} D_l(i, x)$. The *value* $\text{val}(u)$ of a node u , is the winning probability of the players, conditioned on this node being reached (so $\text{val}(l) = 1 - \max_i D_l(i, x)$ for a leaf l with output x , and $\text{val}(u)$ is a weighted average of $\text{val}(u_0)$ and $\text{val}(u_1)$, where u_0 and u_1 are the two children of u). The value of the protocol is the value of its root (which is equal to the overall winning probability). The value $\text{val}(D)$ of a distribution D is the supremum of $\text{val}(\text{root})$, taken over all valid protocol trees. In words, it is the optimal achievable success probability for the Cryptogenography problem with prior distribution D .

Note that our scenario is purely information theoretical. We assume participants and the authorities to be computationally unlimited. This precludes us from using established methods like public-key cryptography. Also, all communicated is public, and thus methods from multi-party communication (which otherwise could easily solve this problem) do not apply here. In general, we consider cryptogenography as being part of *communication complexity*, and we study it to explore the limitations of randomized two-party (or multi-party) communication.

1.1 Previous Work

The Cryptogenography problem was introduced by Brody, Jakobsen, Scheder, and Winkler [2]. They also chose the name Cryptogenography, which roughly translates to “hidden source writing”. This is because we want to protect the source and not, as would be more common in cryptography, the message. They show that finding the optimal value is equivalent to optimizing a function on the simplex $\Delta_{[n] \times \mathcal{X}}$ (the set of probability distributions on $[n] \times \mathcal{X}$), subject to certain concavity constraints (see Section 2). As for concrete bounds, [2] focus on the case $|\mathcal{X}| = 2$, i.e., the secret consists of a single bit. This might feel a bit unrealistic, as most government documents and most movies consist of more than one bit; however, already the 1-bit case turns out to be challenging. For the n -player case, [2] show that a simple majority voting protocol has a success probability of 0.5406 if $n \geq 23$; a more sophisticated protocol, based on voting with abstention, achieves 0.5644 if $n \geq 1200$. They also prove an upper bound of $\frac{3}{4} - \frac{1}{2n}$. For $n = 2$, the simplest non-trivial case, they show a protocol achieving $1/3$ and an upper bound of $3/8$.

The success probability of $1/3$ for $n = 2$ is achieved by a simple three-round protocol. Somewhat surprisingly, it is not optimal. Doerr and Künnemann [4] showed how Cryptogenography protocol design can be viewed as a certain vector splitting game, and used a computer program to find improved protocols. Their best protocol tree consists of 18248 nodes and achieves a success probability of 0.3384; the shortest protocol (in terms of rounds) they found which beats $1/3$ uses up to sixteen rounds of communication. They also improve the upper bound for $n = 2$ from $3/8$ to $\frac{47}{128}$.

What about the general case of larger n , $|\mathcal{X}| > 2$? Sune Jakobsen [5] showed what the skeptical reader might already suspect: as $|\mathcal{X}|$ increases, the optimal success probability goes to 0, regardless of n . Thus, secure secret leakage is impossible, purely information theoretically speaking. In fact, Jakobsen considered an even more general setting, in which $k = \gamma n$ “insider players” know the secret, and $\mathcal{X} = \{0, 1\}^b$, i.e., the secret has b bits, with $b = \beta n$. He gives a simple and beautiful protocol using error correcting codes and Shannon’s Noisy Channel Coding Theorem (see for example [3]) to show that the players have a reasonable success probability if β is not too large compared to γ . He also shows that this protocol is in some sense asymptotically optimal. Unfortunately, his method give exact results only if γ, β are constants and n tends to infinity and thus do not help us for the case $|\mathcal{X}| = 2$.

Furthermore, Jakobsen and Orlandi [6] introduce a model in which a *almost all* communication is public, and only a very small key is sent through an anonymous channel. They show that with high probability, the secret leaker stays anonymous. However, they assume the adversary to be computationally bounded, and thus their techniques do not apply to our purely information theoretic setting.

1.2 Our Contribution

Like Doerr and Künnemann [4], we focus on the 1-bit 2-player case: $n = 2$ and $\mathcal{X} = \{0, 1\}$. We show how to search for *upper bounds* in an automated way. We use the geometric characterization of [2], which states that to find an upper bound on the possible success probability, it is enough to minimize a function $f : \Delta_{[n] \times \mathcal{X}} \mapsto \mathbb{R}$ subject to certain boundary and concavity constraints. Since $n = 2$ and $\mathcal{X} = \{0, 1\}$, the function f has only four variables. If we let f be a degree- d polynomial in those four variable, this becomes an optimization problem *in the coefficients of f* . We make this optimization problem tractable by formulating it as a Sum-of-Squares problem (SoS), which we then solve using the Matlab packages `yalmip`¹ [9] and `sostools` [10]. The following table summarizes our results obtained with `yalmip`. The numerical error arising from the SoS solver needs careful treatment in our case, and indeed causes degree 8 to be *worse* than degree 6.

Degree	SoS opt	numerical error	SoS opt + error	Remark
2	0.375	0	0.375	Same as [2]
4	0.3644	negligible	0.3644	slightly better than 0.3671875 from [4]
6	0.351612	0.000217	0.35183	our best bound so far
8	0.349515	0.0079587	0.35747	worse than degree 6 (numerical errors)

In theory, our techniques easily extend to the multi-player case and beyond the 1-bit case. However, the complexity increases in a way that makes it quickly impractical for current SoS solvers.

¹ <https://yalmip.github.io/>

2 Geometric Formulation

We now formally introduce the geometric characterization given by Brody, Jakobsen, Scheder, and Winkler [2]. We focus on the two-player 1-bit case, since this already contains all main ideas. Consider the simplex $\Delta := \{\text{Alice}, \text{Bob}\} \times \{0, 1\}$, the space of all probability distributions on who has the secret bit and what its value is. We represent a distribution \mathcal{D} as a matrix:

$$\begin{array}{c|cc} & 0 & 1 \\ \hline \text{Alice} & x_0 & x_1 \\ \text{Bob} & y_0 & y_1 \end{array},$$

or short as $\begin{bmatrix} x_0 & x_1 \\ y_0 & y_1 \end{bmatrix}$. Given a protocol tree, we associate a distribution $D_u \in \Delta$ with every node u : namely, $D_u(j, x)$ is the probability that j is the secret holder and x is the secret, conditioned on the protocol passing through u . Suppose it's Alice's turn to send a bit at node u . Let p be the probability that she sends **right** at node u and denote by D , D' , and D'' the distributions at u , its left child, and its right child, respectively. Then $D = p D' + (1 - p) D''$, so the three distributions

$$\begin{bmatrix} x'_0 & x'_1 \\ y'_0 & y'_1 \end{bmatrix}, \begin{bmatrix} x_0 & x_1 \\ y_0 & y_1 \end{bmatrix}, \begin{bmatrix} x''_0 & x''_1 \\ y''_0 & y''_1 \end{bmatrix}$$

lie on a common line. In fact, this line has a crucial additional property: the line through $[y'_0, y'_1]$, $[y_0, y_1]$, $[y''_0, y''_1]$ also passes through the origin! In other words, the ratio between y_0 and y_1 is the same for all three distributions. This is because Alice can give hints about (1) whether she has the secret and (2) what its value is *provided she owns it* but *not* what the value is *if Bob has it*. This property is formally proven in [2]. The upshot is that the line through D, D', D'' can be parametrized as

$$\ell : t \mapsto \begin{bmatrix} x_0 & x_1 \\ y_0 & y_1 \end{bmatrix} + t \begin{bmatrix} r_0 & r_1 \\ s_0 & s_1 \end{bmatrix} \quad (1)$$

where

$$x_0 + x_1 + y_0 + y_1 = 1 \quad (2)$$

$$x_0, x_1, y_0, y_1 \geq 0 \quad (3)$$

$$r_0 + r_1 + s_0 + s_1 = 0 \quad (4)$$

$$s_0 y_1 - s_1 y_0 = 0 \quad (5)$$

$$s_0 s_1 \geq 0. \quad (6)$$

Note that (2) and (3) simply state that D is a distribution; (4) must hold since the line contains two other distributions D' and D'' , too. Equality (5) states that the line $[y_0 \ y_1] + t[s_0 \ s_1]$ contains the origin. Finally, (6) follows from the other constraints and is included only because it turns out to help the SoS solver.

A line of the form (1) satisfying (2)–(6) is an *Alice-line*. If it's Bob's turn to talk at u , then equality (5) and (6) and should be replaced by $r_0 x_1 - r_1 x_0 = 0$ and $r_0 r_1 \geq 0$; we call such a line a *Bob-line*. A *player-line* is a line that is an Alice-line or a Bob-line. A player-line is *non-trivial* if it intersects Δ in more than one point.

To wrap up, if Alice talks at node u , she “splits” the distribution D into D' and D'' such that all three distributions lie on an Alice-line. A certain converse of this actually holds, too: if D is the distribution associated with node u , and D lies between the distributions D' and D'' on a common Alice-line, then there is a way for Alice to sample her message (depending on whether she has the secret and if yes, what it is), such that the two children of u will be labeled with D' and D'' , respectively. The (straightforward) proof can be found in [2].

► **Definition 1.** A function $f : \Delta \rightarrow \mathbb{R}$ is called *admissible* if

(a) $f(D) \geq 0$ for $D = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}$, and $\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$;

(b) $f(D) \geq 1/2$ for $D = \begin{pmatrix} 1/2 & 0 \\ 1/2 & 0 \end{pmatrix}$ and $\begin{pmatrix} 0 & 1/2 \\ 0 & 1/2 \end{pmatrix}$;

(c) If ℓ is a non-trivial player-line, then $f|_{\ell}$, the restriction of f to ℓ , is concave at every point in Δ .

The following theorem gives a geometric characterization of the problem. It is an adaptation of Lemma 4.3 from [2], together with the additional observation, made in [4], that the six “boundary distributions” in Point (a) and Point (b) actually suffice.

► **Theorem 2** (adapted from [2] and [4]). *If f is an admissible function, then $f(D) \geq \text{val}(D)$ for all distributions D .*

The proof of the theorem is simple and works by induction on the protocol tree. Point (a) and Point (b) of Definition 1 serve as the induction base, and concavity along player-lines (c) is used in the induction step.

A similar geometric characterization has been used by Mark Braverman, Ankit Garg, Denis Pankratov, and Omri Weinstein [1] to determine the information complexity of zero-error protocols computing the 2-bit AND function, and subsequently determining the asymptotic communication complexity of DISJOINTNESS, one of the most important functions in communication complexity.

► **Lemma 3.** *A function $f : \Delta \rightarrow \mathbb{R}$ is concave along every Alice-line (respectively, Bob-line), if and only if $\frac{\partial^2}{\partial t^2} f(\ell(t))|_{t=0} \leq 0$ for every Alice-line ℓ as in (1) (respectively, Bob-line).*

Proof. For fixed $x_0, x_1, y_0, y_1, r_0, r_1, s_0, s_1$ the function $g(t) = f(\ell(t))$ is a univariate polynomial in t . For the “if” part, note that if this is not concave for some t^* such that $\ell(t^*) \in \Delta$, then $g''(t^*) > 0$; furthermore, we can re-parametrize the line as $\ell(t^* + t)$; this is also an Alice line (resp., Bob-line), and concavity is violated at $t = 0$. For the “only if” part, note that concavity of $f|_{\ell}$ implies that $g''(0) \leq 0$. ◀

3 Sum of Square Programs

In this section we introduce the absolute basics of Sum of Square (SoS) Programs. SoS programming is a rapidly evolving field with vast literature but unfortunately no standard textbook yet. The reader interested in actually solving SoS programs may have a look at the SOSTOOLS User’s guide [11]. If the reader is more interested in the theoretical background of SoS programming, they might start with Laurent [8].

How do you show that $P(x, y, z) = (x^2 + y)^2(1 + z^2) - 4xyz$ is non-negative? For example, you could check that $P(x, y, z) = (x - yz)^2 + (y - xz)^2$. In other words, you write it as a *sum of squares* polynomial (SoS). Next, consider $Q(x, y) = 1 - x^2y^2 - 2x^2y$. How can one show that $Q \geq 0$ on the unit disk? For example, by writing

$$Q(x, y) = (x^2 - y)^2 + (1 + x^2)(1 - x^2 - y^2).$$

Indeed, $(x^2 - y)^2$ and $1 + x^2$ are SoS, and $1 - x^2 - y^2$ is non-negative on the unit disk. Is this method complete? No: there are non-negative polynomials that are not SoS. Is it correct? Obviously. Even better, it is “tractable”, in a certain sense. Namely, while deciding “Is $P(\mathbf{x}) \geq 0$?” is NP-hard, the question “Is $P(\mathbf{x})$ a SoS” can be translated into a semi-definite program and then efficiently solved (at least up to arbitrary precision). This is the basic idea behind sum-of-square programs. In its full generality, however, it allows us much more. Namely, each line of a SoS program can be

1. A polynomial declaration, like “ P is a polynomial of degree 6 in variables u, w, x, z ”. From the SoS program’s point of view, the coefficients of P are *variables*. We call them *decision variables* since the SoS solver will decide what values to assign to them; the “true” variables u, w, x, z , etc. will be called *polynomial variables*.
2. A linear constraint in the decision variables.
3. A statement like “ P is a Sum of Square polynomial”.
4. A target function “minimize T ”, where T is a linear expression in the decision variables.

Of course, there can be only one target function.

Thus, not only can we ask the SoS solver check that $P = (x^2 + y)^2(1 + z^2) - 4xyz$ is non-negative; we can also use it to find the maximum w for which $(x^2 + y)^2(1 + z^2) - wxyz$ is non-negative. There is no guarantee that it finds the maximum such value w ; instead, it finds the maximum w for which $(x^2 + y)^2(1 + z^2) - wxyz$ is a SoS. Next, set $Q_w(x, y, z) = 1 - x^2y^2 - wx^2y$. We have seen above that Q_2 is non-negative on the unit disk. Can we determine the maximum w for which Q_w is non-negative on the unit disk? Again, the SoS framework offers no such guarantee, but we can write the following SoS program:

$$\begin{aligned} &\text{maximize} && w \\ &\text{subject to} && Q_w = P_1 + P_2(1 - x^2 - y^2) \\ &&& P_1, P_2 \text{ are SoS polynomials of degree } d_1 \text{ and } d_2, \text{ respectively.} \end{aligned}$$

Note that the constraint $Q_w = P_1 + P_2(1 - x^2 - y^2)$ is *linear in the coefficients of the polynomials*. Also, it is not clear a priori what degree d we should choose. As a heuristic, we could choose $d_1 = 4$ and $d_2 = 2$ to make sure that every summand on the right-hand side has degree 4, which is the degree of the left-hand side. If we are lucky, this returns the optimal w . In any case, the result will be a lower bound on the optimum: if P_1, P_2 are SoS and $Q_w = P_1 + P_2(1 - x^2 - y^2)$, then surely $Q_w \geq 0$ on the unit disk.

Let us now try to formulate our hunt for Cryptogenography upper bounds as a SoS program. Referring to Definition 1 and Theorem 2, we define f to be a polynomial of degree d in the variables $U = \{x_0, x_1, y_0, y_1\}$ with unknown coefficients. Point (a) and (b) of the definition are linear in the coefficients. How do we write (c) as a SoS constraint? First, define

$$g := - \left. \frac{\partial^2 f(\ell(t))}{\partial t^2} \right|_{t=0} .$$

This is a polynomial in the variables $V := U \cup \{r_0, r_1, s_0, s_1\}$. We could add the constraint “ g is a SoS” into our program, but this requirement is way too strong. Indeed, g need be non-negative only for those values of V that constitute a player-line, i.e., that satisfy (2)–(6). We define $b_1 := x_0, b_2 := x_1, b_3 := y_0, b_4 := y_1, b_5 := s_0 s_1, c_1 := x_0 + x_1 + y_0 + y_1 - 1$, and $c_2 := r_0 + r_1 + s_0 + s_1$, and $c_3 := y_0 s_1 - y_1 s_0$. Note that (2)–(6) are satisfied if and only if $b_1, \dots, b_5 \geq 0$ and $c_1 = c_2 = c_3 = 0$. We add the following lines to our SoS program:

$$g = c_1 \cdot R_1 + c_2 \cdot R_2 + c_3 \cdot R_3 + \sum_{I \subseteq [5]} Q_I \cdot \prod_{i \in I} b_i , \tag{7}$$

each Q_I is a SoS polynomials of degree d_I ,
 R_1, R_2, R_3 are arbitrary polynomials of degree d_R .

If g is as in (7), then $g \geq 0$ for all values of V constituting an Alice-line, and thus f is concave along all such lines. Obviously, we should add a similar constraint for Bob-lines. For efficiency reason we don't. Instead, we will exploit some inherent symmetries in the problem, as we explain in the next paragraph. Finally, we define the target function of our SoS program: “minimize $f(1/4, 1/4, 1/4, 1/4)$ ”, which is a linear function in the coefficients of f .

3.1 Exploiting Symmetries

The Cryptogenography problem has two fundamental symmetries: we can switch the roles of Alice and Bob, and we can switch 0 and 1, all without changing the optimal value of a protocol for the uniform prior distribution. Suppose our generic admissible function f is a polynomial of degree d , namely $f(\mathbf{x}) = \sum_{\mathbf{a} \in \mathbb{N}_0^4: |\mathbf{a}|_1 \leq d} w_{\mathbf{a}} \mathbf{x}^{\mathbf{a}}$, where $\mathbf{x} = (x_0, x_1, y_0, y_1)$. By symmetry, we can require that the $w_{a,b,c,d} = w_{b,a,d,c} = w_{c,d,a,b} = w_{d,c,b,a}$ for all $a, b, c, d \in \mathbb{N}_0$. Formally, for $\mathbf{a} = (a, b, c, d) \in \mathbb{N}_0^4$, let $\bar{\mathbf{a}}$ denote the lexicographically first among (a, b, c, d) , (b, a, d, c) , (c, d, a, b) , (d, c, b, a) . Thus, we can write f as

$$f(\mathbf{x}) = \sum_{\mathbf{a} \in \mathbb{N}_0^4: |\mathbf{a}|_1 \leq d} w_{\bar{\mathbf{a}}} \mathbf{x}^{\mathbf{a}}. \tag{8}$$

If such a n f is concave along Alice-lines, it is also concave along Bob-lines, by symmetry. Thus, our SoS program only needs to contain a constraint for Alice-lines. As an additional bonus, formulating Point (a) and Point (b) of Definition 1 only takes two constraints rather than six. This basically describes our SoS program.

3.2 Obvious Optimizations

Note that (2) states $x_0 + x_1 + y_0 + y_1 = 1$ and (4) states that $r_0 + r_1 + s_0 + s_1 = 0$. Thus, instead of adding $c_1 \cdot R_1 + c_2 \cdot R_2$ to the right-hand side of (7), we can “by hand” substitute $x_1 = 1 - x_0 - y_0 - y_1$ and $r_1 = -r_0 - s_0 - s_1$. This reduces the number of polynomial variables from 8 to 6, which tremendously improves running time. The only downside is that our SoS program becomes a bit uglier. Putting everything together, here is the pseudocode of our SoS program:

$f(\mathbf{x}) := \sum_{\mathbf{a} \in \mathbb{N}_0^4: \mathbf{a} _1 \leq d} w_{\bar{\mathbf{a}}} \mathbf{x}^{\mathbf{a}}.$
minimize $f(1/4, 1/4, 1/4, 1/4)$ subject to $f(1, 0, 0, 0) \geq 0$ subject to $f(1/2, 0, 1/2, 0) \geq 1/2$
$g(x_0, x_1, y_0, y_1, r_0, r_1, s_0, s_1) := - \left. \frac{\partial^2 f(\ell(t))}{\partial t^2} \right _{t=0}$ $h(x_0, y_0, y_1, r_0, s_0, s_1) := g(x_0, 1 - x_0 - y_0 - y_1, y_0, y_1, r_0, -r_0 - s_0 - s_1, s_0, s_1)$ $b_1 := x_0, b_2 := 1 - x_0 - y_0 - y_1, b_3 := y_0, b_4 := y_1, b_5 := s_0, s_1, c := y_0 s_1 - y_1 s_0$
subject to $h = \sum_{I \subseteq [5]} Q_I \cdot \prod_{i \in I} b_i + R \cdot c$ subject to each Q_I is a SoS polynomial in x_0, y_0, y_1, r_0 of degree d_I , R is an arbitrary polynomial of degree d_R

It remains to specify the degree d_I of each SoS polynomial Q_I . We heuristically set d_I to $d - |I|$ or $d - |I| - 1$, whatever is even, and set $d_R := d - 2$. The intuition is that (a) the degree of the SoS polynomial Q_I must be even and (b) the degree of each summand on the right-hand side should be at most d , the degree of the left-hand side.

To implement the SoS program, we use the matlab package `yalmip` [9], since it turned out to run significantly faster on our problems than `sostools` [10]. In the “exploration phase” of this work we mostly used `sostools`, which might be a bit easier to work with for the non-experienced user. We wrote a `python` program that takes a degree d as input and outputs the `yalmip` code for our SoS program. In particular, it computes $f(\mathbf{x}) = \sum_{\mathbf{a} \in \mathbb{N}_0^4: |\mathbf{a}|_1 \leq d} w_{\mathbf{a}} \mathbf{x}^{\mathbf{a}}$, taking care of all symmetries; the remaining `yalmip` code is the same for each d .

4 Handling Numerical Errors

We are solving a SoS program on a computer; this uses a numerical algorithm, so the result will contain some numerical errors. Inevitably so: SoS solvers are basically SDP solvers plus syntactic sugar, and some semi-definite programs only have irrational solutions.

In particular, the equation $h = \sum_{I \subseteq [5]} Q_I \cdot \prod_{i \in I} b_i + R \cdot c$ will only hold approximately. At least we can be sure that the Q_I on the right are SoS polynomials – we can ask the SoS solver to give us the SoS decomposition $p_1^2 + p_2^2 + \dots + p_k^2$ and then simply define our Q_I to be this sum, and let $\tilde{h} := \sum_{I \subseteq [5]} Q_I \cdot \prod_{i \in I} b_i + R \cdot c$. However, $h = \tilde{h}$ will hold only approximately. Formally, $E := \tilde{h} - h$ is some non-zero polynomial with very small coefficients. By design, $\tilde{h} \geq 0$ for all values constituting an Alice-line; however, h might attain small negative values. If we had an “approximate version” of Theorem 2, stating that if f is “almost concave” along player-lines then it is “almost an upper bound”, we would be done. Unfortunately, we do not have that, so we have to come up with a manual work-around. Let f be tentative upper bound function as found by the SoS solver. For some suitable $\epsilon > 0$, define

$$\hat{f}(\mathbf{x}) = f(\mathbf{x}) - \frac{\epsilon}{2} (x_0^2 + x_1^2 + y_0^2 + y_1^2) + \epsilon.$$

If f satisfies the boundary conditions up to error $\epsilon/2$, that is, if $f(1, 0, 0, 0) \geq -\epsilon/2$ and $f(1/2, 0, 1/2, 0) \geq 1/2 - \epsilon/2$, then \hat{f} does indeed satisfy them. Next, we want to show that \hat{f} satisfies the concavity constraints. By re-parametrizing the equation of a generic Alice-line (1), we can assume that $r_0^2 + r_1^2 + s_0^2 + s_1^2 = 1$. Put differently, one observes that g is homogenous of degree 2 in each of the variables r_0, r_1, s_0, s_1 . So it is non-negative for all inputs in S if and only if it is for all inputs with $r_0^2 + r_1^2 + s_0^2 + s_1^2 = 1$. Consider the polynomials \hat{g} and \hat{h} , defined analogous to g and h , but starting with \hat{f} instead of f . A brief calculation shows that $\hat{g} = g + \epsilon \cdot (r_0^2 + r_1^2 + s_0^2 + s_1^2) = g + \epsilon$ and thus $\hat{h} = h + \epsilon$. Thus, on any input $(\mathbf{x}, \mathbf{r}) \in S$ with $\|\mathbf{r}\|_2 = 1$, we have $h(\mathbf{x}, \mathbf{r}) = \hat{h}(\mathbf{x}, \mathbf{r}) - E(\mathbf{x}, \mathbf{r}) + \epsilon \geq \epsilon - E(\mathbf{x}, \mathbf{r})$. We give a very crude upper bound on $E(\mathbf{x}, \mathbf{r})$: since all input variables are at most 1 in absolute value, $E(\mathbf{x}, \mathbf{r})$ is at most the sum of all coefficients. Let us denote this quantity by $\|E\|_1$. Thus, as long as $\epsilon \geq \|E\|_1$, we can be sure that \hat{f} is indeed admissible, and thus an upper bound. That is, $\text{val}(1/4, 1/4, 1/4, 1/4) \leq \hat{f}(1/4, 1/4, 1/4, 1/4) \leq f(1/4, 1/4, 1/4, 1/4) + \|E\|_1$. The table in Section 1.2 sums up the results for degree 2, 4, 6, 8. The column labeled “numerical error” is our upper bound $\|E\|_1$.

One could attempt to further reduce the number of variables. Indeed, equality (5) states $y_0 s_1 = y_1 s_0$. Using this, we could scale (1) to ensure that $s_0 = y_0$ and $s_1 = y_1$. This reduces the number of variables from six to four, and vastly reduces the running time of `yalmip`. However, we cannot simultaneously ensure that $\|\mathbf{r}\|_2 = 1$ and thus do not know how to handle the resulting numerical error. It would still be interesting to see what this gives for degree 10 and 12, but it would require a leap of faith to conclude that the bound thus computed really holds.

5 Open Questions

Is there a better way to handle numerical errors? If so, then maybe one *can* remove the number of variables to four and explore higher degrees. As for theoretical aspects, can one prove that as $d \rightarrow \infty$, the optimal value of the SoS program converges to the actual optimum of the cryptogenography problem? It is known (see for example Lasserre [7]) that any non-negative $f : [-1, 1]^m \rightarrow \mathbb{R}$ can be arbitrarily well approximated by SoS polynomials. However, we don't require the polynomial g to be non-negative everywhere; only non-negative on a certain set. It is not clear to me whether the approximation result transfers to this setting.

6 Source Code and Data

Since already for degree four, the number of monomials in f and the polynomials Q_I are too large to be comfortably printed in an article, we created the webpage

<http://basics.sjtu.edu.cn/~dominik/sos-cryptogenography/>

to which we uploaded all source code and all output data that enables the reader to verify our results without having to run `yalmip` or any SoS solver.

References

- 1 Mark Braverman, Ankit Garg, Denis Pankratov, and Omri Weinstein. From information to exact communication. 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 151–160. ACM, 2013. doi:10.1145/2488608.2488628.
- 2 Joshua Brody, Sune K. Jakobsen, Dominik Scheder, and Peter Winkler. Cryptogenography. In Moni Naor, editor, *Innovations in Theoretical Computer Science, ITCS'14, Princeton, NJ, USA, January 12-14, 2014*, pages 13–22. ACM, 2014. doi:10.1145/2554797.2554800.
- 3 Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, New York, NY, USA, 2006.
- 4 Benjamin Doerr and Marvin Künnemann. Improved Protocols and Hardness Results for the Two-Player Cryptogenography Problem. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPICs*, pages 150:1–150:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPICs.ICALP.2016.150.
- 5 Sune K. Jakobsen. Information Theoretical Cryptogenography. *J. Cryptology*, 30(4):1067–1115, 2017. doi:10.1007/s00145-016-9242-8.
- 6 Sune K. Jakobsen and Claudio Orlandi. How To Bootstrap Anonymous Communication. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, ITCS '16*, pages 333–344, New York, NY, USA, 2016. ACM. doi:10.1145/2840728.2840743.
- 7 Jean B. Lasserre. A Sum of Squares Approximation of Nonnegative Polynomials. *SIAM J. on Optimization*, 16(3):751–765, March 2006. doi:10.1137/04061413X.
- 8 Monique Laurent. Sums of Squares, Moment Matrices and Optimization Over Polynomials. In Mihai Putinar and Seth Sullivant, editors, *Emerging Applications of Algebraic Geometry*, pages 157–270. Springer New York, New York, NY, 2009. doi:10.1007/978-0-387-09686-5_7.
- 9 J. Löfberg. YALMIP : A Toolbox for Modeling and Optimization in MATLAB. In *In Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004.
- 10 A. Papachristodoulou, J. Anderson, G. Valmorbida, S. Prajna, P. Seiler, and P. A. Parrilo. *SOSTOOLS: Sum of squares optimization toolbox for MATLAB*, 2013. Available from <http://www.eng.ox.ac.uk/control/sostools>, <http://www.cds.caltech.edu/sostools> and <http://www.mit.edu/~parrilo/sostools>. arXiv:1310.4716.

31:10 Cryptogenography Upper Bounds via SoS Programming

- 11 Antonis Papachristodoulou, James Anderson, Giorgio Valmorbida, Stephen Prajna, Pete Seiler, and Pablo A. Parrilo. SOSTOOLS version 3.00 sum of squares optimization toolbox for MATLAB. *CoRR*, abs/1310.4716, 2013. arXiv:1310.4716.

A The yalmip Code for Degree 4

```
clear
yalmip('clear')
tic
echo on

%% begin python generated code
degree = 4;
sdpvar a b c d r0 s0 s1 t w0000 w1000 w2000 w1100 w1010 w1001 w3000
w2100 w2010 w2001 w1110 w4000 w3100
w3010 w3001 w2200 w2110 w2101 w2020 w2011 w2002 w1111;
fvars = [w0000; w1000; w2000; w1100; w1010; w1001; w3000; w2100;
w2010; w2001; w1110; w4000; w3100; w3010;
w3001; w2200; w2110; w2101; w2020; w2011; w2002; w1111];
decisionvars = fvars;
f = w0000*1 + w1000*a + w1000*b + w1000*c + w1000*d + w2000*a^2 + w1100*a*b +
w1010*a*c + w1001*a*d + w2000*b^2 + w1001*b*c + w1010*b*d + w2000*c^2 + w1100*c*d +
w2000*d^2 + w3000*a^3 + w2100*a^2*b + w2010*a^2*c + w2001*a^2*d + w2100*a*b^2 +
w1110*a*b*c + w1110*a*b*d + w2010*a*c^2 + w1110*a*c*d + w2001*a*d^2 + w3000*b^3 +
w2001*b^2*c + w2010*b^2*d + w2001*b*c^2 + w1110*b*c*d + w2010*b*d^2 + w3000*c^3 +
w2100*c^2*d + w2100*c*d^2 + w3000*d^3 + w4000*a^4 + w3100*a^3*b + w3010*a^3*c +
w3001*a^3*d + w2200*a^2*b^2 + w2110*a^2*b*c + w2101*a^2*b*d + w2020*a^2*c^2 +
w2011*a^2*c*d + w2002*a^2*d^2 + w3100*a*b^3 + w2101*a*b^2*c + w2110*a*b^2*d +
w2011*a*b*c^2 + w1111*a*b*c*d + w2011*a*b*d^2 + w3010*a*c^3 + w2110*a*c^2*d +
w2101*a*c*d^2 + w3001*a*d^3 + w4000*b^4 + w3001*b^3*c + w3010*b^3*d +
w2002*b^2*c^2 + w2011*b^2*c*d + w2020*b^2*d^2 + w3001*b*c^3 + w2101*b*c^2*d +
w2110*b*c*d^2 + w3010*b*d^3 + w4000*c^4 + w3100*c^3*d + w2200*c^2*d^2 +
w3100*c*d^3 + w4000*d^4;
%% end python generated code

B = 1-a-c-d;
R1 = -r0-s0-s1;
f_replace = replace(f, [a,b,c,d], [a + t*r0, B + t*R1, c + t*s0, d + t*s1])
h = -replace(jacobian(jacobian(f_replace ,t), t), [t], [0])

corner = replace(f, [a,b,c,d], [1,0,0,0]);
edge = replace(f, [a,b,c,d], [1/2,0,1/2,0]);

Constraintsp0 = [corner == 0, edge == 1/2];
Constraintsp = [];

%% concavity constraint for Alice:

%% Our set S of feasible inputs is defined by five polynomial inequalities
%% and one equality
%% The five inequalities are a, B, c, d, s0*s1 >= 0
%% Each subset of [5] gives us a 'region polynomial', like a*B*s0*s1
```



```

%% For technical reasons we only list the products over non-empty subsets:

region_vector = [a, B, c, d, a*B, a*c, a*d, B*C, B*d, c*d, a*B*c, a*B*d, a*c*d,
B*c*d, a*B*c*d, s0*s1, a*s0*s1, B*s0*s1, c*s0*s1, d*s0*s1, a*B*s0*s1,
a*c*s0*s1, a*d*s0*s1, B*c*s0*s1, B*d*s0*s1, c*d*s0*s1, a*B*c*s0*s1,
a*B*d*s0*s1, a*c*d*s0*s1, B*c*d*s0*s1, a*B*c*d*s0*s1];

%% This is simply a list containing the degrees of the polynomials just defined

deg_of_boundaries = [1,1,1,1,2,2,2,2,2,2,3,3,3,3,4,2,3,3,3,3,4,4,4,4,4,4,5,5,5,5,6];

%% We now define our polynomials Q_I for each (non-empty) subset I of [5]
%% Each Q_I is in the variables a,c,d,s0, s1, r0:

polyvars = [a,c,d, s0, s1, r0];
boundary_polys = [];
for i=1:length(region_vector)

    % Creating the polynomial Q_I
    % remember: the degree of Q_I * prod_{i in I} b_i should be at most <degree>,
    % and the degree of Q_I must be even

    degree_here = max(0,2*floor((degree - deg_of_boundaries(i))/2));

    % We ask yalmip to create a new polynomial Q_I
    [Q_I, coeff] = polynomial(polyvars, degree_here);
    boundary_polys = [boundary_polys; Q_I];

    % this Q_I must be SoS, so add a SoS constraint
    Constraintsp = [Constraintsp, sos(Q_I)];

    % tell yalmip that we created a new bunch of decision variables:
    % the coefficients of Q_I
    decisionvars = [decisionvars; coeff];
end

% In the paper we had the (not necessarily SoS) polynomial R
% which enters the SoS program as R*(y0*s1 - y1*s0)
% Here, we use c,d instead of y0, y1 and, for clarity
% 'alice_line_poly' instead of 'R'

[alice_line_poly, alice_line_poly_coeff] = polynomial(polyvars, degree-2);
decisionvars = [decisionvars; alice_line_poly_coeff];

[normalization_poly, normalization_poly_coeff] = polynomial(polyvars, degree-2);
decisionvars = [decisionvars; normalization_poly_coeff];

Constraints = [sos(h - region_vector * boundary_polys ...
    - (c*s1 - d*s0) * alice_line_poly ...
    - (1 - r0^2 - R1^2 - s0^2 - s1^2) * normalization_poly), Constraintsp];
Constraints = [Constraintsp0, Constraints];

target = replace(f, [a,b,c,d], [1/4,1/4,1/4,1/4]);

```

31:12 Cryptogenography Upper Bounds via SoS Programming

```
ops = sdpsettings('solver','sedumi','sedumi.eps',1e-14);

%% finally solving the SoS program

solvesos(Constraints, target, [ops] , decisionvars);

%% assembling the right-hand side of h, i.e., what we call \tilde{h} in the paper

sdpvar Q0_sos_polys Qi_sos_poly sum_poly error_poly error_poly_sym

sum_poly = 0;

Q0_sos_polys = sosd(Constraints(3));
for i=1:length(Q0_sos_polys)
    sum_poly = sum_poly + Q0_sos_polys(i)^2;
end

for k=1:length(region_vector)
    Qi_sos_polys = sosd(Constraints(k+3));
    for i=1:length(Qi_sos_polys)
        sum_poly = sum_poly + region_vector(k)*(Qi_sos_polys(i)^2);
    end
end

sum_poly = sum_poly ...
    + (c*s1 - d*s0) * replace(alice_line_poly, decisionvars, value(decisionvars))
    + (1 - r0^2 - R1^2 - s0^2 - s1^2) *
    replace(normalization_poly, decisionvars, value(decisionvars));

error_poly = sum_poly - replace(h, decisionvars, value(decisionvars));

error_poly_sym = str2sym( sdisplay(error_poly) );

syms vec_error vec_sum;

vec_error = coeffs(error_poly_sym);

total_error = 0;
for i=1:length(vec_error)
    total_error = total_error + abs(vec_error(i));
end

sdisplay(replace(target,decisionvars,value(decisionvars)))
total_error

toc
```

On the Complexity of Lattice Puzzles

Yasuaki Kobayashi

Graduate School of Informatics, Kyoto University, Yoshida-honmachi, Sakyo-ku,
Kyoto 606-8501 Japan
kobayashi@iip.ist.i.kyoto-u.ac.jp

Koki Suetsugu 

National Institute of Informatics, Hitotsubashi, Chiyoda-ku, Tokyo 101-8430 Japan
suetsugu.koki@gmail.com

Hideki Tsuiki

Graduate School of Human and Environmental Studies, Kyoto University, Yoshida Nihonmatsu-cho,
Sakyo-ku, Kyoto 606-8501 Japan
tsuiki.hideki.8e@kyoto-u.ac.jp

Ryuhei Uehara 

School of Information Science, Japan Advanced Institute of Science and Technology, Asahidai,
Nomi, Ishikawa 923-1292, Japan
uehara@jaist.ac.jp

Abstract

In this paper, we investigate the computational complexity of lattice puzzle, which is one of the traditional puzzles. A lattice puzzle consists of $2n$ plates with some slits, and the goal of this puzzle is to assemble them to form a lattice of size $n \times n$. It has a long history in the puzzle society; however, there is no known research from the viewpoint of theoretical computer science. This puzzle has some natural variants, and they characterize representative computational complexity classes in the class NP. Especially, one of the natural variants gives a characterization of the graph isomorphism problem. That is, the variant is GI-complete in general. As far as the authors know, this is the first non-trivial GI-complete problem characterized by a classic puzzle. Like the sliding block puzzles, this simple puzzle can be used to characterize several representative computational complexity classes. That is, it gives us new insight of these computational complexity classes.

2012 ACM Subject Classification Theory of computation \rightarrow Problems, reductions and completeness

Keywords and phrases Lattice puzzle, NP-completeness, GI-completeness, FPT algorithm

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.32

Related Version An early version was presented orally at JCDCGGG 2017 http://www.jcdcgg.u-tokai.ac.jp/past_conferences/2017.html.

Funding This work is partially supported by KAKENHI grant numbers 15K00015, 17H06287, and 18H04091, and JST CREST JPMJCR1401.

Acknowledgements The authors thank Prof. Shuji Yamada for his introduction of his original designed lattice puzzle to the first three authors. The authors also thank JCDCGGG 2017 for giving a chance to give an oral presentation of the early work on this topic, which motivated the last author to join this research. The last author thanks Hisayoshi Akiyama and Mineyuki Uyematsu who kindly taught him the history of the lattice puzzle.

1 Introduction

In history of theoretical computer science, some puzzles and games play important roles for giving reasonable characterization to computational complexity classes. For example, Conway's game of life is universal [2], and it essentially has the same computational power of the Turing machine. Later, "pebble game" was proposed as a classic model that gives some



© Yasuaki Kobayashi, Koki Suetsugu, Hideki Tsuiki, and Ryuhei Uehara;
licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 32; pp. 32:1–32:12

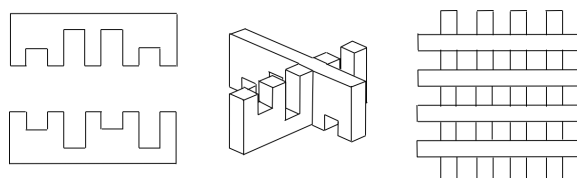
Leibniz International Proceedings in Informatics



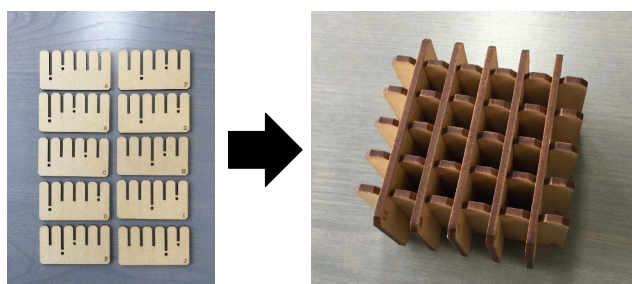
LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

32:2 On the Complexity of Lattice Puzzles

complexity classes in a natural way (see, e.g., [6]). The “constraint logic” is a recent model that succeeds to solve a long standing open problem due to Martin Gardner that asks the computational complexity of sliding block puzzles [5]. Such puzzles and games have been giving us some intuitive insight for some computational complexity classes.



■ **Figure 1** Illustration of a lattice puzzle.

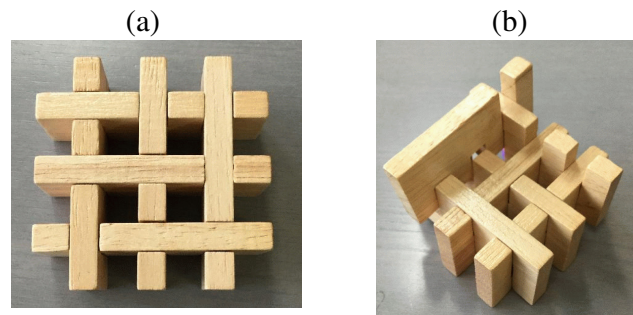


■ **Figure 2** The first lattice puzzle invented by T. Betsumiya (crafted by M. Uyematsu).

In this paper, we investigate the lattice puzzle and its variants. Typical one is illustrated in Figure 1: we are given some plates with slits, and the goal is to assemble them into the form of a lattice. There are many variants of them. In Japanese puzzle society, they say that the first one was called “cross block”, and invented by a Japanese puzzle designer, Toshiaki Betsumiya, in 1992 [1] (see Figure 2). This puzzle consists of ten plates. Each plate has five slits; three slits have depth $1/2$, one slit has depth $1/4$, and the last slit has depth $3/4$ (we normalize it to 1 for simplicity). This puzzle has a beautiful mathematical design ($\binom{5}{2} = 10$), and it is reasonably difficult (it has 20 solutions, however, it is difficult to find one). Since then, many variants are invented and are on the market.

It is natural to consider two variants of the cross block. First, the slits are on the same side of a plate, we call it *one-sided*. We can consider *two-sided* plates whose slits are on both sides as Figure 3. In the cross block, when two plates are assembled, there is no gap at the crossing point. We call it *fit*. That is, two slits of depth p and q can be assembled only if $p + q = 1$ in the fit model. In the *loose* model, we permit to assemble when $p + q > 1$. Some readers may think that the two-sided model should be loose; otherwise we may not be able to assemble/disassemble the plates as Figure 3(b). Actually, the two-sided fit model was invented in the puzzle society (see Figure 4). In this commercial product, each piece is made by rubber, and the puzzle itself is in the fit model. However, we can still consider the problem of assembling/disassembling. Even if the final form is feasible in the loose model, we may not assemble when the plates are rigid. In this paper, we focus on the problem that asks whether the assembled form is feasible or not. This assembling problem is another problem, which is not dealt with in this paper.

We here note that there is an old application of this problem. In the classic Japanese wood craft, called “kumiki” which means “assembling wood”, there is a tricky method to combine bars so that they seem to be “impossible” to assemble. Such a kumiki is actually



■ **Figure 3** Two-sided lattice puzzle.

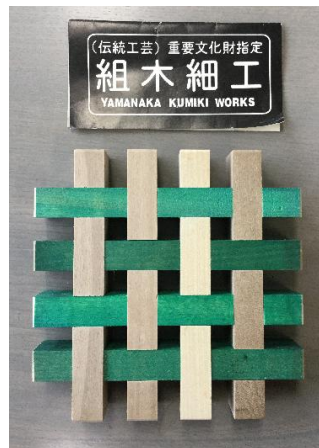


■ **Figure 4** Two-sided lattice puzzle with no gap and one depth.

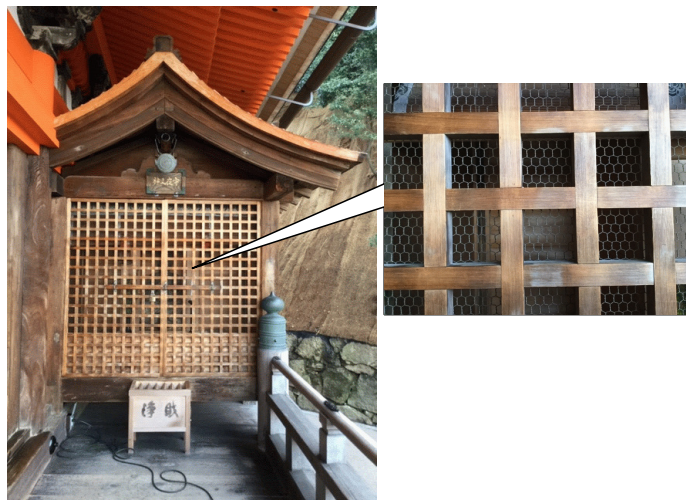
the puzzle which can be seen as the two-sided loose model (Figure 5). This kumiki pattern is known as “chidori-goshi”, which means “thousand-of-birds lattice”. For example, this craft can be found at the Kiyomizu-temple in Kyoto, which is one of the most famous temples in Japan (Figure 6). This kumiki method is too old and traditional to find the inventor¹. As a mark of respect for the inventor, we name this puzzle *lattice puzzle*.

As seen in Figure 6, we can consider the lattice puzzle of size $n \times m$ in general. When $n \neq m$, we can distinguish horizontal plates and vertical plates. However, if the lattice is square of size $n \times n$, we can consider a variant that asks us to partition the given set of $2n$ plates into two sets of n plates. We call this variant a *square lattice puzzle*. We also consider two variants based on the set of operations we use to solve a puzzle. To obtain a solution, one needs to order the plates in a sequence in each set. We call this process the permutation process. One also needs to flip the pieces so that they have correct directions and correct upsides. Note that each plate can be flipped to change its direction in the one-sided model, and to change its direction and its upside in the two-sided model. If the puzzle has a hint of the direction and the upside by, for example, coloring one of the endpoints and one of the sides of each plate, then one can only permute the pieces. We call this variant *permutation lattice puzzle*. If the puzzle has a hint of the order of each set of plates by, for example, assigning numbers to each set of plates, then one can only flip the pieces. We call this variant *flip lattice puzzle*.

¹ We found one in a literature published in 1770 at http://www.wul.waseda.ac.jp/kotenseki/html/116/116_00875/index.html.



■ **Figure 5** Traditional wooden craft in Japan (Kumiki).



■ **Figure 6** Large (im)possible wooden lattice at Kiyomizu temple in Kyoto.

We here summarize the table of our results in Table 1. It is easy to observe that the lattice puzzle is in the class NP since we can check the feasibility of a given solution in polynomial time. We show in Section 3 that the one-sided permutation loose model with 3 depths is NP-complete. This implies that the puzzle is NP-complete in general in the most flexible variant; two-sided, loose, and the number of different depths is not bounded.

On the other hand, we show in Section 4 that when we turn to the one-sided permutation fit model with 2 depths, this puzzle is GI-complete in general. The graph isomorphism problem (GI) is a decision problem that, given two graphs, decides whether they are isomorphic or not. This problem is one of the most well-studied problems in computational complexity and some related areas, and is believed to be in neither P nor NP-complete. There are several work to seek problems that are equivalent to GI. A problem is said to be GI-complete if the problem is as hard as the GI problem for general graphs. (Precisely speaking, a problem P is GI-complete if and only if the graph isomorphism problem for general graphs can be reducible to P and vice versa under polynomial time reduction.) As far as the authors know, there is no known characterization of the GI problem with such a simple puzzle.

■ **Table 1** Summary of results.

square/ colored	one/two sided	operations	#depths	rule	complexity	note
any	one-sided	permutation	3	loose	NP-complete	Theorem 2
colored	one-sided	permutation	2	fit	GI-complete	Theorem 3
any	one-sided	both	3	fit	GI-hard	Corollary 4
colored	one-sided	flip	unbounded	any	poly	Theorem 5
$n \times k$ (k :fixed)	any	both	unbounded	any	FPT for k	Theorem 6

Finally, we show in Section 5 that when we turn to the one-sided flip fit model, the problem can be reduced to the 2SAT problem, which can be solved in linear time for the size of the puzzle. We also consider the case that the lattice size is bounded as $n \times k$ for a fixed constant k . In this case, we show that the problem is fixed parameter tractable for k . That is, this problem can be solved in $f(k)p(n)$ time, where $p(n)$ is a polynomial function of n .

2 Definition of lattice puzzles and preliminaries

We first explain the one-sided model. We assume that each instance of a lattice puzzle P is given by two sets $X = \{x_1, \dots, x_n\}$ and $Y = \{y_1, \dots, y_m\}$ of *plates*. Each plate in X and Y is a rectangle of size $1 \times (m+1)$ and $1 \times (n+1)$, respectively. Let x be a plate in X . On x , we have m *slits* uniformly spaced on the long side. In the one-sided model, we denote the depth of the i th slit from the left by $d_i(x)$ with $0 < d_i(x) < 1$ (since the plate is disconnected if $d_i(x) = 1$, and we have no feasible solution if $d_i(x) = 0$). We define $d_j(y)$ in the same manner for each $y \in Y$. We distinguish a plate and the one obtained by flipping it, and denote by $\text{flip}(x)$ the plate obtained by flipping x . That is, if n is the number of slits of x , $d_i(\text{flip}(x)) = d_{n+1-i}(x)$ for every $1 \leq i \leq n$. We say that two plates x and y *fit* at point (i, j) if $d_i(x) + d_j(y) = 1$, and *weakly fit* if $d_i(x) + d_j(y) \geq 1$.

A solution of the puzzle is an arrangement of the plates in X and Y so that they form a lattice as shown in Figure 1 and Figure 2. More precisely, a solution is a pair of lists of plates $[x'_1, \dots, x'_n]$ and $[y'_1, \dots, y'_m]$ such that every x'_i is x or $\text{flip}(x)$ for some different $x \in X$ and every y'_j is y or $\text{flip}(y)$ for some different $y \in Y$ and x'_i and y'_j (weakly) fit at (j, i) for $1 \leq i \leq n$ and $1 \leq j \leq m$. That is, we have $d_j(x'_i) + d_i(y'_j) = 1$ in the fit model, and $d_j(x'_i) + d_i(y'_j) \geq 1$ in the loose model. We consider three different models based on the operations we can use. The above model is called the *all-operations model*. In the *permutation model*, neither $\text{flip}(x)$ nor $\text{flip}(y)$ do not appear in the list. In the *flip model*, the sets X and Y are ordered from the beginning and we are given two lists $[x_1, \dots, x_n]$ and $[y_1, \dots, y_m]$, and x'_i is x_i or $\text{flip}(x_i)$ for $1 \leq i \leq n$ and y'_j is y_j or $\text{flip}(y_j)$ for $1 \leq j \leq m$.

We here note for the special case $|X| = |Y| = n$. In this case, we cannot distinguish the plates of X and Y from their shapes. Thus, we can consider a variant of the lattice puzzle that a set of $2n$ plates is given and one divides it into two sets X and Y of n plates. We call this variant an $n \times n$ *square lattice puzzle*. Note that when we simply say an $n \times n$ lattice puzzle, we consider that two sets X and Y of plates are given. In this case, we sometimes say it is *colored* (as Figure 5) to emphasis the model.

In the two-sided model, we select one side and call it the positive side and the other one the negative side. If the i th slit of x is on the negative side, then we define $d_i(x)$ as $-1 < d_i(x) < 0$ according to the length of the slit. (That is, we do not allow to have two slits on both sides at the same position.) In this model, we allow two kinds of flip operations. That is, for a plate x , the plate $\text{flip}(x)$ such that $d_i(\text{flip}(x)) = d_{n+1-i}(x)$ for every i and the

plate $\text{sflip}(x)$ such that $d_i(\text{sflip}(x)) = -d_i(x)$ for every i . We say that two plates x and y *fit* at point (i, j) if $d_i(x) + d_j(y) = \pm 1$ and *weakly fit* if $|d_i(x) + d_j(y)| \geq 1$. The solution in the two-sided model can be defined in a similar way.

By default, we consider non-square one-sided all-operations 2-depth fit lattice puzzle and we omit these adjectives. We use adjectives square, two-sided, permutation, flipping, n -depth, any number of depth, and loose if they are.

For any given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with $|V_1| = |V_2|$, a bijection $\phi : V_1 \rightarrow V_2$ is said to be an *isomorphism* when $\{u, v\} \in E_1$ if and only if $\{\phi(u), \phi(v)\} \in E_2$. When there is an isomorphism between G_1 and G_2 , G_1 is *isomorphic* to G_2 . The graph isomorphism problem (GI) is a decision problem that, given two graphs G_1 and G_2 , decides whether G_1 is isomorphic to G_2 . A problem is *graph isomorphism complete* (GI-complete) if the problem is as hard as the graph isomorphism problem on general graphs. (GI-hardness is defined in the same manner of NP-hardness.) It is known that the graph isomorphism problem is GI-complete even if the input graphs are bipartite graphs (see, e.g., [7]).

In this paper, we did not give a definition of *fixed parameter tractability*; see e.g., [3] for the details. In our context, when an instance of a lattice puzzle P is given by two sets X and Y with $|X| = n$ and $|Y| = k$ for any fixed positive constant k , we say that the lattice puzzle P is fixed parameter tractable if there is an algorithm that solves P in $f(k)p(n)$ time, where f is any function of k , and p is a polynomial function of n .

3 NP-completeness

We first show the following key lemma:

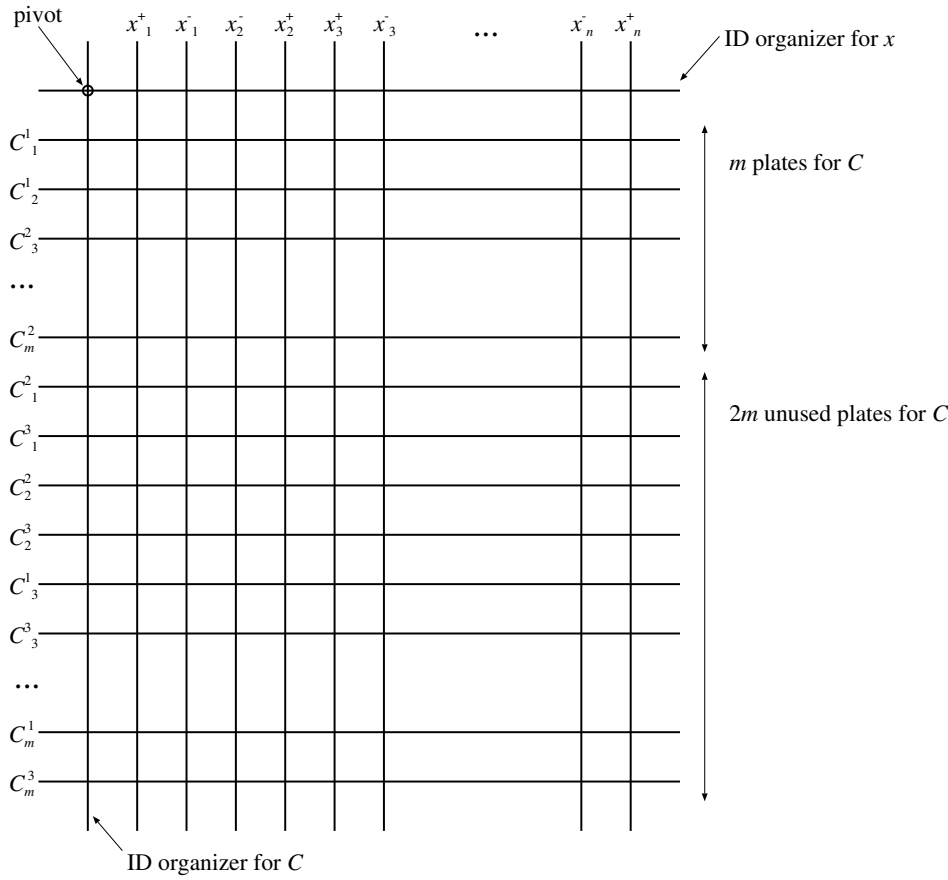
► **Lemma 1.** *The lattice puzzle of size $n \times m$ is NP-complete in the loose model with linear number of depths.*

Proof. We reduce the following positive 1-IN-3SAT problem, which is one of the well known NP-complete problems [4], to one-sided lattice puzzle (hence two-sided lattice puzzle immediately follows):

Input: A collection of clauses C_1, \dots, C_m of variables x_1, \dots, x_n such that each C_j is a disjunction of exactly three positive literals.

Question: Is there a truth assignment to the variables occurring so that exactly one literal is true in each C_j ?

We will use $n' = 2n + 1$ plates for variables and $m' = 3m + 1$ plates for clauses and reduce the above 1-IN-3SAT problem to the problem of solving an $n' \times m'$ lattice puzzle (in Figure 7, vertical plates are for variables, and horizontal ones are for clauses). There are two special horizontal plate p_x and vertical plate p_C which are called *ID organizers* for x and C , respectively. We first define $d_1(p_x) = \epsilon_0$ and $d_1(p_C) = 1 - \epsilon_0$ for sufficiently small $\epsilon_0 > 0$. In the following construction, the depth d of all the other slits satisfy $\epsilon_0 < d < 1 - \epsilon_0$. Therefore, p_x and p_C should be assembled as shown in Figure 7 at the *pivot* $(1, 1)$. We let $0 < \epsilon_0 < \epsilon_1 < \dots < \epsilon_n < \epsilon'_1 < \dots < \epsilon'_m < 1/4$ for some small distinct values. Then we define $d_2(p_x) = d_3(p_x) = \epsilon_1$, $d_4(p_x) = d_5(p_x) = \epsilon_2$, \dots , $d_{2i}(p_x) = d_{2i+1}(p_x) = \epsilon_i$, \dots , $d_{2n}(p_x) = d_{2n+1}(p_x) = \epsilon_n$. For these slits, we prepare $2n$ variable plates $x_1^+, x_1^-, x_2^+, x_2^-, \dots, x_n^+, x_n^-$. For each i with $1 \leq i \leq n$, we let $d_1(x_i^+) = d_1(x_i^-) = 1 - \epsilon_i$. As we will see, the depths of other vertical slits are $1/4$, $1/2$, or $3/4$. Therefore, it is easy to see that each pair of variable plates x_i^+ and x_i^- should be assembled at points $(2i, 1)$ and $(2i + 1, 1)$ (or $(2i + 1, 1)$ and $(2i, 1)$). In a similar way for the plate p_C with small values $\epsilon'_1, \dots, \epsilon'_m$, the ID organizer for C organizes



■ **Figure 7** Reduction from 1-IN-3SAT to lattice puzzle.

the clause plates as follows. We first define $d_2(p_C) = \epsilon'_1$, $d_3(p_C) = \epsilon'_2$, \dots , $d_{m+1}(p_C) = \epsilon'_m$. Then we further define $d_{m+2}(p_C) = d_{m+3}(p_C) = \epsilon'_1$, $d_{m+4}(p_C) = d_{m+5}(p_C) = \epsilon'_2$, \dots , $d_{m+2j}(p_C) = d_{m+2j+1}(p_C) = \epsilon'_j$, \dots , $d_{3m}(p_C) = d_{3m+1}(p_C) = \epsilon'_m$. For these slits, we prepare $3m$ clause plates C_j^1, C_j^2, C_j^3 ($1 \leq j \leq m$). For each j with $1 \leq j \leq m$, we let $d_1(C_j^1) = d_1(C_j^2) = d_1(C_j^3) = 1 - \epsilon'_j$. Thus, one of C_j^1, C_j^2, C_j^3 is assembled at $(1, j + 1)$ and the other two plates are assembled at $(1, m + 2j)$ and $(1, m + 2j + 1)$.

From the construction, we can observe that all ordering of these plates are fixed except (1) we can exchange x_i^+ and x_i^- , and (2) we can exchange C_j^1, C_j^2 , and C_j^3 . Now we give the assignments of depths for each $x_i^+, x_i^-, C_j^1, C_j^2$, and C_j^3 .

First, we define depths of variable plates x_i^+ and x_i^- . As depth, we use three values $3/4, 1/4$, and $1/2$. We set $d_j(x) = 3/4$ for all variable plates x and $j > m + 1$. Note that a slit with depth $3/4$ matches with any slit in the loose model. Therefore, for each $1 \leq j \leq m$, one can put any two (unused) clause plates C_j^k ($k \in \{1, 2, 3\}$) at some j th row for $j > m + 1$ as shown in the lower part of Figure 7.

For each $1 \leq j \leq m$, suppose that the clause C_j contains variables x_{i_1}, x_{i_2} and x_{i_3} . Then we set $d_{j+1}(x_{i_k}^+) = 3/4$ for each $k \in \{1, 2, 3\}$. The depth of other slits of variable plates are set to $1/2$.

Next, we define depths of clause plates. Suppose that C_j contains variables x_{i_1}, x_{i_2} , and x_{i_3} . Let $s_k^+ = 2i_k$ and $s_k^- = 2i_k + 1$, which are the two indices of the rows at which $x_{i_k}^+$ and $x_{i_k}^-$ can be placed. We assign depth $1/4$ to the slits at the following positions of C_j^1, C_j^2 , and

C_j^3 . On C_j^1 , we assign at positions s_1^+ , s_2^- , and s_3^- . On C_j^2 , we assign at positions s_2^+ , s_3^- , and s_1^- . On C_j^3 , we assign at positions s_3^+ , s_1^- , and s_2^- . At all the other positions, of clause plates we assign depth $1/2$.

Suppose that a solution of the original instance of the 1-IN-3SAT is given. If x_i is true, then we set the plate x_i^+ at position $2i$ and the plate x_i^- at position $2i + 1$. If x_i is false, we exchange these two plates. Then, for each $1 \leq j \leq m$, one can observe that exactly one of C_j^1 , C_j^2 , or C_j^3 can be placed on the $(j + 1)$ st line, and we put the other two plates on the lines with indices greater than $m + 1$. Thus, we obtain a solution of the lattice puzzle. This is a one-to-one correspondence, and one can construct a solution of the 1-IN-3SAT problem from a solution of this puzzle. ◀

In the proof of Lemma 1, except ID organizers for x and C , we need depths $1/4$, $1/2$, and $3/4$. Moreover, flipping is not required in the proof. These facts lead us to the following stronger result.

► **Theorem 2.** *The lattice puzzle of size $n \times m$ and the square lattice puzzle of size $n \times n$ are NP-complete in the loose model with 3 depths even if one-sided model and only permutation is permitted.*

Proof. It is enough to show that we can design a “frame” surrounding the puzzle in the proof of Lemma 1. A brief sketch is given in right down in Figure 8. The gray area of the figure forms a frame, which plays ID organizers in the proof of Lemma 1. The magnification of the left up corner of the frame is depicted at left up in Figure 8. Let $n' = \max\{2n, 3m\}$. Then the frame F is the set of $4n'$ plates. For each $i = 1, 2, \dots, n'$, we have two copies of f_i and two copies of f'_i . The set of f_i s plays the role of the ID organizer of C , and the set of f'_i s plays the rule of the ID organizer of x .

In Figure 8, each small circle on f_i corresponds to depth $3/4$, and each small circle on f'_i corresponds to depth $1/4$. The other intersection of f_i and f'_j , the depth is $1/2$.

We apply the same manner for each intersection between f_i and C_k^j and each intersection between f'_i and x_j^+ or x_j^- . Precisely, for example, f_2 has depth $3/4$ at the points corresponding to C_1^1 , C_1^2 , and C_1^3 , and has depth $1/2$ at the other points. In general, f_{i+1} has depth $3/4$ at the points corresponding to C_i^1 , C_i^2 , and C_i^3 . Similarly, each f'_{i+1} has depth $1/4$ at the points corresponding to x_i^+ and x_i^- , and has depth $1/2$ at the other points. The corresponding depths of the plates x_j and C_j are trivial.

We note that they fit without any gap in the gray area of the figure. Since there is no gap at the gray area, we can observe that the shape of the frame is uniquely formed by these $4n'$ plates.

Therefore, combining the reduction in Lemma 1, we prove that the lattice puzzle of size $n \times m$ is NP-complete in the loose model with 3 depths.

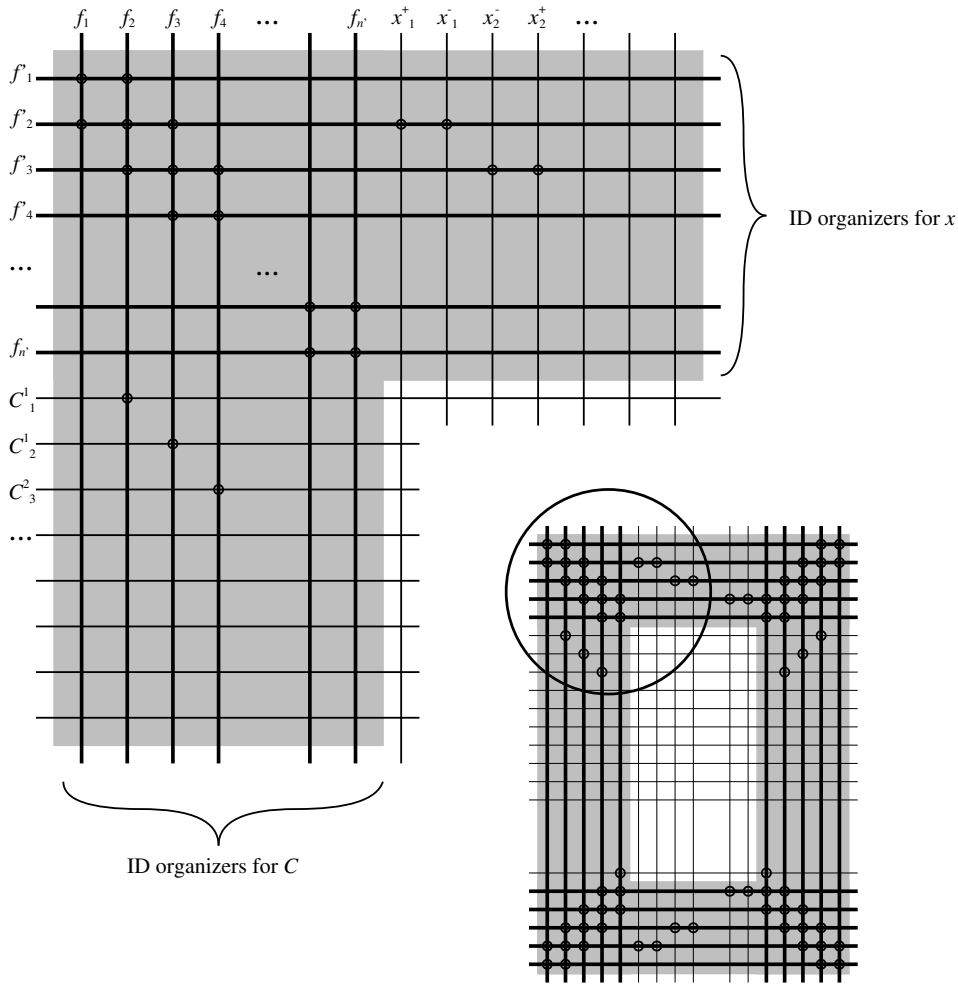
It is easy to see that the reductions work even in one-sided model and only permutation is permitted. Addition of extra plates to make the frame square is trivial. ◀

4 GI-completeness

In this section, we show the following theorem:

► **Theorem 3.** *The permutation lattice puzzle is GI-complete.*

Proof. We show a correspondence between lattice puzzles and graph isomorphism problems of bipartite graphs. Let $G_1 = (A_1, B_1, E_1)$ and $G_2 = (A_2, B_2, E_2)$ be two bipartite graphs with $|A_1| = |A_2| = n$ and $|B_1| = |B_2| = m$. Let $A_1 = \{a_1, a_2, \dots, a_n\}$, $B_1 = \{b_1, b_2, \dots, b_m\}$,



■ **Figure 8** Framing of the lattice puzzle in Lemma 1.

$A_2 = \{a'_1, a'_2, \dots, a'_n\}$ and $B_2 = \{b'_1, b'_2, \dots, b'_m\}$. From these graphs, we construct a lattice puzzle as follows. The set $X = \{x_1, \dots, x_n\}$ of plates is constructed from G_1 : the plate x_i of size $1 \times (m + 1)$ corresponds to $a_i \in A_1$ and $d_j(x_i) = 2/3$ if $\{a_i, b_j\}$ is in E_1 , and $d_j(x_i) = 1/3$ otherwise. The set $Y = \{y_1, \dots, y_m\}$ of plates is constructed from G_2 : the plate y_j of size $1 \times (n + 1)$ corresponds to $b'_j \in B_2$ and $d_i(y_j) = 1/3$ if $\{a'_i, b'_j\}$ is in E_2 , and $d_i(y_j) = 2/3$ otherwise.

Now we check that a graph isomorphism gives us a solution of the puzzle, and vice versa. Let ϕ_A and ϕ_B be permutations on $\{1, \dots, n\}$ and $\{1, \dots, m\}$, respectively. ϕ_A induces a bijection $a_i \mapsto a'_{\phi_A(i)}$ from A_1 to A_2 , and ϕ_B induces a bijection $b_j \mapsto b'_{\phi_B(j)}$ from B_1 to B_2 . They form a graph isomorphism from G_1 to G_2 if and only if $\{a_i, b_j\} \in E_1 \Leftrightarrow \{a'_{\phi_A(i)}, b'_{\phi_B(j)}\} \in E_2$ for $1 \leq i \leq n$ and $1 \leq j \leq m$. On the other hand, ϕ_A induces a permutation on X such that the result of permutation $X' = [x'_1, \dots, x'_n]$ is given as $x'_i = x_{\phi_A^{-1}(i)}$. In the same way, ϕ_B^{-1} induces a permutation on Y such that the result of permutation $Y' = [y'_1, \dots, y'_m]$ is given as $y'_j = y_{\phi_B(j)}$.

Now, we look at the point (i, j) where the plates $x'_i = x_{\phi_A^{-1}(i)}$ and $y'_j = y_{\phi_B(j)}$ are crossing. On the plate x'_i , $d_j(x'_i) = 2/3$ if and only if $\{a_{\phi_A^{-1}(i)}, b_j\} \in E_1$ and on the plate y'_j , $d_i(y'_j) = 1/3$ if and only if $\{a'_i, b'_{\phi_B(j)}\} \in E_2$. Therefore, X' and Y' form a solution of the

32:10 On the Complexity of Lattice Puzzles

puzzle if and only if $\{a_{\phi_A^{-1}(i)}, b_j\} \in E_1 \Leftrightarrow \{a'_i, b'_{\phi_B(j)}\} \in E_2$ for $1 \leq i \leq n$ and $1 \leq j \leq m$. Let $i' = \phi_A^{-1}(i)$. Then, it is equivalent to $\{a_{i'}, b_j\} \in E_1 \Leftrightarrow \{a'_{\phi_A(i)}, b'_{\phi_B(j)}\} \in E_2$ for $1 \leq i' \leq n$ and $1 \leq j \leq m$. That is, G_1 and G_2 are isomorphic. \blacktriangleleft

► Corollary 4.

- (1) *The square lattice puzzle is GI-complete.*
- (2) *The square permutation lattice puzzle is GI-complete.*

Proof. We reduce a colored permutation puzzle of size $n \times n$ to a square all-operations puzzle of size $(n+4) \times (n+4)$. We denote by 0 a slot with depth $1/3$, and by 1 a slot with depth $2/3$. Suppose that an $n \times n$ non-square lattice puzzle with the lists of plates $X = [x_1, \dots, x_n]$, $Y = [y_1, \dots, y_n]$ is given. From X and Y , we form a new set $\tilde{Z} = \{\tilde{x}_1, \dots, \tilde{x}_n, \tilde{y}_1, \dots, \tilde{y}_n, \tilde{z}_1, \dots, \tilde{z}_8\}$ of plates with $n+4$ slots. The sequences of slots are given as follows. Here, \bar{x} is the sequence of slots of x .

$$\begin{array}{llll} \tilde{x}_i & = 1 & 0 & \bar{x}_i \quad 0 & 0 & (1 \leq i \leq n) \\ \tilde{y}_i & = 0 & 1 & \bar{y}_i \quad 0 & 0 & (1 \leq i \leq n) \\ \tilde{z}_i & = 1 & 1 & 1^n & 0 & 0 \quad (i = 1, 2, 3, 4) \\ \tilde{z}_i & = 1 & 0 & 1^n & 0 & 1 \quad (i = 5, 6) \\ \tilde{z}_i & = 0 & 1 & 0^n & 1 & 0 \quad (i = 7, 8) \end{array}$$

Let $[x'_1, \dots, x'_{n+4}]$ and $[y'_1, \dots, y'_{n+4}]$ be the lists of plates which form a solution of this puzzle. We first study $x'_1, x'_2, x'_{n+3}, x'_{n+4}$ and $y'_1, y'_2, y'_{n+3}, y'_{n+4}$. First note that these plates cross at the four 2×2 corners, and each place needs to contain at least one 1-slot. It means we need, in all, 16 1-slots at the positions $1, 2, n+3, n+4$ of the 8 plates. Therefore, these 8 plates must be z_i ($1 \leq i \leq 8$). One can see that $[x'_1, x'_2, x'_{n+3}, x'_{n+4}] = [z_1, z_7, \text{flip}(z_2), z_5]$ and $[y'_1, y'_2, y'_{n+3}, y'_{n+4}] = [z_8, \text{flip}(z_3), z_6, z_4]$ form a solution. In this case, for each $3 \leq j \leq n+2$, the j th slots of $x'_1, x'_2, x'_{n+3}, x'_{n+4}$ form the sequence 0 1 1 1, and those of $y'_1, y'_2, y'_{n+3}, y'_{n+4}$ form the sequence 1 0 1 1. There are other possibilities of the assignments of $x'_1, x'_2, x'_{n+3}, x'_{n+4}, y'_1, y'_2, y'_{n+3}, y'_{n+4}$. However, one can see that, in all of them, we have the same sequences 0 1 1 1 and 1 0 1 1 or their rotation at these slots. It means that $\{x'_i \mid 3 \leq i \leq n+2\}$ must be $\{\tilde{x}_i \mid 1 \leq i \leq n\}$ and $\{y'_i \mid 3 \leq i \leq n+2\}$ must be $\{\tilde{y}_i \mid 1 \leq i \leq n\}$. Note that \tilde{x}_i and \tilde{y}_i cannot be flipped. Therefore, there is a correspondence between solutions of the original non-square permutation puzzle of size $n \times n$ and this square all-operations puzzle of size $(n+2) \times (n+2)$. \blacktriangleleft

5 Polynomial time algorithms

In this section, we show two variants that can be solved in polynomial time.

5.1 Fixed ordering case

In this variant, we assume that the place of each plate is fixed. That is, the lattice of size $n \times m$ is fixed, and each plate can only be flipped. In such a restricted case, we still have 2^{n+m} possible cases. However, we can solve this variant in linear time:

► **Theorem 5.** *The lattice puzzle of size $n \times m$ can be solved in $O(nm)$ time in the flip fit model with 2 depths².*

² The modification of the number of depths from 2 to any positive integer is straightforward and omitted here.

Proof. Without loss of generality, the set $X = \{x_1, \dots, x_n\}$ of n plates with m slits and the set $Y = \{y_1, \dots, y_m\}$ of m plates with n slits are given, their positions are given by their indices, and the depth of each slit is $1/3$ or $2/3$. We reduce this puzzle to the 2SAT problem, which can be solved in linear time.

From the set X of n plates, we define a Boolean matrix A of size $n \times m$ which may contain Boolean variables a_1, \dots, a_n . For each $1 \leq i \leq n$ and $1 \leq j \leq m$, we define

$$A_{i,j} = \begin{cases} T & \text{if } d_j(x_i) = d_{m-j+1}(x_i) = 1/3 \\ F & \text{if } d_j(x_i) = d_{m-j+1}(x_i) = 2/3 \\ a_i & \text{if } d_j(x_i) = 1/3 \text{ and } d_{m-j+1}(x_i) = 2/3 \\ \bar{a}_i & \text{if } d_j(x_i) = 2/3 \text{ and } d_{m-j+1}(x_i) = 1/3. \end{cases}$$

The variable a_i represents the direction of the i th plate. Similarly, the matrix B is defined from the set Y of m plates as

$$B_{i,j} = \begin{cases} T & \text{if } d_i(y_j) = d_{n-i+1}(y_j) = 2/3 \\ F & \text{if } d_i(y_j) = d_{n-i+1}(y_j) = 1/3 \\ b_j & \text{if } d_i(y_j) = 2/3 \text{ and } d_{n-i+1}(y_j) = 1/3 \\ \bar{b}_j & \text{if } d_i(y_j) = 1/3 \text{ and } d_{n-i+1}(y_j) = 2/3, \end{cases}$$

for each $1 \leq i \leq n$ and $1 \leq j \leq m$, where b_1, \dots, b_m are Boolean variables. Observe that the plates x_i and y_j can fit at (i, j) if and only if $A_{i,j} = B_{i,j}$.

Now we solve the assignment problem for these variables. This condition can be represented by two clauses as $(\alpha_i^j \wedge \beta_i^j) \vee (\bar{\alpha}_i^j \wedge \bar{\beta}_i^j)$, where α_i^j and β_i^j are the literals appearing in $A_{i,j}$ and $B_{i,j}$, respectively. We consider the 2-CNF formula obtained as their conjunction. Then, this puzzle is solvable if and only if there is a satisfying assignment for this formula.

Suppose that it is satisfied with a variable assignment to $a_1, \dots, a_n, b_1, \dots, b_m$. We obtain a solution of the puzzle with the following procedure: If $a_i = F$, then we flip x_i for $i = 1, \dots, n$. If $b_j = F$, then we flip y_j for $j = 1, \dots, m$. Since the 2SAT problem can be solved in polynomial time, Theorem 5 follows. ◀

5.2 Fixed parameter tractable algorithm

In this variant, we consider the lattice of size $n \times k$ for a fixed constant k . First, we mention that the fit model with one-sided plates is easy to solve by checking all permutations of k plates of size $1 \times (n + 1)$. Considering the flipping, we have $k!2^k$ ways to arrange these k plates. Once we fix one arrangement of k plates, checking of feasibility is straightforward in $O(kn^2)$ time. Therefore, the algorithm runs in $O(k!2^k kn^2)$ time. For the two-sided plates, the number of possible permutations is $k!4^k$, and the checking of feasibility can be done in $O(kn^2)$ time. Therefore, we can solve the lattice puzzle in $O(k!4^k kn^2)$ time. We extend this idea to the loose model:

▶ **Theorem 6.** *The lattice puzzle of size $n \times k$ in the loose model can be solved by a fixed parameter tractable algorithm with parameter k .*

Proof. We first consider the one-sided plates. The basic idea is similar to the algorithm for the fit model: the algorithm checks all $k!2^k$ permutations. Now we fix a permutation. We have n possible places for n plates. Thus, we construct a bipartite graph $G = (X, Y, E)$ as follows. Let $X = \{x_1, x_2, \dots, x_n\}$ be the set of n plates of size $1 \times (k + 1)$, and $Y = \{y_1, y_2, \dots, y_n\}$ be the places produced by k plates of size $1 \times (n + 1)$. E consists of an edge $\{x_i, y_j\}$ if and only if the plate x_i can be assembled to the place y_j . The construction of the graph G

takes $O(kn^2)$ time. Then, it is easy to see that a solution of the lattice puzzle corresponds to a perfect matching on G . It is known that the perfect matching problem on a bipartite graph can be solved in polynomial time $p(|X| + |Y|) = O(\min\{\sqrt{|X| + |Y|}|E|, (|X| + |Y|)^\omega\})$, where $\omega < 2.373$ is the matrix multiplication exponent.

Therefore, the lattice puzzle can be solved in $O(k!2^k(kn^2 + p(n + k)))$ time. When the plates are two-sided, in the same way, we can solve it in $O(k!4^k(kn^2 + p(n + k)))$ time. ◀

6 Concluding remarks

In this paper, we propose a general framework of simple lattice puzzles. Using this framework, we can characterize some representative computational complexity classes. Especially, we can characterize the problems in NP-complete and GI-complete. As far as the authors know, there is no such a simple framework.

Although we show several results, we still have many unsolved problems. Especially, computational complexity of the simplest problem on $2n$ plates of size $1 \times (n + 1)$ in the one-sided fit model is open. By Theorems 2 and 3, it seems that this problem exists between the NP-complete problem and the GI-complete problem.

We also mention that we focus on the problems that ask if a given set of plates has a feasible state or not in this paper. That is, we do not ask if the feasible state can be assembled even if each plate is rigid. Like the sliding block puzzles, allowing the “movement of pieces”, the assembling puzzle of rigid plates can be PSPACE-complete. Is there a variant of the lattice puzzle with movement which gives us a characterization of PSPACE-completeness?

References

- 1 Hisayoshi Akiyama. *Cube Puzzle Book (in Japanese)*. Shinkigensha, 2004.
- 2 E. R. Berlekamp, J. H. Conway, and R. K. Guy. *Winning Ways for Your Mathematical Plays*, volume 1–4. A K Peters Ltd., 2001–2003.
- 3 M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshantov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 4 M.R. Garey and D.S. Johnson. *Computers and Intractability – A Guide to the Theory of NP-Completeness*. Freeman, 1979.
- 5 R. A. Hearn and E. D. Demaine. *Games, Puzzles, and Computation*. A K Peters Ltd., 2009.
- 6 Takumi Kasai, Akeo Adachi, and Shigeki Iwata. Classes of pebble games and complete problems. *SIAM Journal on Computing*, 1979.
- 7 R. Uehara, S. Toda, and T. Nagoya. Graph Isomorphism Completeness for Chordal Bipartite Graphs and Strongly Chordal Graphs. *Discrete Applied Mathematics*, 145(3):479–482, 2004. doi:10.1016/j.dam.2004.06.008.

The I/O Complexity of Hybrid Algorithms for Square Matrix Multiplication

Lorenzo De Stefani

Department of Computer Science, Brown University, United States of America
lorenzo@cs.brown.edu

Abstract

Asymptotically tight lower bounds are derived for the I/O complexity of a general class of hybrid algorithms computing the product of $n \times n$ square matrices combining “Strassen-like” fast matrix multiplication approach with computational complexity $\Theta(n^{\log_2 7})$, and “standard” matrix multiplication algorithms with computational complexity $\Omega(n^3)$. We present a novel and tight $\Omega\left(\left(\frac{n}{\max\{\sqrt{M}, n_0\}}\right)^{\log_2 7} (\max\{1, \frac{n_0}{M}\})^3 M\right)$ lower bound for the I/O complexity of a class of “uniform, non-stationary” hybrid algorithms when executed in a two-level storage hierarchy with M words of fast memory, where n_0 denotes the threshold size of sub-problems which are computed using standard algorithms with algebraic complexity $\Omega(n^3)$.

The lower bound is actually derived for the more general class of “non-uniform, non-stationary” hybrid algorithms which allow recursive calls to have a different structure, even when they refer to the multiplication of matrices of the same size and in the same recursive level, although the quantitative expressions become more involved. Our results are the first I/O lower bounds for these classes of hybrid algorithms. All presented lower bounds apply even if the recomputation of partial results is allowed and are asymptotically tight.

The proof technique combines the analysis of the Grigoriev’s flow of the matrix multiplication function, combinatorial properties of the encoding functions used by fast Strassen-like algorithms, and an application of the Loomis-Whitney geometric theorem for the analysis of standard matrix multiplication algorithms. Extensions of the lower bounds for a parallel model with P processors are also discussed.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases I/O complexity, Hybrid Algorithm, Matrix Multiplication, Recomputation

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.33

Related Version A full version of the paper is available at <https://arxiv.org/abs/1904.12804>.

Funding This research was supported by NSF Award ISS 1813444.

Acknowledgements I want to thank Gianfranco Bilardi at the University of Padova (Italy) for initial conversations on the topic of this work, and Megumi Ando at Brown University (USA) for her feedback on early versions of this manuscript.

1 Introduction

Data movement plays a critical role in the performance of computing systems, in terms of both time and energy. This technological trend [16] appears destined to continue, as physical limitations on minimum device size and on maximum message speed lead to inherent costs when moving data, whether across the levels of a hierarchical memory system or between processing elements of a parallel system [13]. While the communication requirements of algorithms have been widely investigated in the literature, obtaining significant and tight lower bounds based on such requirements remains an important and challenging task.



© Lorenzo De Stefani;

licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 33; pp. 33:1–33:16

Leibniz International Proceedings in Informatics



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

In this paper, we focus on the I/O complexity of a general class of hybrid algorithms for computing the product of square matrices. Such algorithms combine *fast* algorithms with base case 2×2 similar to Strassen’s matrix multiplication algorithm [36] with algebraic (or *computational*) complexity $\mathcal{O}(n^{\log_2 7})$ with *standard* (or *classic*) matrix multiplication algorithms with algebraic complexity $\Omega(n^3)$. Further, these algorithms allow recursive calls to have a different structure, even when they refer to the multiplication of matrices in the same recursive level and of the same input size. These algorithms are referred in literature as “*non-uniform, non-stationary*”. This class includes, for example, algorithms that optimize for input sizes [19, 20, 23]. Matrix multiplication is a pervasive primitive utilized in many applications.

While of actual practical importance, to the best of our knowledge, no characterization of the I/O complexity of such algorithms has presented before this work. This is likely due to the fact that the irregular nature of hybrid algorithms and, hence, the irregular structure of the corresponding Computational Directed Acyclic Graphs (CDAGs), complicates the analysis of the combinatorial properties of the CDAG which is the foundation of many of I/O lower bound technique presented in literature (e.g., [8, 25, 32]).

The technique used in this work overcomes such challenges and yields asymptotically tight I/O lower bounds which hold even if recomputation of intermediate values is allowed.

Previous and Related Work. Strassen [36] showed that two $n \times n$ matrices can be multiplied with $O(n^\omega)$ operations, where $\omega = \log_2 7 \approx 2.8074$, hence with asymptotically fewer than the n^3 arithmetic operations required by the straightforward implementation of the definition of matrix multiplication. This result has motivated a number of efforts which have led to increasingly faster algorithms, at least asymptotically, with the current record being at $\omega < 2.3728639$ [28].

I/O complexity has been introduced in the seminal work by Hong and Kung [25]; it is essentially the number of data transfers between the two levels of a memory hierarchy with a fast memory of M words and a slow memory with an unbounded number of words. Hong and Kung presented techniques to develop lower bounds to the I/O complexity of computations modeled by *computational directed acyclic graphs* (CDAGs). The resulting lower bounds apply to all the schedules of the given CDAG, including those with recomputation, that is, where some vertices of the CDAG are evaluated multiple times. Among other results, they established a $\Omega(n^3/\sqrt{M})$ lower bound to the I/O complexity of standard, definition-based matrix multiplication algorithms, which matched a known upper bound [15]. The techniques of [25] have also been extended to obtain tight communication bounds for the definition-based matrix multiplication in some parallel settings [4, 24]. Ballard et al. generalized the results on matrix multiplication of Hong and Kung [25] in [7, 6] by using the approach proposed in [24] based on the Loomis-Whitney geometric theorem [29, 37].

In an important contribution, Ballard et al. [8], obtained an $\Omega((n/\sqrt{M})^{\log_2 7} M)$ I/O lower bound for Strassen’s algorithm, using the “*edge expansion approach*”. The authors extend their technique to a class of “*Strassen-like*” fast multiplication algorithms and to fast recursive multiplication algorithms for rectangular matrices [3]. This result was later generalized to increasingly broader classes of “*Strassen-like*” algorithms by Scott et. al [33] using the “*path routing*” technique, and De Stefani [17] using a combination the concept of Grigoriev’s flow of a function and the “*dichotomy width*” technique [14]. While the previously mentioned results hold only under the restrictive assumption that no intermediate result may be more than once (i.e., the *no-recomputation assumption*), in [10] Bilardi and De Stefani

introduced the first asymptotically tight I/O lower bound which holds if recomputation is allowed. Their technique was later extended to the analysis of Strassen-like algorithms with base case 2×2 [30], and to the analysis of Toom-Cook integer multiplication algorithms [11].

A parallel, “*communication avoiding*” implementation of Strassen’s algorithm whose performance matches the known lower bound [8, 33], was proposed by Ballard et al. [5].

In [34], Scott derived a lower bound for the I/O complexity of a class of uniform, non-stationary algorithms combining Strassen-like algorithm with recursive standard algorithms. This result holds only under the restrictive no-recomputation assumption and considers only compositions of recursive matrix multiplication algorithms.

To the best of our knowledge, ours is the first work presenting asymptotically tight I/O lower bounds for non-uniform, non-stationary hybrid algorithms for matrix multiplication that hold when recomputation is allowed.

On the impact of recomputation. While it is of interest to study the I/O complexity under the no-recomputation assumption, it is also very important to investigate what can be achieved with recomputation. For some CDAGs, recomputing intermediate values allows reducing the space and/or the I/O complexity of an algorithm [32]. As shown [12], some algorithms admit a “*portable schedule*” (i.e., a schedule which achieves optimal performance across memory hierarchies with different access costs) only if recomputation is allowed. Recomputation can also enhance the performance of simulations among networks (see [27] and references therein) and plays a key role in the design of efficient area-universal VLSI architectures with constant slowdown [9].

Our results. In our main result, we present the first I/O lower bound for a class \mathfrak{H} of non-uniform, non-stationary hybrid matrix multiplication algorithms when executed in a two-level storage hierarchy with M words of fast memory. Algorithms in \mathfrak{H} combine fast Strassen-like algorithms with base case 2×2 with algebraic complexity $\Theta(n^{\log_2 7})$, and standard algorithms based on the definition with algebraic complexity $\Omega(n^3)$. These algorithms allow recursive calls to have a different structure, even when they refer to the multiplication of matrices in the same recursive level and of the same input size. The result in Theorem 9 relates the I/O complexity of algorithms in \mathfrak{H} to the number and the input size of an opportunely selected set of the sub-problems generated by the algorithms themselves.

By specializing the result in Theorem 9, we also present, in Theorem 11, a novel $\Omega\left(\left(\frac{n}{\max\{\sqrt{M}, n_0\}}\right)^{\log_2 7} (\max\{1, \frac{n_0}{M}\})^3 M\right)$ lower bound for the I/O complexity of algorithms in a subclass $\mathfrak{U}\mathfrak{H}(n_0)$ of \mathfrak{H} composed by *uniform non-stationary* hybrid algorithms where n_0 denotes the threshold input size of sub-problems which are computed using standard algorithms with algebraic complexity $\Omega(n^3)$.

The result in Theorem 11 considerably extends a previous result by Scott [34] as the latter covers only a sub-class of $\mathfrak{U}\mathfrak{H}(n_0)$ composed by uniform, non-stationary algorithms combining Strassen-like algorithms with the recursive standard algorithm, and holds only assuming that no intermediate value is recomputed.

Our lower bounds in Theorem 9 and Theorem 11 allow for recomputation of intermediate values and are asymptotically tight. As the matching upper bounds do not recompute any intermediate value, we conclude that using recomputation may reduce the I/O complexity of the considered classes of hybrid algorithms by at most a constant factor.

Our proof technique is of independent interest since it exploits to a significant extent the “*divide and conquer*” nature exhibited by many algorithms. Our approach combines elements from the “*G-flow*” I/O lower bound technique originally introduced by Bilardi

and De Stefani, with an application of the Loomis-Whitney geometric theorem, which has been used by Irony et al. to study the I/O complexity of standard matrix multiplication algorithms [24], to recover an information which relates to the concept of “*Minimum set*” introduced in Hong and Kung’s method. We follow the “*dominator set*” approach pioneered by Hong and Kung in [25]. However, we focus the dominator analysis only on a select set of target vertices, which, depending on the algorithm structure, correspond either to the outputs of the sub-CDAGs that correspond to sub-problems of a suitable size (i.e., chosen as a function of the fast memory capacity M) computed using a fast Strassen-like algorithm, or to the the vertices corresponding to the elementary products evaluated by a standard (definition) matrix multiplication algorithm.

We derive our main results for the hierarchical memory model (or external memory model). Our results generalize to parallel models with P processors. For these parallel models, we derive lower bounds for the “*bandwidth cost*”, that is for the number of messages (and, hence, the number of memory words) that must be either sent or received by at least one processor during the execution of the algorithm.

Paper organization. In Section 2 we outline the notation and the computational models used in the rest of the presentation. In Section 3 we rigorously define the class of hybrid matrix multiplication algorithms \mathfrak{H} being considered. In Section 4 we discuss the construction and important properties of the CDAGs corresponding to algorithms in \mathfrak{H} . In Section 5 we introduce the concept of Maximal Sup-Problem (MSP) and describe their properties which lead to the I/O lower bounds for algorithms in \mathfrak{H} in Section 6.

2 Preliminaries

We consider algorithms that compute the product of two square matrices $\mathbf{A} \times \mathbf{B} = \mathbf{C}$ with entries from a ring \mathcal{R} . We use A to denote the set of variables each corresponding to an entry of matrix \mathbf{A} . We refer to the number of entries of a matrix \mathbf{A} as its “*size*” and we denote it as $|A|$. We denote the entry on the i -th row of the j -th column of matrix \mathbf{A} as $\mathbf{A}[i][j]$.

In this work we focus on algorithms whose execution can be modeled as a *computational directed acyclic graph* (CDAG) $G = (V, E)$. Each vertex $v \in V$ represents either an input value or the result of a unit-time operation (i.e., an intermediate result or one of the output values) which is stored using a single memory word. For example, each of the input (resp., output) vertices of G corresponds to one of the $2n^2$ entries of the factor matrices \mathbf{A} and \mathbf{B} (resp., to the n^2 entries of the product matrix \mathbf{C}). The *directed* edges in E represent data dependencies. That is, a pair of vertices $u, v \in V$ are connected by an edge (u, v) directed from u to v if and only if the value corresponding to u is an operand of the unit time operation which computes the value corresponding to v . A *directed path* connecting vertices $u, v \in V$ is an ordered sequence of vertices starting with u and ending with v , such that there is an edge in E directed from each vertex in the sequence to its successor.

We say that $G' = (V', E')$ is a *sub-CDAG* of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E \cap (V' \times V')$. Note that, according to this definition, every CDAG is a sub-CDAG of itself. We say that two sub-CDAGs $G' = (V', E')$ and $G'' = (V'', E'')$ of G are *vertex-disjoint* if $V' \cap V'' = \emptyset$. Analogously, two directed paths in G are vertex-disjoint if they do not share any vertex.

When analyzing the properties of CDAGs we make use of the concept of *dominator set* originally introduced in [25]. We use the following – slightly different – definition:

► **Definition 1** (Dominator set). *Given a CDAG $G = (V, E)$, let $I \subseteq V$ denote the set of its input vertices. A set $D \subseteq V$ is a dominator set for $V' \subseteq V$ with respect to $I' \subseteq I$ if every path from a vertex in I' to a vertex in V' contains at least a vertex of D . When $I' = I$, D is simply referred as “a dominator set for V' ”.*

I/O model and machine models. We assume that sequential computations are executed on a system with a two-level memory hierarchy, consisting of a fast memory or *cache* of size M (measured in memory words) and a *slow memory* of unlimited size. An operation can be executed only if all its operands are in cache. We assume that each entry of the input and intermediate results matrices (including entries of the output matrix) is maintained in a single memory word (the results trivially generalize if multiple memory words are used).

Data can be moved from the slow memory to the cache by **read** operations and, in the other direction, by **write** operations. These operations are also called *I/O operations*. We assume the input data to be stored in slow memory at the beginning of the computation. The evaluation of a CDAG in this model can be analyzed by means of the “*red-blue pebble game*” [25]. The number of I/O operations executed when evaluating a CDAG depends on the “*computational schedule*”, that is, it depends on the order in which vertices are evaluated and on which values are kept in/discarded from cache.

The *I/O complexity* $IO_G(M)$ of a CDAG G is defined as the minimum number of I/O operations over all possible computational schedules. We further consider a generalization of this model known as the “*External Memory Model*” by Aggarwal and Vitter [2], where $B \geq 1$ values can be moved between cache and consecutive slow memory locations with a single I/O operation. For $B = 1$, this model clearly reduces to the red-blue pebble game.

Given an algorithm \mathcal{A} , we only consider “*parsimonious execution schedules*”, that is schedules such that: (i) each time an intermediate result (excluding the output entries of \mathbf{C}) is computed, such value is then used to compute at least one of the values of which it is an operand before being removed from the memory (either the cache or slow memory); and (ii) any time an intermediate result is read from slow to cache memory, such value is then used to compute at least one of the values of which it is an operand before being removed from the memory or moved back to slow memory using a *write* I/O operation. Clearly, any non-parsimonious schedule \mathcal{C} can be reduced to a parsimonious schedule \mathcal{C}' by removing all the steps which violate the definition of parsimonious computation. \mathcal{C}' has therefore less computational and I/O operations than \mathcal{C} . Hence, restricting the analysis to parsimonious computations leads to no loss of generality.

We also consider a parallel model where P processors, each with a local memory of size $2n^2/P \leq M < n^2$, are connected by a network. We do not, however, make any assumption on the initial distribution of the input data nor regarding the balance of the computational load among the P processors. Processors can exchange point-to-point messages, with every message containing up to B_m memory words. For this parallel model, we derive lower bounds for the number of messages that must be either sent or received by at least one processor during the CDAG evaluation. The notion of “*parsimonious execution schedules*” straightforwardly extends to this parallel model.

3 Hybrid matrix multiplication algorithms

In this work, we consider a family of hybrid matrix multiplication algorithms obtained by hybridizing the two following classes of algorithms:

Standard matrix multiplication algorithms. This class includes all the square matrix multiplication algorithms which, given the input factor matrices $\mathbf{A}, \mathbf{B} \in \mathcal{R}^{n \times n}$, satisfy the following properties:

- The n^3 elementary products $\mathbf{A}[i][j]\mathbf{B}[j][i]$, for $i, j = 0, \dots, n-1$, are *directly computed*;
- Each of the $\mathbf{C}[i][j]$ is computed by summing the values of the n elementary products $\mathbf{A}[i][z]\mathbf{B}[z][j]$, for $z = 0, \dots, n-1$, through a *summation tree* by additions and subtractions only;
- The evaluations of the $\mathbf{C}[i][j]$'s are *independent of each other*. That is, internal vertex sets of the summation trees of all the $\mathbf{C}[i][j]$'s are *disjoint from each other*.

This class, also referred in literature as *classic*, *naive* or *conventional* algorithms, correspond to that studied for the by Hong and Kung [25] (for the sequential setting) and by Irony et al. [24] (for the parallel setting). Algorithms in this class have computational complexity $\Omega(n^3)$. This class includes, among others, the sequential iterative *definition* algorithm, the sequential recursive divide and conquer algorithm based on block partitioning, and parallel algorithms such as Cannon's "*2D*" algorithm [15], the "*2.5D*" algorithm by Solomonik and Demmel [35], and "*3D*" algorithms [1, 26].

Fast Strassen-like matrix multiplication algorithms with base case 2×2 . This class includes algorithms following a structure similar to that of Strassen's [36] and Winograd's variation [38] (which reduces the leading coefficient of the arithmetic complexity reduced from 7 to 6). For further details on Strassen's algorithm we refer the reader to the extended version of this work [18]). Algorithms in this class follow three common steps:

1. **Encoding:** Generate the inputs, of size $n/2 \times n/2$ of seven *sub-problems*, as linear sums of the input matrices;
2. **Recursive multiplications:** Compute (recursively) the seven generated matrix multiplication sub-products;
3. **Decoding:** Computing the entries of the product matrix \mathbf{C} via linear combinations of the output of the seven sub-problems.

Algorithms in this class have algebraic complexity $\mathcal{O}(n^{\log_2 7})$, which is optimal for algorithms with base case 2×2 [38].

Remarkably, the only properties of relevance for the analysis of the I/O complexity of algorithms in these classes are those used in the characterization of the classes themselves.

In this work we consider a general class of *non-uniform*, *non-stationary* hybrid square matrix multiplication algorithms, which allow mixing of schemes from the fast Strassen-like class with algorithms from the standard class. Given an algorithm \mathcal{A} let P denote the problem corresponding to the computation of the product of the input matrices \mathcal{A} and \mathcal{B} . Consider an "*instruction function*" $f_{\mathcal{A}}(P)$, which, given P as input returns either (a) indication regarding the algorithm from the standard class which is to be used to compute P , or (b) indication regarding the fast Strassen-like algorithm to be used to recursively generate seven sub-problems P_1, P_2, \dots, P_7 and the instruction functions $f_{\mathcal{A}}(P_i)$ for each of the seven sub-problems. We refer to the class of non-uniform, non-stationary algorithms which can be characterized by means of such instruction functions as \mathfrak{H} . Algorithms in \mathfrak{H} allow recursive calls to have a different structure, even when they refer to the multiplication of matrices in the same recursive level. E.g., some of the sub-problems with the same size may be computed using algorithms from the standard class while others may be computed using recursive algorithms from the fast class. This class includes, for example, algorithms that optimize for input sizes, (for sizes that are not an integer power of a constant integer).

This corresponds to actual practical scenarios, as the use of Strassen-like algorithms is mostly beneficial for large input size. As the size of the input of the recursively generated sub-problems decreases, the asymptotic benefit of fast algorithms is lost due to the increasing relative impact of the constant multiplicative factor, and algorithms in the standard class exhibit lower *actual* algebraic complexity. For a discussion on such hybrid algorithms and their implementation issues we refer the reader to [20, 23] (sequential model) and [19] (parallel model).

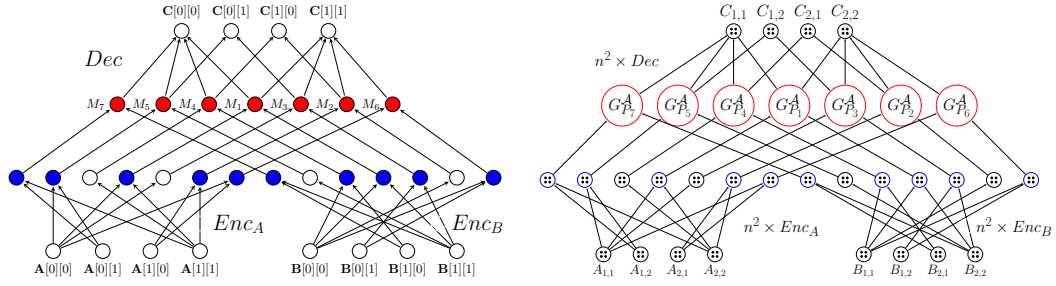
We also consider a sub-class $\mathfrak{U}\mathfrak{H}(n_0)$ of \mathfrak{H} constituted by *uniform, non-stationary* hybrid algorithms which allow mixing of schemes from the fast Strassen-like class for the initial ℓ recursion levels, and then cut the recursion off once the size the generated sub-problems is smaller or equal to a set threshold $n_0 \times n_0$, and switch to using algorithm from the standard class. Algorithms in this class are *uniform*, i.e., sub-problems of the same size are all either recursively computed using a scheme from the fast class, or are all computed using algorithms from the standard class.

4 The CDAG of algorithms in \mathfrak{H}

Let $G^{\mathcal{A}} = (V_{\mathcal{A}}, E_{\mathcal{A}})$ denote the CDAG that corresponds to an algorithm $\mathcal{A} \in \mathfrak{H}$ used for multiplying input matrices $\mathbf{A}, \mathbf{B} \in \mathcal{R}^{n \times n}$. The challenge in the characterization of $G^{\mathcal{A}}$ comes from the fact that rather than considering to a single algorithm, we want to characterize the CDAG corresponding to the class \mathfrak{H} . Further, the class \mathfrak{H} is composed of a rich variety of vastly different and irregular algorithm. Despite such variety, we show a general template for the construction of $G^{\mathcal{A}}$ and we identify some of its properties which crucially hold *regardless* of the implementation details of \mathcal{A} and, hence, of $G^{\mathcal{A}}$.

Construction. $G^{\mathcal{A}}$ can be obtained by using a recursive construction that mirrors the recursive structure of the algorithm itself. Let P denote the entire matrix multiplication problem computed by \mathcal{A} . Consider the case for which, according to the *instruction function* $f_{\mathcal{A}}(P)$, P is to be computed using an algorithm from the standard class. As we do not fix a specific algorithm, we do not correspondingly have a fixed CDAG. The only feature of interest for the analysis is that, in this case, the CDAG $G^{\mathcal{A}}$ corresponds to the execution of an algorithm from the standard class for input matrices of size $n \times n$.

Consider instead the case for which, according to $f_{\mathcal{A}}(P)$, P is to be computed using an algorithm from the fast class. In the base case for $n = 2$ the problem P is computed without generating any further sub-problems. As an example, we present in Figure 1a the base case for Strassen's original algorithm [36]. If $n > 2$, then $f_{\mathcal{A}}(P)$ specifies the divide and conquer scheme to be used to generate the seven sub-problems P_1, P_2, \dots, P_7 , and the *instruction function* for each of them. The sub-CDAGs of $G^{\mathcal{A}}$ corresponding to each of the seven sub-problems P_i , denoted as $G_{P_i}^{\mathcal{A}}$ are constructed according to $f_{\mathcal{A}}(P_i)$, following recursively the steps discussed previously. $G^{\mathcal{A}}$ can then be constructed by composing the seven sub-CDAGs $G_{P_i}^{\mathcal{A}}$. n^2 disjoint copies of an *encoder sub-CDAG* $Enc_{\mathcal{A}}$ (resp., $Enc_{\mathcal{B}}$) are used to connect the input vertices of $G^{\mathcal{A}}$, which correspond to the values of the input matrix \mathbf{A} (resp., \mathbf{B}) to the appropriate input vertices of the seven sub-CDAGs $G_{P_i}^{\mathcal{A}}$; the output vertices of the sub-CDAGs $G_{P_i}^{\mathcal{A}}$ (which correspond to the outputs of the seven sub-products) are connected to the appropriate output vertices of the entire $G^{\mathcal{A}}$ CDAG using n^2 copies of the decoder sub-CDAG Dec . We present an example of such recursive construction in Figure 1b.



(a) G^A CDAG for base case $n = 2$, using Strassen's algorithm [36] (for details see extended version [18]). (b) Recursive construction of G^A . $A_{i,j}$, $B_{i,j}$ and $C_{i,j}$ denote the block-partition of \mathbf{A} , \mathbf{B} and \mathbf{C} .

■ **Figure 1** Blue vertices represent combinations of the input values from the factor matrices \mathbf{A} and \mathbf{B} used as input values for the seven sub-problems; red vertices represent the output of the seven sub-problems which are used to compute the values of the output matrix \mathbf{C} .

Properties of G^A . While the actual internal structure G^A , and, in particular, the structure of encoder and decoder sub-CDAGs depends on the specific Strassen-like algorithm being used by \mathcal{A} , all versions share some properties of great importance. Let $G(X, Y, E)$ denote an encoder CDAG for a fast multiplication algorithm 2×2 base case, with X (resp., Y) denoting the set of input (resp., output) vertices, and E denoting the set of edges directed from X to Y .

► **Lemma 2** (Lemma 3.3 [30]). *Let $G = (X, Y, E)$ denote an encoder graph for a fast matrix multiplication algorithm with base case 2×2 . There are no two vertices in Y with identical neighbors sets.*

While the correctness of this Lemma can be simply verified by inspection in the case of Strassen's algorithm [36], Lemma 2 generalizes the statement to *all* encoders corresponding to fast matrix multiplication algorithms with base case 2×2 . From Lemma 2 we have:

► **Lemma 3.** *Let $\mathcal{A} \in \mathfrak{H}$ and let P_1 and P_2 be any two sub-problems generated by \mathcal{A} with input size greater than $n_0 \times n_0$, such that P_2 is not recursively generated while computing P_1 and vice versa. Then, the sub-CDAGs of G^A corresponding, respectively, to P_1 and to P_2 are vertex-disjoint.*

The following lemma, originally introduced for Strassen's algorithm in [10] and then generalized for Strassen-like algorithms with base case 2×2 in [30], captures a connectivity property of encoder sub-CDAGs.

► **Lemma 4** (Lemma 3.1 [30]). *Given an encoder CDAG for any Strassen-like algorithm with base case 2×2 , for any subset Y of its output vertices, there exists a subset X of its input vertices, with $\min\{|Y|, 1 + \lceil (|Y| - 1) / 2 \rceil\} \leq |X| \leq |Y|$, such that there exist $|X|$ vertex-disjoint paths connecting the vertices in X to vertices in Y .*

The proofs of Lemma 2 and Lemma 4 are based on an argument originally presented by Hopcroft and Kerr [22]. We refer the reader to [30] for the proofs.

5 Maximal sub-problems and their properties

For an algorithm $\mathcal{A} \in \mathfrak{H}$, let P' denote a sub-problem generated by \mathcal{A} . In our presentation we consider the entire matrix multiplication problem as an *improper* sup-problem generated by \mathcal{A} . Given a sub-problem P' let P'_0, P'_2, \dots, P'_i be the sequence of sub-problems generated

by \mathcal{A} such that P'_{j+1} was recursively generated to compute P'_j for $j = 0, 1, \dots, i-1$, and such that P' was recursively generated to compute P'_i . We refer to the sub-problems in such sequence as the *ancestor sub-problems* of P' . If P' is the entire problem, it has no ancestors.

Towards studying the I/O complexity of algorithms in \mathfrak{H} we focus on the analysis of a particular set of sub-problems.

► **Definition 5** (Maximal Sub-Problems (MSP)). *Let $\mathcal{A} \in \mathfrak{H}$ be an algorithm used to multiply matrices $\mathbf{A}, \mathbf{B} \in \mathcal{R}^{n \times n}$. If $n \leq 2\sqrt{M}$ we say that \mathcal{A} does not generate any Maximal Sub-Problem (MSP).*

Let P_i be a sub-problem generated by \mathcal{A} with input size $n_i \times n_i$, with $n_i \geq 2M$, and such that all its ancestors sub-problems are computed, according to \mathcal{A} using algorithms from the fast class. We say that:

- *P_i is a Type 1 MSP of \mathcal{A} if, according to \mathcal{A} , is computed using an algorithm from the standard class. If the entire problem is solved using an algorithm from the standard class, we say that the entire problem is the unique (improper) Type 1 MSP generated by \mathcal{A} .*
- *P_i is a Type 2 MSP of \mathcal{A} if, according to \mathcal{A} , is computed by generating 7 sub-problems according to the recursive scheme corresponding to an algorithm from the fast (Strassen-like) class, and if the generated sub-problems have input size strictly smaller than $2\sqrt{M} \times 2\sqrt{M}$. If the entire problem uses a recursive algorithm from the fast class to generate 7 sub-problems with input size smaller than $2\sqrt{M} \times 2\sqrt{M}$, we say that the entire problem is the unique, improper, Type 2 MSP generated by \mathcal{A} .*

In the following we denote as ν_1 (resp., ν_2) the number of Type 1 (resp., Type 2) MSPs generated by \mathcal{A} .

Let P_i denote the i -th MSP generated by \mathcal{A} and let $G_{P_i}^{\mathcal{A}}$ denote the corresponding sub-CDAG of $G^{\mathcal{A}}$. We denote as \mathbf{A}_i and \mathbf{B}_i (resp., \mathbf{C}_i) the input factor matrices (resp., the output product matrix) of P_i .

Properties of MSPs and their corresponding sub-CDAGs. By Definition 5, we have that for each pair of distinct MMSPs P_1 and P_2 , P_2 is not recursively generated by \mathcal{A} in order to compute P_1 or vice versa. Hence, by Lemma 3, the sub-CDAGs of $G^{\mathcal{A}}$ that correspond each to one of the MSPs generated by \mathcal{A} are vertex-disjoint.

In order to obtain our I/O lower bound for algorithms in \mathfrak{H} , we characterize properties regarding the minimum dominator size of an arbitrary subset of \mathcal{Y} and \mathcal{Z} .

► **Lemma 6.** *Let $G^{\mathcal{A}}$ be the CDAG corresponding to an algorithm $\mathcal{A} \in \mathfrak{H}$ which admits n_1 Type 1 MSPs. For each Type 1 MSP P_i let \mathcal{Y}_i denote the set of input vertices of the associated sub-CDAG $G_{P_i}^{\mathcal{A}}$ which correspond each to an entry of the input matrices \mathbf{A}_i and \mathbf{B}_i . Further, we define $\mathcal{Y} = \cup_{i=1}^{\nu_1} \mathcal{Y}_i$.*

Let $Y \subseteq \mathcal{Y}$ in $G^{\mathcal{A}}$ such that $|Y \cap \mathcal{Y}_i| = a_i / \sqrt{b_i}$, with $a_i, b_i \in \mathbb{N}$, $a_i \geq b_i$ for $i = 1, 2, \dots, \nu_1$, and such that $b_i = 0$ if and only if $a_i = 0$.¹ Any dominator set D of Y satisfies $|D| \geq \min\{2M, \sum_{i=1}^{\nu_1} a_i / \sqrt{\sum_{i=1}^{\nu_1} b_i}\}$.

► **Lemma 7.** *Let $G^{\mathcal{A}}$ be the CDAG corresponding to an algorithm $\mathcal{A} \in \mathfrak{H}$ which admits n_2 Type 2 MSPs. Further let \mathcal{Z} denote the set of vertices corresponding to the entries of the output matrices of the n_2 Type 2 MSPs. Given any subset $Z \subseteq \mathcal{Z}$ in $G^{\mathcal{A}}$ with $|Z| \leq 4M$, any dominator set D of Z satisfies $|D| \geq |Z|/2$.*

¹ Here we use as convention that $0/0 = 0$.

33:10 The I/O Complexity of Hybrid Algorithms for Square Matrix Multiplication

For each Type 1 MSP P_i generated by \mathcal{A} , with input size² $n_i \times n_i$, we denote as T_i the set of variables whose value correspond to the n_i^3 elementary products $\mathbf{A}_i[j][k]\mathbf{B}_i[k][j]$ for $j, k = 0, 1, \dots, n_i - 1$. Further, we denote as \mathcal{T}_i the set of vertices corresponding to the variables in T_i , and we define $\mathcal{T} = \cup_{i=1}^{\nu_1} \mathcal{T}_i$.

► **Lemma 8.** *For any Type 1 MSPs generated by \mathcal{A} consider $T'_i \subseteq T_i$. Let $\mathcal{Y}'_i^{(\mathbf{A})} \subseteq \mathcal{Y}_i$ (resp., $\mathcal{Y}'_i^{(\mathbf{B})} \subseteq \mathcal{Y}_i$) denote a subset of the vertices corresponding to entries of \mathbf{A}_i (resp., \mathbf{B}_i) which are multiplied in at least one of the elementary products in T'_i . Then any dominator D of the vertices corresponding to T'_i with respect to the the vertices in \mathcal{Y}_i is such that*

$$|D| \geq \max\{|\mathcal{Y}'_i \cap A_i|, |\mathcal{Y}'_i \cap B_i|\}.$$

For the proofs of Lemmas 6, 7 and 8 we refer the reader to the extended version of this work [18]. The proofs are based on the analysis of the combinatorial properties of Strassen-like algorithms, and on the analysis of the *Grigoriev's information flow* of the square matrix multiplication function [21, 31].

6 I/O lower bounds for algorithms in \mathfrak{H} and \mathfrak{UH} (n_0)

► **Theorem 9.** *Let $\mathcal{A} \in \mathfrak{H}$ be an algorithm to multiply two square matrices $\mathbf{A}, \mathbf{B} \in \mathcal{R}^{n \times n}$. If run on a sequential machine with cache of size M and such that up to B memory words stored in consecutive memory locations can be moved from cache to slow memory and vice versa using a single memory operation, \mathcal{A} 's I/O complexity satisfies:*

$$IO_{\mathcal{A}}(n, M, B) \geq \max\{2n^2, c|\mathcal{T}|M^{-1/2}, \nu_2 M\}B^{-1} \quad (1)$$

for $c = 0.38988157484$, where $|\mathcal{T}|$ denotes the total number of internal elementary products computed by the Type 1 MSPs generated by \mathcal{A} and ν_2 denotes the total number of Type 2 MSPs generated by \mathcal{A} .

If run on P processors each equipped with a local memory of size $M < n^2$ and where for each I/O operation it is possible to move up to $B_m \leq M$ words, \mathcal{A} 's I/O complexity satisfies:

$$IO_{\mathcal{A}}(n, M, B_m, P) \geq \max\{c|\mathcal{T}|M^{-1/2}, \nu_2 M\}(PB_m)^{-1} \quad (2)$$

Proof. We prove the result in (1) (resp., (2)) for the case $B = 1$ (resp., $B_m = 1$). The result then trivially generalizes for a generic B (resp., B_m). We first prove the result for the sequential case in in (1). The bound for the parallel case in (2) will be obtained as a simple generalization. For simplicity of presentation, we assume $\sqrt{M} \in \mathbb{N}^+$.

The fact that $IO_{\mathcal{A}}(n, M, 1) \geq 2n^2$ follows trivially from the fact that as in our model the input matrices \mathbf{A} and \mathbf{B} are initially stored in slow memory, it will necessary to move the entire input to the cache at least once using at least $2n^2$ I/O operations. If \mathcal{A} does not generate any MSPs the statement in (1) is trivially verified. In the following, we assume $\nu_1 + \nu_2 \geq 1$.

Let $G^{\mathcal{A}}$ denote the CDAG associated with algorithm \mathcal{A} according to the construction in Section 4. By definition, and from Lemma 3, the $\nu_1 + \nu_2$ sub-CDAGs of $G^{\mathcal{A}}$ corresponding each to one of the MSPs generated by \mathcal{A} are vertex-disjoint. Hence, the \mathcal{T}_i 's are a partition of \mathcal{T} and $|\mathcal{T}| = \sum_{i=1}^{\nu_1} |\mathcal{T}_i|$.

² In general, different Type 1 MSP may have different input sizes

By Definition 5, the MSP generated by \mathcal{A} have input (resp., output) matrices of size greater or equal to $2\sqrt{M} \times 2\sqrt{M}$. Recall that we denote as \mathcal{Z} the set of vertices which correspond to the outputs of the ν_2 Type 2 MSPs, we have $|\mathcal{Z}| \geq 4M\nu_l$.

Let \mathcal{C} be any computation schedule for the sequential execution of \mathcal{A} using a cache of size M . We partition \mathcal{C} into non-overlapping segments $\mathcal{C}_1, \mathcal{C}_2, \dots$ such that during each \mathcal{C}_j either (a) exactly $M^{3/2}$ distinct values corresponding to vertices in \mathcal{T} , denoted as $\mathcal{T}^{(j)}$, are explicitly computed (i.e., not loaded from slow memory), or (b) $4M$ distinct values corresponding to vertices in \mathcal{Z} (denoted as \mathcal{Z}_j) are evaluated for the *first time*. Clearly there are at least $\max\{|\mathcal{T}|/M^{3/2}, \nu_2\}$ such segments. Below we show that the number g_j of I/O operations executed during each \mathcal{C}_j satisfies $g_j \geq cM$ for case (a) and $g_j \geq M$ for case (b), from which the theorem follows.

Case (a): For each Type 1 MSP P_i let $\mathcal{T}_i^{(j)} = \mathcal{T}^{(j)} \cap \mathcal{T}_i$. As the ν_1 sub-CDAGs corresponding each to one of the Type 1 MSPs are vertex-disjoint, so are the sets \mathcal{T}_i . Hence, the $\mathcal{T}_i^{(j)}$'s constitute a partition of $\mathcal{T}^{(j)}$. Let \mathbf{A}_i and \mathbf{B}_i (resp., \mathbf{C}_i) denote the input matrices (resp., output matrix) of P_i with $\mathbf{A}_i, \mathbf{B}_i, \mathbf{C}_i \in \mathcal{R}^{n_i \times n_i}$, and let A_i and B_i (resp., C_i) denote the set of the variables corresponding to the entries of \mathbf{A}_i and \mathbf{B}_i (resp., \mathbf{C}_i). Further, we denote as \mathcal{T}_i the set of values corresponding to the vertices in \mathcal{T}_i . For $r, s = 0, 1, \dots, n_i - 1$, we say that $\mathbf{C}_i[r][s]$ is “*active during \mathcal{C}_j* ” if *any* of the elementary multiplications $\mathbf{A}_i[r][k]\mathbf{B}_i[k][s]$, for $k = 0, 1, \dots, n_i - 1$, correspond to any of the vertices in $\mathcal{T}_i^{(j)}$. Further we say that a $\mathbf{A}_i[r][s]$ (resp., $\mathbf{B}_i[r][s]$) is “*accessed during \mathcal{C}_j* ” if *any* of the elementary multiplications $\mathbf{A}_i[r][s]\mathbf{B}_i[s][k]$ (resp., $\mathbf{A}_i[k][r]\mathbf{B}_i[r][s]$), for $k = 0, 1, \dots, n_i - 1$, correspond to any of the vertices in $\mathcal{T}_i^{(j)}$. Our analysis makes use of the following property of standard matrix multiplication algorithms:

► **Lemma 10** (Loomis-Whitney inequality [24, Lemma 2.2]). *Let $\mathcal{Y}'_{i,\mathbf{A}}$ (resp., $\mathcal{Y}'_{i,\mathbf{B}}$) denote the set of vertices corresponding to the entries of \mathbf{A}_i (resp., \mathbf{B}_i) which are accessed during \mathcal{C}_j , and let \mathcal{Z}'_i denote the set of vertices corresponding to the entries of \mathbf{C}_i which are active during \mathcal{C}_j . Then*

$$|\mathcal{T}_i^{(j)}| \leq \sqrt{|\mathcal{Y}'_{i,\mathbf{A}}||\mathcal{Y}'_{i,\mathbf{B}}||\mathcal{Z}'_i|}. \quad (3)$$

Lemma 10 is a reworked version of a property originally presented by Irony et al. [24, Lemma 2.2], which itself is a consequence of the Loomis-Whitney geometric theorem [29]. We refer the reader to [24, Lemma 2.2] for the proof of Lemma 10.

Let $\mathbf{C}_i[r][s]$ be active during \mathcal{C}_j . In order to compute $\mathbf{C}_i[r][s]$ *entirely* during \mathcal{C}_j (i.e., without using partial accumulation of the summation $\sum_{k=0}^{n_i-1} \mathbf{A}_i[r][k]\mathbf{B}_i[k][s]$), it will be necessary to evaluate all the n_i elementary products $\mathbf{A}_i[r][k]\mathbf{B}_i[k][s]$, for $k = 0, 1, \dots, n_i - 1$, during \mathcal{C}_j itself. Thus, at most $\lfloor |\mathcal{T}_i^{(j)}|/n_i \rfloor$ entries of $\mathbf{C}_i[r][s]$ can be entirely computed during \mathcal{C}_j .

Let $\mathbf{C}_i[r][s]$ denote an entry of \mathbf{C}_i which is active but not entirely computed during \mathcal{C}_j . There are two possible scenarios:

- $\mathbf{C}_i[r][s]$ is computed during \mathcal{C}_j : The computation thus requires for a partial accumulation of $\sum_{k=0}^{n_i-1} \mathbf{A}_i[r][k]\mathbf{B}_i[k][s]$ to have been previously computed and either held in the cache at the beginning of \mathcal{C}_j , or to moved to cache using a **read** I/O operation during \mathcal{C}_j ;
- $\mathbf{C}_i[r][s]$ is not computed during \mathcal{C}_j : As \mathcal{C} is a parsimonious computation, the partial accumulation of $\sum_{k=0}^{n_i-1} \mathbf{A}_i[r][k]\mathbf{B}_i[k][s]$ obtained from the elementary products computed during \mathcal{C}_j must either remain in the cache at the end of \mathcal{C}_j , or be moved to slow memory using a **write** I/O operation during \mathcal{C}_j ;

33:12 The I/O Complexity of Hybrid Algorithms for Square Matrix Multiplication

In both cases, any partial accumulation either held in memory at the beginning (resp., end) of \mathcal{C}_j or read from slow memory to cache (resp., written from cache to slow memory) during \mathcal{C}_j is, by definition, not shared between multiple entries in \mathbf{C}_i .

Let $G_{P_i}^A$ denote the sub-CDAG of G^A corresponding to the Type 1 MSP P_i . In the following, we refer as D'_i to the set of vertices of $G_{P_i}^A$ corresponding to the values of such partial accumulators. For each of the least $|D'_i| = \max\{0, |\mathcal{Z}'_i| - |\mathcal{T}_i^{(j)}|/n_i\}$ entries of \mathbf{C}_i which are active but not entirely computed during \mathcal{C}_j , either one of the entries of the cache must be occupied at the beginning of \mathcal{C}_j , or one I/O operation is executed during \mathcal{C}_j . Let $D' = \cup_{i=1}^{\nu_1} D'_i$. As, by Lemma 3, the sub-CDAGs corresponding to the ν_1 Type 1 MSPs are vertex-disjoint, so are the the sets D'_i . Let $\mathcal{Z}' = \sum_{i=1}^{\nu_1} |\mathcal{Z}'_i|$. We have:

$$|D'| = \sum_{i=1}^{\nu_1} |D'_i| = \sum_{i=1}^{\nu_1} \max\{0, |\mathcal{Z}'_i| - |\mathcal{T}_i^{(j)}|/n_i\} \geq |\mathcal{Z}'| - |\mathcal{T}^{(j)}|/2\sqrt{M}, \quad (4)$$

where the last passage follows from the fact that, by Definition 5, $n_i \geq 2M$.

From Lemma 10, the set of vertices $\mathcal{Y}'_{i,\mathbf{A}}$ (resp., $\mathcal{Y}'_{i,\mathbf{B}}$) which correspond to entries of \mathbf{A}_i (resp., \mathbf{B}_i) which are accessed during \mathcal{C}_j satisfies $|\mathcal{Y}'_{i,\mathbf{A}}||\mathcal{Y}'_{i,\mathbf{B}}| \geq |\mathcal{T}_i^{(j)}|^2/|\mathcal{Z}'_i|$. Hence, at least $|\mathcal{Y}'_{i,\mathbf{A}}| + |\mathcal{Y}'_{i,\mathbf{B}}| \geq 2|\mathcal{T}_i^{(j)}|/\sqrt{|\mathcal{Z}'_i|}$ entries from the input matrices of P_i are accessed during \mathcal{C}_j . Let \mathcal{Y} denote the set of vertices corresponding to the entries of the input matrices $\mathbf{A}_i, \mathbf{B}_i$ of P_i . From Lemma 8 we have that there exists a set $\mathcal{Y}'_i \subseteq \mathcal{Y}_i$, with $|\mathcal{Y}'_i| \geq \max\{|\mathcal{Y}'_{i,\mathbf{A}}|, |\mathcal{Y}'_{i,\mathbf{B}}|\} \geq |\mathcal{T}_i^{(j)}|/\sqrt{|\mathcal{Z}'_i|}$, such that the vertices in \mathcal{Y}'_i are connected by vertex-disjoint pats to the vertices in $\mathcal{T}_i^{(j)}$. Let $Y = \cup_{i=1}^{\nu_1} \mathcal{Y}'_i$. As, by Lemma 3, the sub-CDAGs corresponding to the ν_1 Type 1 MSPs are vertex-disjoint, so are the the sets \mathcal{Y}'_i for $i = 1, 2, \dots, \nu_1$. Hence

$$|Y| = \sum_{i=1}^{\nu_1} |\mathcal{Y}'_i| \geq \sum_{i=1}^{\nu_1} \frac{|\mathcal{T}_i^{(j)}|}{\sqrt{|\mathcal{Z}'_i|}}.$$

From Lemma 6 any dominator D_Y of Y , must be such that

$$|D_Y| \geq \min\left\{2M, \frac{\sum_{i=1}^{\nu_1} |\mathcal{T}_i^{(j)}|}{\sum_{i=1}^{\nu_1} \sqrt{|\mathcal{Z}'_i|}}\right\} = \min\left\{2M, \frac{|\mathcal{T}^{(j)}|}{\sqrt{|\mathcal{Z}'|}}\right\}.$$

Hence, we can conclude that any dominator D'' of $\mathcal{T}^{(j)}$ must be such that

$$|D''| \geq \min\{2M, |\mathcal{T}^{(j)}|/\sqrt{|\mathcal{Z}'|}\}. \quad (5)$$

Consider the set D of vertices of G_A corresponding to the at most M values stored in the cache at the beginning of \mathcal{C}_j and to the at most g_j values loaded into the cache from the slow memory (resp., written into the slow memory from the cache) during \mathcal{C}_j by means of a **read** (resp., **write**) I/O operation. Clearly, $|D| \leq M + g_j$.

In order for the $M^{3/2}$ values from $\mathcal{T}^{(j)}$ to be computed during \mathcal{C}_j there must be no path connecting any vertex in $\mathcal{T}^{(j)}$, and, hence, Y , to any input vertex of G^A which does not have at least one vertex in D , that is D has to admit a subset $D'' \subseteq D$ such that D'' is a *dominator set* of $\mathcal{T}^{(j)}$. Note that, as the values corresponding to vertices in $\mathcal{T}^{(j)}$ are actually computed during \mathcal{C}_j (i.e., not loaded from memory using a **read** I/O operation), D'' does not include vertices in $\mathcal{T}^{(j)}$ itself. Further, as motivated in the previous discussion, D must include all the vertices in the set D' corresponding to values of partial accumulators of the active output values of Type 1 MSPs during \mathcal{C}_j .

By construction, D' and D'' are vertex-disjoint. Hence, from (4) and (5) we have:

$$|D| \geq |D'| + |D''| \geq |\mathcal{Z}'| - |\mathcal{T}^{(j)}|/2\sqrt{M} + \min\{2M, |\mathcal{T}^{(j)}|/\sqrt{|\mathcal{Z}'|}\}.$$

As, by construction, $|\mathcal{T}^{(j)}| = M^{3/2}$, we have:

$$|D| > |\mathcal{Z}'| - M/2 + \min\{2M, M^{3/2}/\sqrt{|\mathcal{Z}'|}\}. \quad (6)$$

By studying its derivative after opportunely accounting for the minimum, we have that (6) is minimized for $|\mathcal{Z}'| = 2^{-2/3}M$. Hence we have: $|D| > 2^{-2/3}M + 2^{1/3}M^{3/2} - M/2 = 1.38988157484M$. Whence $|D| \leq M + g_j$, which implies $g_j \geq |D| - M > 0.38988157484M$, as stated above.

Case (b): In order for the $4M$ values from \mathcal{Z}_j to be computed during \mathcal{C}_j there must be no path connecting any vertex in \mathcal{Z}_j to any input vertex of $G_{\mathcal{A}}$ which does not have at least one vertex in D_j , that is D_j has to be a *dominator set* of \mathcal{Z}_j . From Lemma 7, any dominator set D of any subset $Z \subseteq \mathcal{Z}$ with $|Z| \leq 4M$ satisfies $|D| \geq |Z|/2$, whence $M + g_i \geq |D_i| \geq |\mathcal{Z}_j|/2 = 2M$, which implies $g_j \geq M$ as stated above. This concludes the proof for the sequential case in (1).

The proof for the bound for the parallel model in (2), follows from the observation that at least one of the P processors, denoted as P^* , must compute at least $|\mathcal{T}|/P$ values corresponding to vertices in \mathcal{T} or $|\mathcal{Z}|/P$ values corresponding to vertices in \mathcal{Z} (or both). The bound follows by applying the same argument discussed for the sequential case to the computation executed by P^* . ◀

Note that if \mathcal{A} is such that the product is entirely computed using an algorithm from the standard class (resp., a fast matrix multiplication algorithm), the bounds of Theorem 9 corresponds asymptotically to the results in [25] for the sequential case, and in [24] for the parallel case (resp., the results in [10]).

For the sub-class of uniform, non stationary algorithms $\mathfrak{U}\mathfrak{H}(n_0)$, given the values of n , M and n_0 is possible to compute a closed form expression for the values of ν_1, ν_2 and $|\mathcal{T}|$. Then, by applying Theorem 9 we have:

► **Theorem 11.** *Let $\mathcal{A} \in \mathfrak{U}\mathfrak{H}(n_0)$ be an algorithm to multiply two square matrices $\mathbf{A}, \mathbf{B} \in \mathcal{R}^{n \times n}$. If run on a sequential machine with cache of size M and such that up to B memory words stored in consecutive memory locations can be moved from cache to slow memory and vice versa using a single memory operation, \mathcal{A} 's I/O complexity satisfies:*

$$IO_{\mathcal{A}}(n, M, B) \geq \max\{2n^2, \left(\frac{n}{\max\{n_0, 2\sqrt{M}\}}\right)^{\log_2 7} \left(\max\left\{1, \frac{n_0}{2\sqrt{M}}\right\}\right)^3 M\} B^{-1} \quad (7)$$

If run on P processors each equipped with a local memory of size $M < n^2$ and where for each I/O operation it is possible to move up to B_m memory words, \mathcal{A} 's I/O complexity satisfies:

$$IO_{\mathcal{A}}(n, M, B_m, P) \geq \left(\frac{n}{\max\{n_0, 2\sqrt{M}\}}\right)^{\log_2 7} \left(\max\left\{1, \frac{n_0}{2\sqrt{M}}\right\}\right)^3 \frac{M}{PB_m}. \quad (8)$$

Proof. The proof follows by bounding the values ν_1, ν_2 and $|\mathcal{T}|$ for $\mathcal{A} \in \mathfrak{U}\mathfrak{H}(n_0)$, and by applying the general result from Theorem 9. To simplify the presentation, in the following we assume that the values n, n_0 and \sqrt{M} are powers of two. If that is not the case, the theorem holds with some minor adjustments to the constant multiplicative factor.

Let let i be the smallest value in \mathbb{N} such that $n/2^i = \max\{n_0, 2\sqrt{M}\}$. By definition of \mathfrak{H} , at each of the i recursive levels \mathcal{A} generates 7^i sub-problems of size $n/2^i$.

- If $n_0 > 2\sqrt{M}$, \mathcal{A} generates $\nu_1 = 7^i = 7^{\log_2 n/n_0} = (n/n_0)^{\log_2 7}$ Type 1 MSP each with input size $n_0 \times n_0$. As, by Definition 5, the Type 1 MSP are input-disjoint we have $|\mathcal{T}| = \nu_1 n_0^3 = (n/n_0)^{\log_2 7} n_0^3$.

33:14 The I/O Complexity of Hybrid Algorithms for Square Matrix Multiplication

- Otherwise, if $n_0 \leq 2\sqrt{M}$, \mathcal{A} generates $\nu_2 = 7^i = 7^{\log_2 n/2\sqrt{M}} = \left(n/2\sqrt{M}\right)^{\log_2 7}$ Type 2 MSP each with input size $2\sqrt{M} \times 2\sqrt{M}$.

The statement then follows by applying the result in Theorem 9. ◀

The constants terms in (7) and (8) hold under the assumption that n, n_0 and \sqrt{M} are powers of two. If that is not the case the statement holds with minor adjustments to said constant factors.

Theorem 11 extends the result by Scott [34] by expanding the class of hybrid matrix multiplication algorithms being considered (e.g., it does not limit the class of standard matrix multiplication to the divide and conquer algorithm based on block-partitioning), and by removing the assumption that no intermediate value may be recomputed.

On the tightness of the bound. An opportune composition of the cache-optimal version of the Strassen's algorithm [36] (as discussed in [5]) with the standard cache-optimal divide and conquer algorithm for square matrix multiplication based on block-partitioning [15] leads to a sequential hybrid algorithms in \mathfrak{H} (resp., $\mathfrak{UH}(n_0)$) whose I/O cost asymptotically matches the I/O complexity lower bounds in Theorem 9 (1) (resp., Theorem 11 (7)).

Parallel algorithms in \mathfrak{H} (resp., $\mathfrak{UH}(n_0)$) asymptotically matching the I/O lower bounds in for the parallel case in Theorem 9 (2) (resp., Theorem 11 (8)) can be obtained by composing the communication avoiding version of Strassen's algorithm by Ballard et al. [5] with the communication avoiding “2.5” standard algorithm by Solomonik and Demmel [35].

Hence, the lower bounds in Theorem 9 and Theorem 11 are asymptotically tight and the mentioned algorithms from \mathfrak{H} and $\mathfrak{UH}(n_0)$ whose I/O cost asymptotically match the lower bounds are indeed I/O optimal. Further, as the mentioned I/O optimal algorithms from \mathfrak{H} and $\mathfrak{UH}(n_0)$ do not recompute any intermediate value, we can conclude that using recomputation may lead to at most a constant factor reduction of the I/O cost of hybrid algorithms in \mathfrak{H} and $\mathfrak{UH}(n_0)$. Note that the replication of the input used by the mentioned algorithms as recomputation it should be considered as *repeated access* to the input values, and not as recomputation.

Generalization to fast matrix multiplication model with base other than 2×2 . The general statement of Theorem 9 can be extended by enriching \mathfrak{H} to include any fast Strassen-like algorithm with base case other than 2×2 provided that the associated encoder CDAG satisfies properties equivalent to those expressed by Lemma 3 (i.e., the input disjointedness of the sub-problems generated at each recursive step) and Lemma 4 (i.e., the connectivity between input and output of the encoder CDAGs via vertex disjoint paths) for the 2×2 base. If these properties hold, so does the general structure of Theorem 9, given an opportune adjustment of the definition of maximal sub-problem.

7 Conclusion

This work contributed to the characterization of the I/O complexity of hybrid matrix multiplication algorithms combining fast Strassen-like algorithms with standard algorithms. We established asymptotically tight lower bounds that hold even when recomputation is allowed. The generality of the technique used for the analysis makes it promising for the analysis of other hybrid recursive algorithms, e.g., hybrid algorithms for integer multiplication [11].

References

- 1 Alok Aggarwal, Bowen Alpern, Ashok Chandra, and Marc Snir. A model for hierarchical memory. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 305–314. ACM, 1987.
- 2 Alok Aggarwal, Jeffrey Vitter, et al. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988.
- 3 G. Ballard, J. Demmel, O. Holtz, B. Lipshitz, and O. Schwartz. Graph expansion analysis for communication costs of fast rectangular matrix multiplication. In *Design and Analysis of Algorithms*, pages 13–36. Springer, 2012.
- 4 Grey Ballard, James Demmel, Olga Holtz, Benjamin Lipshitz, and Oded Schwartz. Brief announcement: strong scaling of matrix multiplication algorithms and memory-independent communication lower bounds. In *Proceedings of the twenty-fourth annual ACM symposium on Parallelism in algorithms and architectures*, pages 77–79. ACM, 2012.
- 5 Grey Ballard, James Demmel, Olga Holtz, Benjamin Lipshitz, and Oded Schwartz. Communication-optimal parallel algorithm for Strassen’s matrix multiplication. In *Proceedings of the twenty-fourth annual ACM symposium on Parallelism in algorithms and architectures*, pages 193–204. ACM, 2012.
- 6 Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz. Communication-optimal parallel and sequential Cholesky decomposition. *SIAM Journal on Scientific Computing*, 32(6):3495–3523, 2010.
- 7 Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz. Minimizing communication in numerical linear algebra. *SIAM Journal on Matrix Analysis and Applications*, 32(3):866–901, 2011.
- 8 Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz. Graph expansion and communication costs of fast matrix multiplication. *Journal of the ACM (JACM)*, 59(6):32, 2012.
- 9 Sandeep N Bhatt, Gianfranco Bilardi, and Geppino Pucci. Area-time tradeoffs for universal VLSI circuits. *Theoretical Computer Science*, 408(2-3):143–150, 2008.
- 10 Gianfranco Bilardi and Lorenzo De Stefani. The I/O complexity of Strassen’s matrix multiplication with recomputation. In *Workshop on Algorithms and Data Structures*, pages 181–192. Springer, 2017.
- 11 Gianfranco Bilardi and Lorenzo De Stefani. The I/O complexity of Toom-Cook integer multiplication. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2034–2052. SIAM, 2019.
- 12 Gianfranco Bilardi and Enoch Peserico. A characterization of temporal locality and its portability across memory hierarchies. In *International Colloquium on Automata, Languages, and Programming*, pages 128–139. Springer, 2001.
- 13 Gianfranco Bilardi and Franco P Preparata. Horizons of parallel computation. *Journal of Parallel and Distributed Computing*, 27(2):172–182, 1995.
- 14 Gianfranco Bilardi and Franco P Preparata. Processor—Time Tradeoffs under Bounded-Speed Message Propagation: Part II, Lower Bounds. *Theory of Computing Systems*, 32(5):531–559, 1999.
- 15 Lynn Elliot Cannon. *A cellular computer to implement the Kalman filter algorithm*. PhD thesis, Montana State University-Bozeman, College of Engineering, 1969.
- 16 National Research Council et al. *Getting up to speed: The future of supercomputing*. National Academies Press, 2005.
- 17 Lorenzo De Stefani. *On space constrained computations*. PhD thesis, University of Padova, 2016.
- 18 Lorenzo De Stefani. The I/O complexity of hybrid algorithms for square matrix multiplication. *arXiv preprint*, 2019. [arXiv:1904.12804](https://arxiv.org/abs/1904.12804).

33:16 The I/O Complexity of Hybrid Algorithms for Square Matrix Multiplication

- 19 Frédéric Desprez and Frédéric Suter. Impact of mixed-parallelism on parallel implementations of the Strassen and Winograd matrix multiplication algorithms. *Concurrency and Computation: practice and experience*, 16(8):771–797, 2004.
- 20 Craig C Douglas, Michael Heroux, Gordon Sliselman, and Roger M Smith. GEMMW: a portable level 3 BLAS Winograd variant of Strassen’s matrix-matrix multiply algorithm. *Journal of Computational Physics*, 110(1):1–10, 1994.
- 21 Dmitrii Yur’evich Grigor’ev. Application of separability and independence notions for proving lower bounds of circuit complexity. *Zapiski Nauchnykh Seminarov POMI*, 60:38–48, 1976.
- 22 John E Hopcroft and Leslie R Kerr. On minimizing the number of multiplications necessary for matrix multiplication. *SIAM Journal on Applied Mathematics*, 20(1):30–36, 1971.
- 23 Steven Huss-Lederman, Elaine M Jacobson, Jeremy R Johnson, Anna Tsao, and Thomas Turnbull. Implementation of Strassen’s algorithm for matrix multiplication. In *Supercomputing’96: Proceedings of the 1996 ACM/IEEE Conference on Supercomputing*, pages 32–32. IEEE, 1996.
- 24 Dror Irony, Sivan Toledo, and Alexander Tiskin. Communication lower bounds for distributed-memory matrix multiplication. *Journal of Parallel and Distributed Computing*, 64(9):1017–1026, 2004.
- 25 Hong Jia-Wei and Hsiang-Tsung Kung. I/O complexity: The red-blue pebble game. In *Proceedings of the thirteenth annual ACM symposium on Theory of computing*, pages 326–333. ACM, 1981.
- 26 S Lennart Johnsson. Minimizing the communication time for matrix multiplication on multiprocessors. *Parallel Computing*, 19(11):1235–1257, 1993.
- 27 Richard R. Koch, F. T. Leighton, Bruce M. Maggs, Satish B. Rao, Arnold L. Rosenberg, and Eric J. Schwabe. Work-preserving Emulations of Fixed-connection Networks. *J. ACM*, 44(1):104–147, January 1997. doi:10.1145/256292.256299.
- 28 François Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th international symposium on symbolic and algebraic computation*, pages 296–303. ACM, 2014.
- 29 Lynn H Loomis and Hassler Whitney. An inequality related to the isoperimetric inequality. *Bulletin of the American Mathematical Society*, 55(10):961–962, 1949.
- 30 Roy Nissim and Oded Schwartz. Revisiting the I/O-Complexity of Fast Matrix Multiplication with Recomputations. In *Proceedings of the 33rd IEEE International Parallel and Distributed Processing Symposium*, pages 714–716, 2019.
- 31 J. E. Savage. *Models of Computation: Exploring the Power of Computing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1997.
- 32 John E Savage. Extending the Hong-Kung model to memory hierarchies. In *International Computing and Combinatorics Conference*, pages 270–281. Springer, 1995.
- 33 Jacob Scott, Olga Holtz, and Oded Schwartz. Matrix multiplication I/O-complexity by path routing. In *Proceedings of the 27th ACM symposium on Parallelism in Algorithms and Architectures*, pages 35–45. ACM, 2015.
- 34 Jacob N Scott. *An I/O-Complexity Lower Bound for All Recursive Matrix Multiplication Algorithms by Path-Routing*. PhD thesis, UC Berkeley, 2015.
- 35 Edgar Solomonik and James Demmel. Communication-optimal parallel 2.5 D matrix multiplication and LU factorization algorithms. In *European Conference on Parallel Processing*, pages 90–109. Springer, 2011.
- 36 Volker Streets. Gaussian elimination is not optimal. *numerical mathematics*, 13(4):354–356, 1969.
- 37 Y. D. Burago V. A. Zalgaller, A. B. Sossinsky. Geometric Inequalities. *The American Mathematical Monthly*, 96(6):544–546, 1989.
- 38 Shmuel Winograd. On multiplication of 2×2 matrices. *Linear algebra and its applications*, 4(4):381–388, 1971.

Accurate MapReduce Algorithms for k -Median and k -Means in General Metric Spaces

Alessio Mazzetto¹

Department of Computer Science, Brown University, Providence, USA
alessio_mazzetto@brown.edu

Andrea Pietracaprina

Department of Information Engineering, University of Padova, Padova, Italy
andrea.pietracaprina@unipd.it

Geppino Pucci

Department of Information Engineering, University of Padova, Padova, Italy
geppino.pucci@unipd.it

Abstract

Center-based clustering is a fundamental primitive for data analysis and becomes very challenging for large datasets. In this paper, we focus on the popular k -median and k -means variants which, given a set P of points from a metric space and a parameter $k < |P|$, require to identify a set S of k centers minimizing, respectively, the sum of the distances and of the squared distances of all points in P from their closest centers. Our specific focus is on general metric spaces, for which it is reasonable to require that the centers belong to the input set (i.e., $S \subseteq P$). We present coresets-based 3-round distributed approximation algorithms for the above problems using the MapReduce computational model. The algorithms are rather simple and obviously adapt to the intrinsic complexity of the dataset, captured by the doubling dimension D of the metric space. Remarkably, the algorithms attain approximation ratios that can be made arbitrarily close to those achievable by the best known polynomial-time sequential approximations, and they are very space efficient for small D , requiring local memory sizes substantially sublinear in the input size. To the best of our knowledge, no previous distributed approaches were able to attain similar quality-performance guarantees in general metric spaces.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Theory of computation → Facility location and clustering; Theory of computation → MapReduce algorithms

Keywords and phrases Clustering, k -median, k -means, MapReduce, Coreset

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.34

Funding This work was supported, in part, by the University of Padova under grant SID2017 and by MIUR, the Italian Ministry of Education, University and Research, under grant PRIN *AHeAD: efficient Algorithms for HArnessing networked Data* and grant L. 232 “*Dipartimenti di Eccellenza*”.

1 Introduction

Clustering is a fundamental primitive in the realms of data management and machine learning, with applications in a large spectrum of domains such as database search, bioinformatics, pattern recognition, networking, operations research, and many more [15]. A prominent clustering subspecies is *center-based clustering* whose goal is to partition a set of data items into k groups, where k is an input parameter, according to a notion of similarity, captured by a given measure of closeness to suitably chosen representatives, called centers. There is

¹ This work was done while the author was a graduate student at University of Padova.



a vast and well-established literature on sequential strategies for different instantiations of center-based clustering [3]. However, the explosive growth of data that needs to be processed often rules out the use of these sequential strategies, which are often impractical on large data sets, due to their time and space requirements. Therefore, it is of paramount importance to devise efficient distributed clustering strategies tailored to the typical computational frameworks for big data processing, such as MapReduce [20].

In this paper, we focus on the k -median and k -means clustering problems. Given a set P of points in a general metric space and a positive integer $k \leq |P|$, the k -median (resp., k -means) problem requires to find a subset $S \subseteq P$ of k points, called *centers*, so that the sum of all distances (resp., square distances) between the points of P to their closest center is minimized. Once S is determined, the association of each point to its closest center naturally defines a clustering of P . While scarcely meaningful for general metric spaces, for Euclidean spaces, the widely studied *continuous* variant of these two problems removes the constraint that S is a subset of P , hence allowing a much richer choice of centers from the entire space. Along with k -center, which requires to minimize the maximum distance of a point to its closest center, k -median and k -means are the most popular instantiations of center-based clustering, whose efficient solution in the realm of big data has attracted vast attention in the recent literature [10, 5, 6, 24, 7]. One of the reference models for big data computing, also adopted in most of the aforementioned works, is MapReduce [9, 22, 20], where a set of processors with limited-size local memories process data in a sequence of parallel rounds. Efficient MapReduce algorithms should aim at minimizing the number of rounds while using substantially sublinear local memory.

A natural approach to solving large instances of combinatorial optimization problems relies on the extraction of a much smaller “summary” of the input instance, often dubbed *coreset* in the literature [14], which embodies sufficient information to enable the extraction of a good approximate solution of the whole input. This approach is profitable whenever the (time and space) resources needed to compute the coreset are considerably lower than those required to compute a solution by working directly on the input instance. Coresets with different properties have been studied in the literature to solve different variants of the aforementioned clustering problems [21].

The main contributions of this paper are novel coreset-based space/round-efficient MapReduce algorithms for k -median and k -means.

1.1 Related work

The k -median and k -means clustering problems in general metric spaces have been extensively studied, and constant approximation algorithms are known for both problems [3]. In recent years, there has been growing interest in the development of distributed algorithms to attack these problems in the big data scenario (see [24] and references therein). While straightforward parallelizations of known iterative sequential strategies tend to be inefficient due to high round complexity, the most relevant efforts to date rely on distributed constructions of coresets of size much smaller than the input, upon which a sequential algorithm is then run to obtain the final solution. Ene et al. [10] present a randomized MapReduce algorithm which computes a coreset for k -median of size $O(k^2|P|^\delta)$ in $O(1/\delta)$ rounds, for any $\delta \in (0, 1)$. By using an α -approximation algorithm on this coreset, a weak $(10\alpha + 3)$ -approximate solution is obtained. In the paper, the authors claim that their approach extends also to the k -means problem, but do not provide the analysis. For this latter problem, in [5] a parallelization of the popular k -means++ algorithm by [1] is presented, which builds an $O(k \log |P|)$ -size coreset for k -means in $O(\log |P|)$ rounds. By running an α -approximation algorithm on

the coresets, the returned solution features an $O(\alpha)$ approximation ratio. A randomized MapReduce algorithm for k -median has been recently presented in [24], where the well known local-search PAM algorithm [19] is employed to extract a small family of possible solutions from random samples of the input. A suitable refinement of the best solution in the family is then returned. While extensive experiments support the effectiveness of this approach in practice, no tight theoretical analysis of the resulting approximation quality is provided.

In the continuous setting, Balcan et al. [6] present randomized 2-round algorithms to build coresets in \mathbb{R}^d of size $O\left(\frac{kd}{\epsilon^2} + Lk\right)$ for k -median, and $O\left(\frac{kd}{\epsilon^4} + Lk \log(Lk)\right)$ for k -means, for any choice of $\epsilon \in (0, 1)$, where the computation is distributed among L processing elements. By using an α -approximation algorithm on the coresets, the overall approximation factor is $\alpha + O(\epsilon)$. For k -means, a recent improved construction yields a coreset which is a factor $O(\epsilon^2)$ smaller and features very fast distributed implementation [4]. It is not difficult to show that a straightforward adaptation of these algorithms to general spaces (hence in a non-continuous setting) would yield $(c \cdot \alpha + O(\epsilon))$ -approximations, with $c \geq 2$, thus introducing a non-negligible gap with respect to the quality of the best sequential approximations.

Finally, it is worth mentioning that there is a rich literature on sequential coreset constructions for k -median and k -means, which mostly focus on the continuous case in Euclidean spaces [11, 14, 13, 23, 8]. We do not review the results in these works since our focus is on distributed algorithms in general metric spaces. We also note that the recent work of [16] addresses the construction of coresets for k -median and k -means in general metric spaces, where the coreset sizes are expressed as a function of the doubling dimension. However, their construction strategy is rather complex and it is not clear how to adapt it to the distributed setting.

1.2 Our contribution

We devise new distributed coreset constructions and show how to employ them to yield accurate space-efficient 3-round MapReduce algorithms for k -median and k -means. Our coresets are built in a *composable* fashion [17] in the sense that they are obtained as the union of small local coresets computed in parallel (in 2 MapReduce rounds) on distinct subsets of a partition of the input. The final solution is obtained by running a sequential approximation algorithm on the coreset in the third MapReduce round. The memory requirements of our algorithms are analyzed in terms of the desired approximation guarantee, and of the *doubling dimension* D of the underlying metric space, a parameter which generalizes the dimensionality of Euclidean spaces to general metric spaces and is thus related to the increasing difficulty of spotting good clusterings as the parameter D grows.

Let α denote the best approximation ratio attainable by a sequential algorithm for either k -median or k -means on general metric spaces. Our main results are 3-round $(\alpha + O(\epsilon))$ -approximation MapReduce algorithms for k -median and k -means, which require $O(|P|^{2/3} k^{1/3} (c/\epsilon)^{2D} \log^2 |P|)$ local memory, where $c > 0$ is a suitable constant that will be specified in the analysis, and $\epsilon \in (0, 1)$ is a user-defined precision parameter. To the best of our knowledge, these are the first MapReduce algorithms for k -median and k -means in general metric spaces which feature approximation guarantees that can be made arbitrarily close to those of the best sequential algorithms, and run in few rounds using local space substantially sublinear for low-dimensional spaces. In fact, prior to our work existing MapReduce algorithms for k -median and k -means in general metric spaces either exhibited approximation factors much larger than α [10, 5], or missed a tight theoretical analysis of the approximation factor [24].

Our algorithms revolve around novel coreset constructions somehow inspired by those proposed in [14] for Euclidean spaces. As a fundamental tool, the constructions make use of a procedure that, starting from a set of points P and a set of centers C , produces a (not much) larger set C' such that for any point $x \in P$ its distance from C' is significantly smaller than its distance from C . Simpler versions of our constructions can also be employed to attain 2-round MapReduce algorithms for the continuous versions of the two problems, featuring $\alpha + O(\epsilon)$ approximation ratios. While similar approximation guarantees have already been achieved in the literature using more space-efficient but randomized coreset constructions [6, 4], this result provides evidence of the general applicability of our novel approach.

Finally, we want to point out that a very desirable feature of our MapReduce algorithms is that they do not require a priori knowledge of the doubling dimension D and, in fact, it is easily shown that they adapt to the dimensionality of the dataset which, in principle, can be much lower than the one of the underlying space.

Organization of the paper. The rest of the paper is organized as follows. Section 2 contains a number of preliminary concepts, including various properties of coresets that are needed to achieve our results. Section 3 presents our novel coreset constructions for k -median (Subsection 3.2) and k -means (Subsection 3.3). Based on these constructions, Subsection 3.4 derives the MapReduce algorithms for the two problems. Finally, Section 4 offers some concluding remarks.

2 Preliminaries

Let \mathcal{M} be a metric space with distance function $d(\cdot, \cdot)$. We define the *ball of radius r centered at x* as the set of points at distance at most r from x . The *doubling dimension* of \mathcal{M} is the smallest integer D such that for any r and $x \in \mathcal{M}$, the ball of radius r centered at x can be covered by at most 2^D balls of radius $r/2$ centered at points of \mathcal{M} . Let $x \in \mathcal{M}$ and $Y \subseteq \mathcal{M}$. We define $d(x, Y) = \min_{y \in Y} d(x, y)$ and $x^Y = \arg \min_{y \in Y} d(x, y)$. A set of points $P \subseteq \mathcal{M}$ can be weighted by assigning a positive integer $w(p)$ to each $p \in P$. In this case, we will use the notation P_w (note that an unweighted set of points can be considered weighted with unitary weights). Let X_w and Y be two subsets of \mathcal{M} . We define $\nu_{X_w}(Y) = \sum_{x \in X_w} w(x)d(x, Y)$ and $\mu_{X_w}(Y) = \sum_{x \in X_w} w(x)d(x, Y)^2$. The values $\nu_{X_w}(Y)$ and $\mu_{X_w}(Y)$ are also referred to as *costs*.

In the *k -median problem* (resp., *k -means problem*), we are given in input an instance $\mathcal{I} = (P, k)$, with $P \subseteq \mathcal{M}$ and k a positive integer. A set $S \subseteq P$ is a solution of \mathcal{I} if $|S| \leq k$. The objective is to find the solution S with minimum cost $\nu_P(S)$ (resp., $\mu_P(S)$). Given an instance \mathcal{I} of one of these two problems, we denote with $\text{opt}_{\mathcal{I}}$ its optimal solution. Moreover, for $\alpha \geq 1$, we say that S is an α -*approximate solution* for \mathcal{I} if its cost is within a factor α from the cost of $\text{opt}_{\mathcal{I}}$. In this case, the value α is also called approximation factor. An α -*approximation algorithm* computes an α -approximate solution for any input instance. The two problems are immediately generalized to the case of weighted instances (P_w, k) . In fact, all known approximations algorithms can be straightforwardly adapted to handle weighted instances keeping the same approximation quality.

Observe that the squared distance does not satisfy the triangle inequality. During the analysis, we will use the following weaker bound.

► **Proposition 2.1.** *Let $x, y, z \in \mathcal{M}$. For every $c > 0$ we have that $d(x, y)^2 \leq (1 + 1/c)d(x, z)^2 + (1 + c)d(z, y)^2$.*

Proof. Let a, b be two real numbers. Since $(a/\sqrt{c} - b \cdot \sqrt{c})^2 \geq 0$, we obtain that $2ab \leq a^2/c + c \cdot b^2$. Hence, $(a + b)^2 \leq (1 + 1/c)a^2 + (1 + c)b^2$. The proof follows since $d(x, y)^2 \leq [d(x, z) + d(z, y)]^2$ by triangle inequality. \blacktriangleleft

A coreset is a small (weighted) subset of the input which summarizes the whole data. The concept of summarization can be captured with the following definition, which is commonly adopted to describe coresets for k -means and k -median (e.g., [14, 11, 16]).

► **Definition 2.2.** A weighted set of points C_w is an ϵ -approximate coreset of an instance $\mathcal{I} = (P, k)$ of k -median (resp., k -means) if for any solution S of \mathcal{I} it holds that $|\nu_P(S) - \nu_{C_w}(S)| \leq \epsilon \cdot \nu_P(S)$ (resp., $|\mu_P(S) - \mu_{C_w}(S)| \leq \epsilon \cdot \mu_P(S)$).

Informally, the cost of any solution is approximately the same if computed from the ϵ -approximate coreset rather than from the full set of points. In the paper we will also make use of the following different notion of coreset (already used in [14, 10]), which upper bounds the aggregate “proximity” of the input points from the coreset as a function of the optimal cost.

► **Definition 2.3.** Let $\mathcal{I} = (P, k)$ be an instance of k -median (resp., k -means). A set of points C_w is an ϵ -bounded coreset of \mathcal{I} if it exists a map $\tau : P \rightarrow C_w$ such that $\sum_{x \in P} d(x, \tau(x)) \leq \epsilon \cdot \nu_P(\text{opt}_{\mathcal{I}})$ (resp., $\sum_{x \in P} d(x, \tau(x))^2 \leq \epsilon \cdot \mu_P(\text{opt}_{\mathcal{I}})$) and for any $x \in C_w$, $w(x) = |\{y \in P : \tau(y) = x\}|$. We say that C_w is weighted according to τ .

The above two kind of coresets are related, as shown in the following two lemmas.

► **Lemma 2.4.** Let C_w be an ϵ -bounded coreset of a k -median instance $\mathcal{I} = (P, k)$. Then C_w is also a ϵ -approximate coreset of \mathcal{I} .

Proof. Let τ be the map of the definition of ϵ -bounded coreset. Let S be a solution of \mathcal{I} . Using triangle inequality, we can easily see that $d(x, S) - d(x, \tau(x)) \leq d(\tau(x), S)$ and $d(\tau(x), S) \leq d(\tau(x), x) + d(x, S)$ for any $x \in P$. Summing over all points in P , we obtain that

$$\nu_P(S) - \sum_{x \in P} d(x, \tau(x)) \leq \nu_{C_w}(S) \leq \sum_{x \in P} d(x, \tau(x)) + \nu_P(S)$$

To conclude the proof, we observe that $\sum_{x \in P} d(x, \tau(x)) \leq \epsilon \cdot \nu_P(\text{opt}_{\mathcal{I}}) \leq \epsilon \cdot \nu_P(S)$. \blacktriangleleft

► **Lemma 2.5.** Let C_w be an ϵ -bounded coreset of a k -means instance $\mathcal{I} = (P, k)$. Then C_w is also a $(\epsilon + 2\sqrt{\epsilon})$ -approximate coreset of \mathcal{I} .

Proof. Let τ be the map of the definition of ϵ -bounded coreset. Let S be a solution of \mathcal{I} . We want to bound the quantity $|\mu_P(S) - \mu_{C_w}(S)| = \sum_{x \in P} |d(x, S)^2 - d(\tau(x), S)^2|$. We rewrite $|d(x, S)^2 - d(\tau(x), S)^2|$ as $[d(x, S) + d(\tau(x), S)] \cdot |d(x, S) - d(\tau(x), S)|$. By triangle inequality, we have that $d(x, S) \leq d(x, \tau(x)) + d(\tau(x), S)$ and $d(\tau(x), S) \leq d(\tau(x), x) + d(x, S)$. By combining these two inequalities, it results that $|d(x, S) - d(\tau(x), S)| \leq d(x, \tau(x))$. Moreover, $d(x, S) + d(\tau(x), S) \leq 2d(x, S) + d(x, \tau(x))$. Hence

$$\begin{aligned} |\mu_P(S) - \mu_{C_w}(S)| &\leq \sum_{x \in P} d(x, \tau(x)) [2d(x, S) + d(x, \tau(x))] \\ &\leq \epsilon \cdot \mu_P(S) + 2 \sum_{x \in P} d(x, \tau(x)) d(x, S) \end{aligned}$$

where we used the fact that $\sum_{x \in P} d(x, \tau(x))^2 \leq \epsilon \cdot \mu_P(\text{opt}_{\mathcal{I}}) \leq \epsilon \cdot \mu_P(S)$. We now want to bound the sum over the products of the two distances. Arguing as in the proof of Proposition 2.1, we can write:

$$2 \sum_{x \in P} d(x, \tau(x))d(x, S) \leq \sqrt{\epsilon} \cdot \sum_{x \in P} d(x, S)^2 + \frac{1}{\sqrt{\epsilon}} \sum_{x \in P} d(x, \tau(x))^2 \leq 2\sqrt{\epsilon} \cdot \mu_P(S)$$

To wrap it up, it results that $|\mu_P(S) - \mu_{C_w}(S)| \leq (\epsilon + 2\sqrt{\epsilon}) \cdot \mu_P(S)$. ◀

In our work, we will build coresets by working in parallel over a partition of the input instance. The next lemma provides known results on the relations between the optimal solution of the whole input points and the optimal solution of a subset of the input points.

► **Lemma 2.6.** *Let $C_w \subseteq P$. Let $\mathcal{I} = (P, k)$ and $\mathcal{I}' = (C_w, k)$. Then: (a) $\nu_{C_w}(\text{opt}_{\mathcal{I}'}) \leq 2\nu_{C_w}(\text{opt}_{\mathcal{I}})$; and (b) $\mu_{C_w}(\text{opt}_{\mathcal{I}'}) \leq 4\mu_{C_w}(\text{opt}_{\mathcal{I}})$.*

Proof. We first prove point (b). Let $X = \{x^{C_w} : x \in \text{opt}_{\mathcal{I}'}\}$. The set X is a solution of \mathcal{I}' . By optimality of $\text{opt}_{\mathcal{I}'}$, we have that $\mu_{C_w}(\text{opt}_{\mathcal{I}'}) \leq \mu_{C_w}(X)$. Also, by triangle inequality, it holds that $\mu_{C_w}(X) \leq \sum_{x \in C_w} w(x) [d(x, \text{opt}_{\mathcal{I}}) + d(x^{\text{opt}_{\mathcal{I}'}})]^2$. We observe that $d(x^{\text{opt}_{\mathcal{I}'}}) \leq d(x, \text{opt}_{\mathcal{I}})$ by definition of X . Thus, we obtain that $\mu_{C_w}(\text{opt}_{\mathcal{I}'}) \leq 4\mu_{C_w}(\text{opt}_{\mathcal{I}})$. The proof of (a) follows the same lines with a factor 2 less since we do not square. ◀

Bounded coresets have the nice property to be *composable*. That is, we can partition the input points into different subsets and compute a bounded coreset separately in each subset: the union of those coresets is a bounded coreset of the input instance. This property, which is formally stated in the following lemma, is crucial to develop efficient MapReduce algorithms for the clustering problems.

► **Lemma 2.7.** *Let $\mathcal{I} = (P, k)$ be an instance of k -median (resp., k -means). Let P_1, \dots, P_L be a partition of P . For $\ell = 1, \dots, L$, let $C_{w,\ell}$ be an ϵ -bounded coreset of $\mathcal{I}_\ell = (P_\ell, k)$. Then $C_w = \cup_{\ell} C_{w,\ell}$ is a 2ϵ -bounded coreset (resp., a 4ϵ -bounded coreset) of \mathcal{I} .*

Proof. We prove the lemma for k -median. The proof for k -means is similar. For $\ell = 1, \dots, L$, let τ_ℓ be the map from P_ℓ to $C_{w,\ell}$ of Definition 2.3. Now, for any $x \in P$, let ℓ be the integer such that $x \in P_\ell$; we define $\tau(x) = \tau_\ell(x)$.

$$\sum_{x \in P} d(x, \tau(x)) \leq \sum_{\ell=1}^L \sum_{x \in P_\ell} d(x, \tau_\ell(x)) \leq \epsilon \sum_{\ell=1}^L \nu_{P_\ell}(\text{opt}_{\mathcal{I}_\ell}) \leq 2\epsilon \cdot \nu_P(\text{opt}_{\mathcal{I}})$$

In the last inequality, we used the fact that $\nu_{P_\ell}(\text{opt}_{\mathcal{I}_\ell}) \leq 2\nu_{P_\ell}(\text{opt}_{\mathcal{I}})$ from Lemma 2.6. ◀

In the paper, we will need the following additional characterization of a representative subset of the input, originally introduced in [14].

► **Definition 2.8.** *Let $\mathcal{I} = (P, k)$ be an instance of k -median (resp., k -means). A set C is said to be an ϵ -centroid set of \mathcal{I} if there exists a subset $X \subseteq C$, $|X| \leq k$, such that $\nu_P(X) \leq (1 + \epsilon)\nu_P(\text{opt}_{\mathcal{I}})$ (resp., $\mu_P(X) \leq (1 + \epsilon)\mu_P(\text{opt}_{\mathcal{I}})$).*

Our algorithms are designed for the *MapReduce* model of computation which has become a de facto standard for big data algorithmics in recent years. A MapReduce algorithm [9, 22, 20] executes in a sequence of parallel *rounds*. In a round, a multiset X of key-value pairs is first transformed into a new multiset X' of key-value pairs by applying a given *map function* (simply called *mapper*) to each individual pair, and then into a final multiset Y

of pairs by applying a given *reduce function* (simply called *reducer*) independently to each subset of pairs of X' having the same key. The model features two parameters, M_L , the *local memory* available to each mapper/reducer, and M_A , the *aggregate memory* across all mappers/reducers.

3 Coresets construction in MapReduce

Our coreset constructions are based on a suitable point selection algorithm called `CoverWithBalls`, somewhat inspired by the exponential grid construction used in [14] to build ϵ -approximate coresets in \mathbb{R}^d for the continuous case. Suppose that we want to build an ϵ -bounded coreset of a k -median instance $\mathcal{I} = (P, k)$ and that a β -approximate solution T for \mathcal{I} is available. A simple approach would be to find a set C_w such that for any x in P there exists a point $\tau(x) \in C$ for which $d(x, \tau(x)) \leq (\epsilon/2\beta) \cdot d(x, T)$. Indeed, if C_w is weighted according to τ , it can be seen that C_w is an ϵ -bounded coreset of \mathcal{I} . The set C_w can be constructed greedily by iteratively selecting an arbitrary point $p \in P$, adding it to C_w , and discarding all points $q \in P$ (including p) for which the aforementioned property holds with $\tau(q) = p$. The construction ends when all points of P are discarded. However, note that the points of P which are already very close to T , say at a distance $\leq R$ for a suitable tolerance threshold R , do not contribute much to $\nu_P(T)$, and so to the sum $\sum_{x \in P} d(x, \tau(x))$. For these points, we can relax the constraint and discard them from P as soon their distance to C_w becomes at most $(\epsilon/2\beta) \cdot R$. This relaxation is crucial to bound the size of the returned set as a function of the doubling dimension of the space. Algorithm `CoverWithBalls` is formally

■ **Algorithm 1** `CoverWithBalls`(P, T, R, ϵ, β).

```

1  $C_w \leftarrow \emptyset$ 
2 while  $P \neq \emptyset$  do
3    $p \leftarrow$  arbitrarily selected point in  $P$ 
4    $C_w \leftarrow C_w \cup \{p\}$ ,  $w(p) \leftarrow 0$ 
5   foreach  $q \in P$  do
6     if  $d(p, q) \leq \epsilon/(2\beta) \max\{R, d(q, T)\}$  then
7       remove  $q$  from  $P$ 
8        $w(p) \leftarrow w(p) + 1$            /* (i.e.  $\tau(q) = p$ , see Lemma 3.1) */
9     end
10  end
11 end
12 return  $C_w$ 

```

described in the pseudocode below. It receives in input two sets of points, P and T , and three positive real parameters R , ϵ , and β , with $\epsilon < 1$ and $\beta \geq 1$ and outputs a weighted set $C_w \subseteq P$ which satisfies the property stated in the following lemma.

► **Lemma 3.1.** *Let C_w be the output of `CoverWithBalls`(P, T, R, ϵ, β). C_w is weighted according to a map $\tau : P \rightarrow C_w$ such that, for any $x \in P$, $d(x, \tau(x)) \leq \epsilon/(2\beta) \max\{R, d(x, T)\}$.*

Proof. For any $x \in P$, we define $\tau(x)$ as the point in C_w which caused the removal of x from P during the execution of the algorithm. The statement immediately follows. ◀

While in principle the size of C_w can be arbitrarily close to $|P|$, the next theorem shows that this is not the case for low dimensional spaces, as a consequence of the fact that there cannot be too many points which are all far from one another. We first need a technical lemma. A set of points X is said to be an r -clique if for any $x, y \in X$, $x \neq y$, it holds that $d(x, y) > r$. We have:

► **Lemma 3.2.** *Let $0 < \epsilon < 1$. Let \mathcal{M} be a metric space with doubling dimension D . Let $X \subseteq \mathcal{M}$ be an $\epsilon \cdot r$ -clique and assume that X can be covered by a ball of radius r centered at a point of \mathcal{M} . Then, $|X| \leq (4/\epsilon)^D$.*

Proof. By recursively applying the definition of doubling dimension, we observe that the ball of radius r which covers X can be covered by $2^{j \cdot D}$ balls of radius $2^{-j} \cdot r$, where j is any non negative integer. Let i be the least integer for which $2^{-i} \cdot r \leq \epsilon/2 \cdot r$ holds. Any of the $2^{i \cdot D}$ balls with radius $2^{-i} \cdot r$ can contain at most one point of X , since X is a $\epsilon \cdot r$ -clique. Thus $|X| \leq 2^{i \cdot D}$. As $i = 1 + \lceil \log_2(1/\epsilon) \rceil$, we finally obtain that $|X| \leq (4/\epsilon)^D$. ◀

► **Theorem 3.3.** *Let C_w be the set returned by the execution of `CoverWithBalls`(P, T, R, ϵ, β). Suppose that the points in P and T belong to a metric space with doubling dimension D . Let c be a real value such that, for any $x \in P$, $c \cdot R \geq d(x, T)$. Then,*

$$|C_w| \leq |T| \cdot (16\beta/\epsilon)^D \cdot (\log_2 c + 2)$$

Proof. Let $T = \{t_1, \dots, t_{|T|}\}$ be the set in input to the algorithm. For any i , $1 \leq i \leq |T|$, let $P_i = \{x \in P : x^T = t_i\}$ and $B_i = \{x \in P_i : d(x, T) \leq R\}$. In addition, for any integer value $j \geq 0$ and for any feasible value of i , we define $D_{i,j} = \{x \in P_i : 2^j \cdot R < d(x, T) \leq 2^{j+1} \cdot R\}$. We observe that for any $j \geq \lceil \log_2 c \rceil$, the sets $D_{i,j}$ are empty, since $d(x, T) \leq c \cdot R$. Together, the sets B_i and $D_{i,j}$ are a partition of P_i .

For any i , let $C_i = C_w \cap B_i$. We now want to show that the set C_i is a $\epsilon/(2\beta) \cdot R$ -clique. Let c_1, c_2 be any two different points in C_i and suppose, without loss of generality, that c_1 was added first to C_w . Since c_2 was not removed from P , this means that $d(c_1, c_2) > \epsilon/(2\beta) \cdot \max\{d(c_2, T), R\} \geq \epsilon/(2\beta)R$, where we used the fact that $d(c_2, T) \leq R$ since c_2 belongs to B_i . Also, the set $C_i \subseteq B_i$ is contained in a ball of radius R centered in t_i , thus we can apply Lemma 3.2 and bound its size, obtaining that $|C_i| \leq (8\beta/\epsilon)^D$.

For any i and j , let $C_{i,j} = C_w \cap D_{i,j}$. We can use a similar strategy to bound the size of those sets. We first show that the sets $C_{i,j}$ are $\frac{\epsilon}{4\beta} \cdot 2^{j+1}R$ -cliques. Let c_1, c_2 be any two different points in $C_{i,j}$ and suppose, without loss of generality, that c_1 was added first to C_w . Since c_2 was not removed from P , this means that $d(c_1, c_2) > \epsilon/(2\beta) \cdot \max\{d(c_2, T), R\} \geq \epsilon/(4\beta)2^{j+1}R$, where we used the fact that $d(c_2, T) > 2^j \cdot R$ since c_2 belongs to $D_{i,j}$. Also, the set $C_{i,j} \subseteq D_{i,j}$ is contained in a ball of radius $2^{j+1}R$ centered in t_i , thus we can apply Lemma 3.2 and obtain that $|C_{i,j}| \leq (16\beta/\epsilon)^D$. Since the sets C_i and $C_{i,j}$ partition C_w , we can bound the size of C_w as the sum of the bounds of the size of those sets. Hence:

$$|C_w| \leq \sum_{i=1}^{|T|} |C_i| + \sum_{i=1}^{|T|} \sum_{j=0}^{\lceil \log_2 c \rceil - 1} |C_{i,j}| \leq |T| \cdot (16\beta/\epsilon)^D \cdot (\log_2 c + 2). \quad \blacktriangleleft$$

3.1 A first approach to coreset construction for k -median

In this subsection we present a 1-round MapReduce algorithm that builds a weighted coreset $C_w \subseteq P$ of a k -median instance $\mathcal{I} = (P, k)$. The algorithm is parametrized by a value $\epsilon \in (0, 1)$, which represents a tradeoff between coreset size and accuracy. The returned coreset has the following property. Let $\mathcal{I}' = (C_w, k)$. If we run an α -approximation algorithm

on \mathcal{I}' , then the returned solution is a $(2\alpha + O(\epsilon))$ -approximate solution of \mathcal{I} . Building on this construction, in the next subsection we will obtain a better coreset which allows us to reduce the final approximation factor to the desired $\alpha + O(\epsilon)$ value. The coreset construction algorithm operates as follows. The set P is partitioned into L equally-sized subsets P_1, \dots, P_L . In parallel, on each k -median instance $\mathcal{I}_\ell = (P_\ell, k)$, with $\ell = 1, \dots, L$, the following operations are performed:

1. Compute a set T_ℓ of $m \geq k$ points such that $\nu_{P_\ell}(T_\ell) \leq \beta \cdot \nu_{P_\ell}(\text{opt}_{\mathcal{I}_\ell})$.
2. $R_\ell \leftarrow \nu_{P_\ell}(T_\ell)/|P_\ell|$.
3. $C_{w,\ell} \leftarrow \text{CoverWithBalls}(P_\ell, T_\ell, R_\ell, \epsilon, \beta)$.

The set $C_w = \cup_{\ell=1}^L C_{w,\ell}$ is the output of the algorithm.

In Step 1, the set T_ℓ can be computed through a sequential (possibly bi-criteria) approximation algorithm for m -median, with a suitable $m \geq k$, to yield a small value of β . If we assume that such an algorithm requires space linear in P_ℓ , the entire coreset construction can be implemented in a single MapReduce round, using $O(|P|/L)$ local memory and $O(|P|)$ aggregate memory. For example, using one of the known linear-space, constant-approximation algorithms (e.g., [2]), we can get $\beta = O(1)$ with $m = k$.

► **Lemma 3.4.** *For $\ell = 1, \dots, L$, $C_{w,\ell}$ is an ϵ -bounded coreset of the k -median instance \mathcal{I}_ℓ .*

Proof. Fix a value of ℓ . Let τ_ℓ be the map between the points in $C_{w,\ell}$ and the points in P_ℓ of Lemma 3.1. The set $C_{w,\ell}$ is weighted according to τ_ℓ . Also, it holds that:

$$\sum_{x \in P_\ell} d(x, \tau_\ell(x)) \leq \frac{\epsilon}{2\beta} \sum_{x \in P_\ell} (R_\ell + d(x, T_\ell)) \leq \frac{\epsilon}{2\beta} (R_\ell \cdot |P_\ell| + \nu_{P_\ell}(T_\ell)) \leq \epsilon \cdot \nu_{P_\ell}(\text{opt}_{\mathcal{I}_\ell}) \quad \blacktriangleleft$$

By combining Lemma 3.4 and Lemma 2.7, the next lemma immediately follows.

► **Lemma 3.5.** *Let $\mathcal{I} = (P, k)$ be a k -median instance. The set C_w returned by the above MapReduce algorithm is a 2ϵ -bounded coreset of \mathcal{I} .*

It is possible to bound the size of C_w as a function of the doubling dimension D . For any $\ell = 1, \dots, L$ and $x \in P_\ell$, it holds that $R_\ell \cdot |P_\ell| = \nu_{P_\ell}(T_\ell) \geq d(x, T_\ell)$, thus we can bound the size of $C_{w,\ell}$ by using Theorem 3.3. Since C_w is the union of those sets, this argument proves the following lemma.

► **Lemma 3.6.** *Let $\mathcal{I} = (P, k)$ be a k -median instance. Suppose that the points in P belong to a metric space with doubling dimension D . Let C_w be the set returned by the above MapReduce algorithm with input \mathcal{I} and $m \geq k$. Then, $|C_w| = O(L \cdot m \cdot (16\beta/\epsilon)^D \log |P|)$*

Let S be an α -approximate solution of $\mathcal{I}' = (C_w, k)$, with constant α . We will now show that $\nu_P(S)/\nu_P(\text{opt}_{\mathcal{I}}) = 2\alpha + O(\epsilon)$. Let τ be the map of from P to C_w (see Lemma 3.1). By triangle inequality, $\nu_P(S) \leq \sum_{x \in P} d(x, \tau(x)) + \nu_{C_w}(S)$. We have that $\sum_{x \in P} d(x, \tau(x)) \leq 2\epsilon \cdot \nu_P(\text{opt}_{\mathcal{I}})$ since, by Lemma 3.5, C_w is a 2ϵ -bounded coreset. By the fact that S is an α -approximate solution of \mathcal{I}' and by Lemma 2.6, we have that $\nu_{C_w}(S) \leq \alpha \cdot \nu_{C_w}(\text{opt}_{\mathcal{I}'}) \leq 2\alpha \cdot \nu_{C_w}(\text{opt}_{\mathcal{I}})$. By Lemma 2.4, C_w is also a 2ϵ -approximate coreset of \mathcal{I} , thus $\nu_{C_w}(\text{opt}_{\mathcal{I}}) \leq (1 + 2\epsilon)\nu_P(\text{opt}_{\mathcal{I}})$. Putting it all together, we have that $\nu_P(S)/\nu_P(\text{opt}_{\mathcal{I}}) \leq 2\alpha(1 + 2\epsilon) + 2\epsilon = 2\alpha + O(\epsilon)$. We observe that the factor 2 is due to the inequality which relates $\text{opt}_{\mathcal{I}}$ and $\text{opt}_{\mathcal{I}'}$, namely $\nu_{C_w}(\text{opt}_{\mathcal{I}'}) \leq 2\nu_{C_w}(\text{opt}_{\mathcal{I}})$. In the next subsection, we will show how to get rid of this factor.

Application to the continuous case

The same algorithm of this subsection can also be used to build a $O(\epsilon)$ -approximate coresets in the continuous scenario where centers are not required to belong to P . It is easy to verify that the construction presented in this subsection also works in the continuous case, with the final approximation factor improving to $(\alpha + O(\epsilon))$. Indeed, we can use the stronger inequality $\nu_{C_w}(\text{opt}_{\mathcal{I}'}) \leq \nu_{C_w}(\text{opt}_{\mathcal{I}})$, as $\text{opt}_{\mathcal{I}}$ is also a solution of \mathcal{I}' , which allows us to avoid the factor 2 in front of α . While the same approximation guarantee has already been achieved in the literature using more space-efficient but randomized coresets constructions [6, 4], as mentioned in the introduction, this result provides evidence of the general applicability of our approach.

3.2 Coresets construction for k -median

In this subsection, we present a 2-round MapReduce algorithm which computes a weighted subset which is both an $O(\epsilon)$ -bounded coresets and an $O(\epsilon)$ -centroid set of an input instance $\mathcal{I} = (P, k)$ of k -median. The algorithm is similar to the one of the previous subsection, but applies `CoverWithBalls` twice in every subset of the partition. This idea is inspired by the strategy presented in [14] for \mathbb{R}^d , where a double exponential grid construction is used to ensure that the returned subset is a centroid set.

First Round. P is partitioned into L equally-sized subsets P_1, \dots, P_L . Then in parallel, on each k -median instance $\mathcal{I}_\ell = (P_\ell, k)$, with $\ell = 1, \dots, L$, the following steps are performed:

1. Compute a set T_ℓ of $m \geq k$ points such that $\nu_{P_\ell}(T_\ell) \leq \beta \cdot \nu_{P_\ell}(\text{opt}_{\mathcal{I}_\ell})$.
2. $R_\ell \leftarrow \nu_{P_\ell}(T_\ell)/|P_\ell|$.
3. $C_{w,\ell} \leftarrow \text{CoverWithBalls}(P_\ell, T_\ell, R_\ell, \epsilon, \beta)$.

Second Round. Let $C_w = \cup_{\ell=1}^L C_{w,\ell}$. The same partition of P of the first round is used. Together with P_ℓ , the ℓ -th reducer receives a copy of C_w , and all values R_i computed in the previous round, for $i = 1, \dots, L$. On each k -median instance $\mathcal{I}_\ell = (P_\ell, k)$, with $\ell = 1, \dots, L$, the following steps are performed:

1. $R \leftarrow \sum_{i=1}^L |P_i| \cdot R_i / |P|$
2. $E_{w,\ell} \leftarrow \text{CoverWithBalls}(P_\ell, C_w, R, \epsilon, \beta)$.

The set $E_w = \cup_{\ell=1}^L E_{w,\ell}$ is the output of the algorithm. The computation of T_ℓ in the first round is accomplished as described in the previous section.

The following lemma characterizes the properties of E_w .

► **Lemma 3.7.** *Let $\mathcal{I} = (P, k)$ be a k -median instance. Then, the set E_w returned by the above MapReduce algorithm is both a 2ϵ -bounded coresets and a 7ϵ -centroid set of \mathcal{I} .*

Proof. The first three steps of the algorithm are in common with the algorithm of Subsection 3.2. By Lemma 3.4, for $\ell = 1, \dots, L$, the sets $C_{w,\ell}$ are ϵ -bounded coresets of \mathcal{I}_ℓ . Let $C_w = \cup_{\ell=1}^L C_{w,\ell}$. By Lemma 2.7, the set C_w is a 2ϵ -bounded coresets of \mathcal{I} , and also, by Lemma 2.4, a 2ϵ -approximate coresets. Let $\tau(x)$ be the map from P to C_w as specified in Definition 2.3. It holds that $\nu_P(C_w) \leq \sum_{x \in P} d(x, \tau(x)) \leq 2\epsilon \cdot \nu_P(\text{opt}_{\mathcal{I}})$. Let ϕ_ℓ be the map of Lemma 3.1 from the points in P_ℓ to the points in $E_{w,\ell}$. By reasoning as in the proof of Lemma 3.4, we obtain that $\sum_{x \in P_\ell} d(x, \phi_\ell(x)) \leq \epsilon / (2\beta) [|P_\ell| \cdot R + \nu_{P_\ell}(C_w)]$. For any $x \in P$, let $\hat{\ell}$ be the index for which $x \in P_{\hat{\ell}}$, we define $\phi(x) = \phi_{\hat{\ell}}(x)$. We have that

$$\sum_{x \in P} d(x, \phi(x)) \leq \frac{\epsilon}{2\beta} \sum_{\ell=1}^L [R \cdot |P_\ell| + \nu_{P_\ell}(C_w)] = \frac{\epsilon}{2\beta} \left(\left(\sum_{\ell=1}^L |P_\ell| \cdot R_\ell \right) + \nu_P(C_w) \right)$$

where in the last equality we applied the definition of R . Since $|P_\ell| \cdot R_\ell = \nu_{P_\ell}(T_\ell) \leq \beta \cdot \nu_{P_\ell}(\text{opt}_{\mathcal{I}}) \leq 2\beta \cdot \nu_P(\text{opt}_{\mathcal{I}})$, where the last inequality follows from Lemma 2.6, we have that $\sum_{\ell=1}^L |P_\ell| \cdot R_\ell \leq 2\beta \cdot \nu_P(\text{opt}_{\mathcal{I}})$. Additionally, $\nu_P(C_w) \leq 2\epsilon \cdot \nu_P(\text{opt}_{\mathcal{I}})$ as argued previously in the proof. Therefore E_w is a 2ϵ -bounded coreset.

We now show that E_w is a 7ϵ -centroid set of \mathcal{I} . Let $X = \{x^{E_w} : x \in \text{opt}_{\mathcal{I}}\}$. We will prove that $\nu_P(X) \leq (1 + 7\epsilon)\nu_P(\text{opt}_{\mathcal{I}})$. By triangle inequality, we obtain that:

$$\nu_P(X) = \sum_{x \in P} d(x, X) \leq \sum_{x \in P} d(x, \tau(x)) + \sum_{x \in P} d(\tau(x), X)$$

The first term of the above sum can be bounded as $\sum_{x \in P} d(x, \tau(x)) \leq 2\epsilon \cdot \nu_P(\text{opt}_{\mathcal{I}})$, since C_w is a 2ϵ -bounded coreset. Also, we notice that the second term of the sum can be rewritten as $\sum_{x \in P} d(\tau(x), X) = \sum_{x \in C_w} w(x)d(x, X)$, due to the relation between τ and w . By triangle inequality, we obtain that:

$$\sum_{x \in C_w} w(x)d(x, X) \leq \sum_{x \in C_w} w(x)d(x, x^{\text{opt}_{\mathcal{I}}}) + \sum_{x \in C_w} w(x)d(x^{\text{opt}_{\mathcal{I}}}, X)$$

Since C_w is a 2ϵ -approximate coreset, we can use the bound $\sum_{x \in C_w} w(x)d(x, x^{\text{opt}_{\mathcal{I}}}) = \nu_{C_w}(\text{opt}_{\mathcal{I}}) \leq (1 + 2\epsilon)\nu_P(\text{opt}_{\mathcal{I}})$. Also, by using the definition of X , we observe that

$$\begin{aligned} \sum_{x \in C_w} w(x)d(x^{\text{opt}_{\mathcal{I}}}, X) &= \sum_{x \in C_w} w(x)d(x^{\text{opt}_{\mathcal{I}}}, E_w) \leq \sum_{x \in C_w} w(x)d(x^{\text{opt}_{\mathcal{I}}}, \phi(x^{\text{opt}_{\mathcal{I}}})) \\ &\leq \frac{\epsilon}{2\beta} \sum_{x \in C_w} w(x) \cdot (R + d(x^{\text{opt}_{\mathcal{I}}}, C_w)) \leq \frac{\epsilon}{2\beta} \left(\left(\sum_{\ell=1}^L |P_\ell| \cdot R_\ell \right) + \nu_{C_w}(\text{opt}_{\mathcal{I}}) \right) \end{aligned}$$

In the last inequality, we used the definition of R , and the simple observation that for any $x \in C_w$, $d(x^{\text{opt}_{\mathcal{I}}}, C_w) \leq d(x, x^{\text{opt}_{\mathcal{I}}}) = d(x, \text{opt}_{\mathcal{I}})$. As argued previously in the proof, we have that $\sum_{\ell} |P_\ell| \cdot R_\ell \leq 2\beta \cdot \nu_P(\text{opt}_{\mathcal{I}})$. Also, $\nu_{C_w}(\text{opt}_{\mathcal{I}}) \leq (1 + 2\epsilon)\nu_P(\text{opt}_{\mathcal{I}})$ as C_w is a 2ϵ -approximate coreset of \mathcal{I} . Since we assume that $\beta \geq 1$, we finally obtain:

$$\sum_{x \in C_w} w(x)d(x^{\text{opt}_{\mathcal{I}}}, X) \leq \frac{\epsilon}{2\beta} (2\beta + 1 + 2\epsilon)\nu_P(\text{opt}_{\mathcal{I}}) \leq 3\epsilon \cdot \nu_P(\text{opt}_{\mathcal{I}})$$

We conclude that $\nu_P(X) \leq (2\epsilon + 1 + 2\epsilon + 3\epsilon)\nu_P(\text{opt}_{\mathcal{I}}) = (1 + 7\epsilon) \cdot \nu_P(\text{opt}_{\mathcal{I}})$ ◀

The next lemma establishes an upper bound on the size of E_w .

► **Lemma 3.8.** *Let $\mathcal{I} = (P, k)$ be a k -median instance. Suppose that the points in P belong to a metric space with doubling dimension D . Let E_w be the set returned by the above MapReduce algorithm with input \mathcal{I} and $m \geq k$. Then $|E_w| = O(L^2 \cdot m \cdot (16\beta/\epsilon)^{2D} \log^2 |P|)$.*

Proof. From the previous subsection, we know that $|C_w| = O(L \cdot m \cdot (16\beta/\epsilon)^D \log |P|)$. Also, by Lemma 3.4, we have that $\nu_{P_\ell}(C_{w,\ell}) \leq \epsilon \cdot \nu_{P_\ell}(\text{opt}_{\mathcal{I}_\ell})$ for any $\ell = 1, \dots, L$. For every $x \in P$ we have that $\epsilon|P| \cdot R = \epsilon \sum_{\ell} |P_\ell| \cdot R_\ell = \epsilon \sum_{\ell} \nu_{P_\ell}(T_\ell) \geq \sum_{\ell} \epsilon \cdot \nu_{P_\ell}(\text{opt}_{\mathcal{I}_\ell}) \geq \sum_{\ell} \nu_{P_\ell}(C_{w,\ell}) \geq \nu_P(C_w) \geq d(x, C_w)$. The lemma follows by applying Theorem 3.3 to bound the sizes of the sets $E_{w,\ell}$. ◀

We are now ready to state the main result of this subsection.

► **Theorem 3.9.** *Let $\mathcal{I} = (P, k)$ be a k -median instance and let E_w be the set returned by the above MapReduce algorithm for a fixed $\epsilon \in (0, 1)$. Let \mathcal{A} be an α -approximation algorithm for the k -median problem, with constant α . If S is the solution returned by \mathcal{A} with input $\mathcal{I}' = (E_w, k)$, then $\nu_P(S)/\nu_P(\text{opt}_{\mathcal{I}}) \leq \alpha + O(\epsilon)$.*

Proof. Let τ be the map from P to E_w of Definition 2.3. By triangle inequality, it results that $\nu_P(S) \leq \sum_{x \in P} d(x, \tau(x)) + \nu_{E_w}(S)$. The set E_w is a 2ϵ -bounded coreset of \mathcal{I} , so we have that $\sum_{x \in P} d(x, \tau(x)) \leq 2\epsilon \cdot \nu_P(\text{opt}_{\mathcal{I}})$. Since \mathcal{A} is an α -approximation algorithm, we have that $\nu_{E_w}(S) \leq \alpha \cdot \nu_{E_w}(\text{opt}_{\mathcal{I}'})$. As E_w is also a 7ϵ -centroid set, there exists a solution $X \subseteq E_w$ such that $\nu_P(X) \leq (1 + 7\epsilon)\nu_P(\text{opt}_{\mathcal{I}'})$. We obtain that $\nu_{E_w}(\text{opt}_{\mathcal{I}'}) \leq \nu_{E_w}(X) \leq (1 + 2\epsilon)(1 + 7\epsilon)\nu_P(\text{opt}_{\mathcal{I}'})$. In the last inequality, we used the fact that E_w is a 2ϵ -approximate coreset of \mathcal{I} due to Lemma 2.4. To wrap it up, $\nu_P(X)/\nu_P(\text{opt}_{\mathcal{I}}) \leq \alpha(1 + 7\epsilon)(1 + 2\epsilon) + 2\epsilon = \alpha + O(\epsilon)$. ◀

3.3 Coreset construction for k -means

In this subsection, we present a 2-round MapReduce algorithm to compute a weighted subset E_w which is both an $O(\epsilon^2)$ -approximate coreset and a $O(\epsilon)$ -centroid set of an instance \mathcal{I} of k -means and then show that an α -approximate solution of $\mathcal{I}' = (E_w, k)$ is an $(\alpha + O(\epsilon))$ -approximate solution of \mathcal{I} . The algorithm is an adaptation of the one devised in the previous subsection for k -median, with suitable tailoring of the parameters involved to account for the presence of squared distances in the objective function of k -means.

First Round. P is partitioned into L equally-sized subsets P_1, \dots, P_L . Then in parallel, on each k -means instance $\mathcal{I}_\ell = (P_\ell, k)$, with $\ell = 1, \dots, L$, the following steps are performed:

1. Compute a set T_ℓ of $m \geq k$ points such that $\mu_{P_\ell}(T_\ell) \leq \beta \cdot \mu_{P_\ell}(\text{opt}_{\mathcal{I}_\ell})$.
2. $R_\ell \leftarrow \sqrt{\mu_{P_\ell}(T_\ell)/|P_\ell|}$.
3. $C_{w,\ell} \leftarrow \text{CoverWithBalls}(P_\ell, T_\ell, R_\ell, \sqrt{2}\epsilon, \sqrt{\beta})$.

Second Round. Let $C_w = \cup_{\ell=1}^L C_{w,\ell}$. The same partition of P of the first round is used. Together with P_ℓ , the ℓ -th reducer receives a copy of C_w , and all values R_i computed in the previous round, for $i = 1, \dots, L$. On each k -means instance $\mathcal{I}_\ell = (P_\ell, k)$, with $\ell = 1, \dots, L$, the following steps are performed:

1. $R \leftarrow \sqrt{\sum_{i=1}^L |P_i| \cdot R_i^2 / |P|}$
2. $E_{w,\ell} \leftarrow \text{CoverWithBalls}(P_\ell, C_w, R, \sqrt{2}\epsilon, \sqrt{\beta})$.

The set $E_w = \cup_{\ell=1}^L E_{w,\ell}$ is the output of the algorithm. The computation of T_ℓ in the first round can be accomplished using the the linear-space constant approximation algorithms of [12, 18].

The analysis follows the lines of the one carried out for the k -median coreset construction. The following lemma establishes the properties of each $C_{w,\ell}$.

► **Lemma 3.10.** *For $\ell = 1, \dots, L$, $C_{w,\ell}$ is a ϵ^2 -bounded coreset of the k -means instance \mathcal{I}_ℓ .*

Proof. Fix a value of ℓ . Let τ_ℓ be the map between the points in $C_{w,\ell}$ and the points in P_ℓ of Lemma 3.1. The set $C_{w,\ell}$ is weighted according to τ_ℓ . Also, it holds that:

$$\begin{aligned} \sum_{x \in P_\ell} d(x, \tau_\ell(x))^2 &\leq \frac{\epsilon^2}{2\beta} \sum_{x \in P_\ell} [R_\ell^2 + d(x, T_\ell)^2] \leq \frac{\epsilon^2}{2\beta} [R_\ell^2 \cdot |P_\ell| + \mu_{P_\ell}(T_\ell)] \\ &\leq \epsilon^2 \cdot \mu_{P_\ell}(\text{opt}_{\mathcal{I}_\ell}). \end{aligned} \quad \blacktriangleleft$$

Next, in the following two lemmas, we characterize the properties and the size of E_w .

► **Lemma 3.11.** *Let $\mathcal{I} = (P, k)$ be a k -means instance and assume that ϵ is a positive value such that $\epsilon + \epsilon^2 \leq 1/8$. Then, the set E_w returned by the above MapReduce algorithm is both a $4\epsilon^2$ -bounded coreset and a 27ϵ -centroid set of \mathcal{I} .*

Proof. Let ϕ_ℓ be the map of Lemma 3.1 from the points in P_ℓ to the points in $E_{w,\ell}$. We have that $\sum_{x \in P_\ell} d(x, \phi_\ell(x))^2 \leq \epsilon^2 / (2\beta) (|P_\ell| \cdot R_\ell^2 + \mu_{P_\ell}(C_w))$. For any $x \in P$, let ℓ be the index for which $x \in P_\ell$, we define $\phi(x) = \phi_\ell(x)$. We have that:

$$\sum_{x \in P} d(x, \phi(x))^2 \leq \frac{\epsilon^2}{2\beta} \sum_{\ell=1}^L [R_\ell^2 |P_\ell| + \mu_{P_\ell}(C_w)] = \frac{\epsilon^2}{2\beta} \left(\left(\sum_{\ell=1}^L |P_\ell| \cdot R_\ell^2 \right) + \mu_P(C_w) \right)$$

Using the fact that $|P_\ell| \cdot R_\ell^2 = \mu_{P_\ell}(T_\ell) \leq \beta \cdot \mu_{P_\ell}(\text{opt}_{\mathcal{I}_\ell}) \leq 4\beta \cdot \mu_{P_\ell}(\text{opt}_{\mathcal{I}})$, where the last inequality is due to Lemma 2.6, we have that $\sum_{\ell} R_\ell^2 |P_\ell| \leq \sum_{\ell} 4\beta \cdot \mu_{P_\ell}(\text{opt}_{\mathcal{I}}) \leq 4\beta \cdot \mu_P(\text{opt}_{\mathcal{I}})$. Also, by Lemma 3.10 and Lemma 2.7, C_w is an $4\epsilon^2$ -bounded coreset of P , thus $\mu_P(C_w) \leq 4\epsilon^2 \cdot \mu_P(\text{opt}_{\mathcal{I}})$. Therefore, E_w is an $4\epsilon^2$ -bounded coreset of \mathcal{I} .

We now show that E_w is a centroid set of \mathcal{I} . Let $X = \{x^{E_w} : x \in \text{opt}_{\mathcal{I}}\}$. By Lemma 2.5, C_w is a γ -approximate coreset of \mathcal{I} , with $\gamma = 4(\epsilon + \epsilon^2) \leq 1/2$. Hence, $\mu_P(X) \leq 1/(1 - \gamma) \cdot \mu_{C_w}(X)$. By Proposition 2.1, we have:

$$\mu_{C_w}(X) = \sum_{x \in C_w} w(x) d(x, X)^2 \leq (1 + \epsilon) \mu_{C_w}(\text{opt}_{\mathcal{I}}) + (1 + 1/\epsilon) \sum_{x \in C_w} w(x) d(x^{\text{opt}_{\mathcal{I}}}, X)^2$$

Since C_w is a γ -approximate coreset, it holds that $\mu_{C_w}(\text{opt}_{\mathcal{I}}) \leq (1 + \gamma) \mu_P(\text{opt}_{\mathcal{I}})$. By reasoning as in the proof of Lemma 3.7, we have that $\sum_{x \in C_w} w(x) d(x^{\text{opt}_{\mathcal{I}}}, X)^2 \leq (5\epsilon^2/2 + \gamma\epsilon^2/2) \mu_P(\text{opt}_{\mathcal{I}})$. Putting it all together, we conclude:

$$\mu_P(X) / \mu_P(\text{opt}_{\mathcal{I}}) \leq (1 + \gamma + 5\epsilon^2/2 + \gamma\epsilon^2/2 + 7\epsilon/2 + 3\gamma\epsilon/2) / (1 - \gamma).$$

Since $\gamma \leq 1/2$, we have that $1/(1 - \gamma) \leq 1 + 2\gamma$. By using the constraint on ϵ and the definition of γ , after some tedious computations, we obtain $\mu_P(X) / \mu_P(\text{opt}_{\mathcal{I}}) \leq 1 + 27\epsilon$. ◀

► **Lemma 3.12.** *Let $\mathcal{I} = (P, k)$ be a k -means instance. Suppose that the points in P belong to a metric space with doubling dimension D . Let E_w be the set returned by the above MapReduce algorithm with input \mathcal{I} and $m \geq k$. Then, $|E_w| = O(L^2 \cdot m \cdot (8\sqrt{2}\beta/\epsilon)^{2D} \log^2 |P|)$*

Proof. For any $\ell = 1, \dots, L$ and $x \in P_\ell$, it holds that $R_\ell \cdot \sqrt{|P_\ell|} = \sqrt{\mu_{P_\ell}(T_\ell)} \geq d(x, T_\ell)$. By using Theorem 3.3, we obtain that $|C_{w,\ell}| = O(m \cdot (8\sqrt{2}\beta/\epsilon)^D \log |P|)$, and we can bound the size of C_w with an union bound. By Lemma 3.10, $C_{w,\ell}$ is a ϵ^2 -bounded coreset of \mathcal{I}_ℓ , hence $\mu_{P_\ell}(C_{w,\ell}) \leq \epsilon^2 \mu_{P_\ell}(\text{opt}_{\mathcal{I}_\ell})$. For any $x \in P$ we have that $\epsilon \sqrt{|P|} \cdot R = \sqrt{\epsilon^2 \sum_{\ell} |P_\ell| R_\ell^2} = \sqrt{\epsilon^2 \sum_{\ell} \mu_{P_\ell}(T_\ell)} \geq \sqrt{\epsilon^2 \sum_{\ell} \mu_{P_\ell}(\text{opt}_{\mathcal{I}_\ell})} \geq \sqrt{\sum_{\ell} \mu_{P_\ell}(C_{w,\ell})} \geq \sqrt{\mu_P(C_w)} \geq d(x, C_w)$. Thus, the lemma follows by applying Theorem 3.3 to bound the sizes of the sets $E_{w,\ell}$. ◀

We are now ready to state the main result of this subsection.

► **Theorem 3.13.** *Let $\mathcal{I} = (P, k)$ be a k -means instance and let E_w be the set returned by the above MapReduce algorithm for a fixed positive ϵ such that $\epsilon + \epsilon^2 \leq 1/8$. Let \mathcal{A} be an α -approximation algorithm for the k -means problem, with constant α . If S is the solution returned by \mathcal{A} with input $\mathcal{I}' = (E_w, k)$, then $\mu_P(S) / \mu_P(\text{opt}_{\mathcal{I}}) \leq \alpha + O(\epsilon)$.*

Proof. By Lemma 3.11 and Lemma 2.5, E_w is a $(4\epsilon^2 + 4\epsilon)$ -approximate coreset of \mathcal{I} . Therefore, $\mu_P(S) \leq (1/(1 - 4\epsilon - 4\epsilon^2)) \cdot \mu_{E_w}(S)$. Since \mathcal{A} is an α -approximation algorithm, $\mu_{E_w}(S) \leq \alpha \cdot \mu_{E_w}(\text{opt}_{\mathcal{I}'})$. Also, E_w is a 27ϵ -centroid set, thus there exists a solution $X \subseteq E_w$ such that $\mu_P(X) \leq (1 + 27\epsilon) \cdot \mu_P(\text{opt}_{\mathcal{I}'})$. We have that $\mu_{E_w}(\text{opt}_{\mathcal{I}'}) \leq \mu_{E_w}(X) \leq (1 + 4\epsilon + 4\epsilon^2) \cdot \mu_P(X) \leq (1 + 4\epsilon + 4\epsilon^2)(1 + 27\epsilon) \cdot \mu_P(\text{opt}_{\mathcal{I}'})$, where the second inequality follows again from the fact that E_w is a $(4\epsilon^2 + 4\epsilon)$ -approximate coreset of \mathcal{I} . Because of the constraints on ϵ , we have that $1/(1 - 4\epsilon - 4\epsilon^2) \leq 1 + 8\epsilon + 8\epsilon^2$. Therefore, it finally results that $\mu_P(S)/\mu_P(\text{opt}_{\mathcal{I}'}) \leq \alpha \cdot (1 + 8\epsilon + 8\epsilon^2)(1 + 4\epsilon + 4\epsilon^2)(1 + 27\epsilon) = \alpha + O(\epsilon)$. \blacktriangleleft

As noted in Subsection 3.1, a simpler version of this algorithm can be employed if we restrict our attention to the continuous case. Indeed, if we limit the algorithm to the first round and output the set $C_w = \cup_{\ell} C_{w,\ell}$, it is easy to show that an α -approximate algorithm executed on the coreset C_w returns a $(\alpha + O(\epsilon))$ -approximate solution.

3.4 MapReduce algorithms for k -median and k -means

Let $\mathcal{I} = (P, k)$ be a k -median (resp., k -means) instance. We can compute an approximate solution of \mathcal{I} in three MapReduce rounds: in the first two rounds, a weighted coreset E_w is computed using the algorithm described in Subsection 3.2 (resp., Subsection 3.3), while in the third round the final solution is computed by running a sequential approximation algorithm for the weighted variant of the problem on E_w . Suppose that in the first of the two rounds of coreset construction we use a linear-space algorithm to compute the sets T_{ℓ} of size $m = O(k)$, and cost at most a factor β times the optimal cost, and that in the third round we run a linear-space α -approximation algorithm on E_w , with constant α . Setting $L = \sqrt[3]{|P|/k}$ we obtain the following theorem as an immediate consequence of Lemmas 3.8 and 3.12, and Theorems 3.9 and 3.13.

► **Theorem 3.14.** *Let $\mathcal{I} = (P, k)$ be an instance of k -median (resp., k -means). Suppose that the points in P belong to a metric space with doubling dimension D . For any $\epsilon \in (0, 1)$ (with $\epsilon + \epsilon^2 \leq 1/8$ for k -means) the 3-round MapReduce algorithm described above computes an $(\alpha + O(\epsilon))$ -approximate solution of \mathcal{I} using local space $O(|P|^{2/3} k^{1/3} (16\beta/\epsilon)^{2D} \log^2 |P|)$ (resp., $O(|P|^{2/3} k^{1/3} (8\sqrt{2}\beta/\epsilon)^{2D} \log^2 |P|)$).*

Note that for a wide range of the relevant parameters, the local space of the MapReduce algorithms is substantially sublinear in the input size, and it is easy to show that the aggregate space is linear in $|P|$. As concrete instantiations of the above result, both the T_{ℓ} 's and the final solution may be obtained through the sequential algorithms in [2] for k -median, and in [12] for k -means. Both algorithms are based on local search and feature approximations $\alpha = 3 + 2/t$ for k -median, and $\alpha = 5 + 4/t$ for k -means, where t is the number of simultaneous swaps allowed. With this choice, the result of the above theorem holds with $\beta = \alpha = O(1)$. Alternatively, for the T_{ℓ} 's we could use k -means++ [5] as a bi-criteria approximation algorithm (e.g, see [25]), which yields a smaller β , at the expense of a slight, yet constant, increase in the size m of the T_{ℓ} 's. For larger D , this might be a better choice as the coreset size (hence the local memory) is linear in m and β^{2D} (resp., β^D). Moreover, bi-criteria approximations are usually faster to compute than actual solutions.

4 Conclusions

We presented distributed coreset constructions that can be used in conjunction with sequential approximation algorithms for k -median and k -means in general metric spaces to obtain the first space-efficient, 3-round MapReduce algorithms for the two problems, which are almost as

accurate as their sequential counterparts. The constructions for the two problems are based on a uniform strategy, and crucially leverage the properties of spaces of bounded doubling dimension, specifically those related to ball coverings of sets of points. One attractive feature of our constructions is their simplicity, which makes them amenable to fast practical implementations.

References

- 1 D. Arthur and S. Vassilvitskii. k-means++: the advantages of careful seeding. In *Proc. 18th ACM-SIAM SODA*, pages 1027–1035, 2007.
- 2 V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. Local Search Heuristics for k-Median and Facility Location Problems. *SIAM J. Comput.*, 33(3):544–562, 2004.
- 3 P. Awasthi and M.F. Balcan. Center based clustering: A foundational perspective. In *Handbook of cluster analysis*. CRC Press, 2015.
- 4 O. Bachem, M. Lucic, and A. Krause. Scalable k -Means Clustering via Lightweight Coresets. In *Proc. 24th ACM KDD*, pages 1119–1127, 2018.
- 5 B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii. Scalable K-Means++. *PVLDB*, 5(7):622–633, 2012.
- 6 M.F. Balcan, S. Ehrlich, and Y. Liang. Distributed k-means and k-median clustering on general communication topologies. In *Proc. 27th NIPS*, pages 1995–2003, 2013.
- 7 M. Ceccarello, A. Pietracaprina, and G. Pucci. Solving k-center Clustering (with Outliers) in MapReduce and Streaming, almost as Accurately as Sequentially. *PVLDB*, 12(7), 2019.
- 8 E. Cohen, S. Chechik, and H. Kaplan. Clustering Small Samples With Quality Guarantees: Adaptivity With One2all PPS. In *Proc. 32nd AAAI*, pages 2884–2891, 2018.
- 9 J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(1):107–113, 2008.
- 10 A. Ene, S. Im, and B. Moseley. Fast Clustering Using MapReduce. In *Proc. 17th ACM KDD*, pages 681–689, 2011.
- 11 D. Feldman and M. Langberg. A Unified Framework for Approximating and Clustering Data. In *Proc. 43rd ACM STOC*, pages 569–578, 2011.
- 12 A. Gupta and K. Tangwongsan. Simpler Analyses of Local Search Algorithms for Facility Location. *CoRR*, abs/0809.2554, 2008. [arXiv:0809.2554](https://arxiv.org/abs/0809.2554).
- 13 S. Har-Peled and A. Kushal. Smaller Coresets for K-median and K-means Clustering. In *Proc. 21st SCG*, pages 126–134, 2005.
- 14 S. Har-Peled and S. Mazumdar. On Coresets for K-means and K-median Clustering. In *Proc. 36th ACM STOC*, pages 291–300, 2004.
- 15 C. Hennig, M. Meila, F. Murtagh, and R. Rocci. *Handbook of cluster analysis*. CRC Press, 2015.
- 16 L. Huang, S. Jiang, J. Li, and X. Wu. Epsilon-Coresets for Clustering (with Outliers) in Doubling Metrics. In *Proc. 59th IEEE FOCS*, pages 814–825, 2018.
- 17 P. Indyk, S. Mahabadi, M. Mahdian, and V.S. Mirrokni. Composable Core-sets for Diversity and Coverage Maximization. In *Proc. 33rd ACM PODS*, pages 100–108, 2014.
- 18 T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu. A Local Search Approximation Algorithm for K-means Clustering. In *Proc. 18th SCG*, pages 10–18, 2002.
- 19 L. Kaufmann and P. Rousseeuw. Clustering by Means of Medoids. *Data Analysis based on the L1-Norm and Related Methods*, pages 405–416, 1987.
- 20 J. Leskovec, A. Rajaraman, and J.D. Ullman. *Mining of Massive Datasets, 2nd Ed.* Cambridge University Press, 2014.
- 21 J. M. Phillips. Coresets and Sketches. *Handbook of Discrete and Computational Geometry, 3rd Ed.*, 2016.

34:16 Accurate MapReduce Algorithms for k -Median/ k -Means in General Metric Spaces

- 22 A. Pietracaprina, G. Pucci, M. Riondato, F. Silvestri, and E. Upfal. Space-Round Tradeoffs for MapReduce Computations. In *Proc. 26th ACM ICS*, pages 235–244, 2012.
- 23 C. Sohler and D. P. Woodruff. Strong Coresets for k -Median and Subspace Approximation: Goodbye Dimension. In *Proc. 59th IEEE FOCS*, pages 802–813, 2018.
- 24 H. Song, J.G. Lee, and W.S. Han. PAMAE: parallel k -medoids clustering with high accuracy and efficiency. In *Proc. 23rd ACM KDD*, pages 1087–1096, 2017.
- 25 D. Wei. A Constant-Factor Bi-Criteria Approximation Guarantee for k -means++. In *Proc. 30th NIPS*, pages 604–612, 2016.

On Optimal Balance in B-Trees: What Does It Cost to Stay in Perfect Shape?

Rolf Fagerberg

University of Southern Denmark, Odense, Denmark
rolf@imada.sdu.dk

David Hammer

Goethe University Frankfurt, Germany
University of Southern Denmark, Odense, Denmark
hammer@imada.sdu.dk

Ulrich Meyer

Goethe University Frankfurt, Germany
umeyer@ae.cs.uni-frankfurt.de

Abstract

Any B-tree has height at least $\lceil \log_B(n) \rceil$. Static B-trees achieving this height are easy to build. In the dynamic case, however, standard B-tree rebalancing algorithms only maintain a height within a constant factor of this optimum. We investigate exactly how close to $\lceil \log_B(n) \rceil$ the height of dynamic B-trees can be maintained as a function of the rebalancing cost. In this paper, we prove a lower bound on the cost of maintaining optimal height $\lceil \log_B(n) \rceil$, which shows that this cost must increase from $\Omega(1/B)$ to $\Omega(n/B)$ rebalancing per update as n grows from one power of B to the next. We also provide an almost matching upper bound, demonstrating this lower bound to be essentially tight. We then give a variant upper bound which can maintain near-optimal height at low cost. As two special cases, we can maintain optimal height for all but a vanishing fraction of values of n using $\Theta(\log_B(n))$ amortized rebalancing cost per update and we can maintain a height of optimal plus one using $O(1/B)$ amortized rebalancing cost per update. More generally, for any rebalancing budget, we can maintain (as n grows from one power of B to the next) optimal height essentially up to the point where the lower bound requires the budget to be exceeded, after which optimal height plus one is maintained. Finally, we prove that this balancing scheme gives B-trees with very good storage utilization.

2012 ACM Subject Classification Theory of computation \rightarrow Data structures design and analysis

Keywords and phrases B-trees, Data structures, Lower bounds, Complexity

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.35

Funding *Rolf Fagerberg*: Supported by the Independent Research Fund Denmark, Natural Sciences, grant DFF-7014-00041.

David Hammer: Supported by the Deutsche Forschungsgemeinschaft (DFG) under grants ME 2088/3-2 and ME 2088/4-2.

Ulrich Meyer: Supported by the Deutsche Forschungsgemeinschaft (DFG) under grants ME 2088/3-2 and ME 2088/4-2.

1 Introduction

1.1 Motivation

B-trees are search trees particularly suited for data organization in external memory. They are widely employed in database systems and file systems [12] and since their introduction in 1972 by Bayer and McCreight [7], they have been the subject of much theoretical and practical study.



© Rolf Fagerberg, David Hammer, and Ulrich Meyer;
licensed under Creative Commons License CC-BY

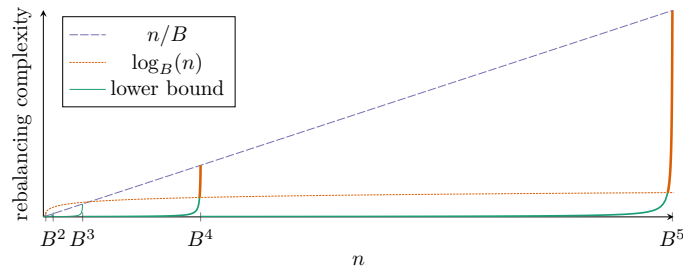
30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 35; pp. 35:1–35:16

Leibniz International Proceedings in Informatics



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



■ **Figure 1** Plot illustrating how the lower bound for rebalancing cost increases as n approaches different powers of B . Intervals where the cost exceeds $\log_B(n)$ are highlighted in orange.

A standard B-tree of order B is a search tree where all leaves are on the same level and every internal node has between $\lceil B/2 \rceil$ and B children except for the root which may have any number of children between 2 and B . One common generalization is (a, b) -trees [16] where internal nodes have between a and b children for $2 \leq a \leq \lceil b/2 \rceil$. The standard operations for rebalancing B-trees are *split* of an overfull node into two nodes (increasing the degree of the parent by one), *merge* which is the reverse of split, and *share* which is a redistribution of keys among neighboring nodes.

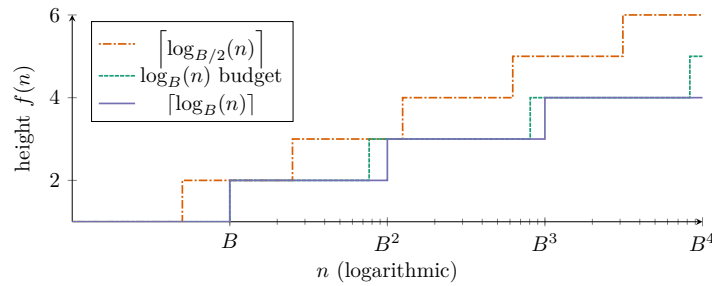
In search trees, the cost of an operation is measured as the number of nodes accessed. For a search operation, this cost is determined by the height of the tree. For any search tree with maximal fanout B , $\lceil \log_B(n) \rceil$ is a lower bound on its height. In the static case, this height is easily achieved by a bottom-up linear (that is, $O(n/B)$ assuming presorted keys) cost construction method. In the dynamic case, standard B-trees maintain an upper bound of $\lceil 1 + \log_{\lceil B/2 \rceil}(n/2) \rceil$ on their height using $O(\log_B(n))$ rebalancing cost per insertion and deletion. Using (a, b) -trees, the amortized rebalancing cost per update can be reduced to $O(1)$ for $a = \lfloor b/2 \rfloor$ and to $O(1/b)$ for $a = b/4$, at the price of a moderate increase in the height bound [16, 22].

The upper bound of $\lceil 1 + \log_{\lceil B/2 \rceil}(n/2) \rceil$ on the height is a constant factor of approximately $\log B / (\log B - 1)$ away from the lower bound. The optimal bound $\lceil \log_B(n) \rceil$ can of course be achieved by spending linear cost on a rebuilding after each update, but this cost is prohibitive in most situations and begs the question: can optimal height be maintained more cheaply?

Intuitively, maintaining optimal height should become harder as n approaches the next power of B , since the amount of available space in the tree decreases, giving less flexibility during rebalancing. Or put differently, the number of different trees of optimal height shrinks as n increases – when reaching a power of B , there is only one such tree. Overall, we can expect that there must be a trade-off between how full the tree is and how costly rebalancing to optimal height will be. The goal we pursue in this paper is to find this trade-off. More generally, we would like to know which height bounds can be maintained at which costs – a correlation which may be called the intrinsic rebalancing cost of B-trees.

1.2 Our contributions

For any n , let N denote the next power of B (i.e., $N = B^{\lceil \log_B n \rceil}$) and define ϵ by $n = N(1 - \epsilon)$. Our first main result is a lower bound showing that for any B-tree of optimal height, there exists an insertion forcing $\Omega(1/(\epsilon B))$ nodes to be rebalanced before the tree can again have optimal height. This expression describes how the rebalancing cost must change from $\Omega(1/B)$ to $\Omega(n/B)$ as n approaches the next power of B . See Figure 1 for a visualization of this bound.



■ **Figure 2** Plot showing different B-tree heights as a function of n . The functions represent the height of standard B-trees, the height achieved by our new scheme using a rebalancing budget of $O(\log_B(n))$, and the optimal height bound. For this particular plot, $B = 10$.

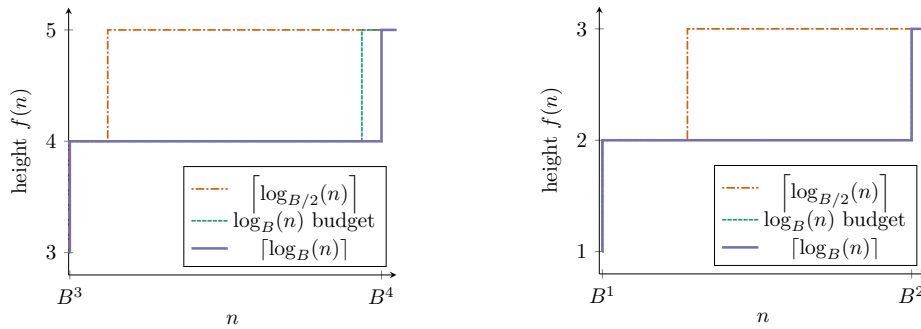
Our second main result is an almost matching upper bound, which maintains optimal height using amortized $O(\log^2 B/(\epsilon B))$ rebalancing per update, thereby proving the lower bound essentially tight.

This lower bound (and hence the upper bound) approaches linear cost as n approaches the next power of two. As our third main result, we give a variant rebalancing scheme that allows almost optimal height at much lower amortized cost. More precisely, for any rebalancing budget $f(n)$, we can maintain optimal height $\lceil \log_B(n) \rceil$ as n approaches the next power of B essentially up to the point where the lower bound result requires the budget to be exceeded, after which height $\lceil \log_B(n) \rceil + 1$ is maintained. To our knowledge, this is the first rebalancing scheme for B-trees with a height bound whose difference compared to optimal is an additive constant rather than a multiplicative constant.

One natural choice of budget is $f(n) = \log_B(n)$ as this matches the search cost. Figure 2 illustrates the height bound of this new scheme, the height bound of standard B-trees, and the optimal height bound. As is hinted by the figure, the fraction of values of n for which this scheme does not maintain optimal height $\lceil \log_B(n) \rceil$ is actually vanishing for growing n . Since in real life uses of B-trees in external memory the values of B are large and realistic values of n are fairly bounded in terms of possible powers of B , one may also want to look at the concrete improvements in height bounds for some practically occurring values of B and n . In database systems and file systems there are two main regimes, often termed OLTP and OLAP in the database setting. In the former, $B = 256$ is a typical value, in the latter, $B = 10^6$ is a typical value. For $B = 256$, the expression $B^3 < M \ll n \lesssim B^4$ (where M denotes the number of keys that can be held in internal memory) describes many real situations. Figure 3a repeats the plot of Figure 2 for the interval $B^3 < n < B^4$, but with $B = 256$ and the horizontal axis linear (not logarithmic). Standard B-trees and our scheme achieve the optimal height bound 4 for some part of the interval, but that part differs significantly in size: 12.16% versus 93.71%. For $B = 10^6$, the expression $B^1 < M \ll n \lesssim B^2$ describes many real situations. As illustrated by Figure 3b, the part of the interval $B^1 < n < B^2$ where optimal height is guaranteed is here 27.49% and 99.98% for the two schemes.

Another natural choice of budget is $f(n) = O(1/B)$ as this is the best possible amortized rebalancing cost (we can always make one node overflow at least every B insertions). With this budget, our scheme maintains height at most $\lceil \log_B(n) \rceil + 1$ for all values of n .

Finally, we prove that our near-optimal balancing scheme gives B-trees with very good storage utilization. Storage utilization is the fraction of the total space in the nodes allocated which is occupied by tree pointers, keys, and elements, i.e., the average use of the space in a node. High storage utilization is a desirable quality since it allows more of the tree to be cached in main memory and it has been the topic of much previous literature on B-trees, in particular in the database setting.



(a) $B = 256$. Standard B-trees are optimal in 12.16% of the interval, our scheme is in 93.71% of the interval.

(b) $B = 10^6$. Standard B-trees are optimal in 27.49% of the interval, our scheme is in 99.98% of the interval. Note: two of the plots are essentially coincident.

■ **Figure 3** Plots showing different B-tree heights as a function of n similar to Figure 2 but for two specific, realistic settings. The horizontal axes are non-logarithmic to more faithfully show proportions.

1.3 Previous work

Not much work has been done with an explicit focus on the height of B-trees. The concept of storage utilization is rather closely related, since optimizing the height requires increasing the average fanout of nodes, which again increases storage utilization. A number of previous results present different trade-offs between update complexity and storage utilization, and we now survey these. Standard B-trees have a worst-case storage utilization of approximately $1/2$ and a logarithmic rebalancing cost. It is well known that lowering the worst-case storage utilization of B-trees slightly by using (a, b) -trees with $a < \lceil b/2 \rceil$ allows for amortized constant rebalancing costs [22, 16]. The average storage utilization may be somewhat better than the worst-case. In [26], an analysis of 2–3 trees with random insertions suggests that such trees tend towards an expected storage utilization of $\ln 2 \approx 0.69$.

To improve the storage utilization in dynamic B-trees, Bayer and McCreight [7] proposed an *overflow* technique: full nodes share their load with their siblings (when possible) instead of splitting (Knuth [18] refers to this variant as B*-trees). This improves the storage utilization in the worst case from $1/2$ to $2/3$, and the height bound from $\lceil 1 + \log_{\lceil B/2 \rceil}(n/2) \rceil$ to $\lceil 1 + \log_{\lceil 2B/3 \rceil}(n/2) \rceil$, at the price of increasing the update cost by a constant factor. As suggested in [7], this approach can be generalized to redistribution in bigger groups of nodes before splitting, thus further improving the storage utilization and the height bound at the cost of even higher rebalancing costs. However, no matter the group size, the height bound will be some constant factor away from optimal. The average storage utilization in a randomized setting using this approach is analyzed in [19], indicating that storage utilization converges to $m \ln((g+1)/g)$ for random trees under insertions where g is the overflow group size.

Similarly, an overflow scheme is tested in [25] which attaches overflow nodes to groups of nodes to delay splitting. They compute the average storage utilization to be $2g/(2g+3)$ when each overflow node is shared by a group of up to g leaf nodes. Both theoretical analysis and practical simulations show improved storage utilization at a cost of increased complexity during updates.

Rosenberg and Snyder [24] introduce so-called *compact B-trees* which are node-oriented trees shown to be close to optimal with respect to access costs while requiring minimal space. However, it is essentially a static structure as compaction incurs linear cost and

no faster compactness-preserving update algorithm is known. The empirical analysis in a dynamic setting presented in [6] indeed shows that insertion into compact B-trees is very costly and that, without compaction, storage utilization rapidly degrades when the number of updates grows.

Recently, Brown [10, 11] has developed a practically motivated variation called *B-slack trees* which has amortized logarithmic update complexity while achieving high storage utilization. *Slack* for a node refers to the difference between its maximum degree and actual degree (number of children “missing”). Special to B-slack trees is that the children of any internal node have a combined slack of at most $B - 1$ where $B > 4$ is the maximum degree of nodes. Maintaining this property is shown to ensure that a tree with n keys occupies at most $\frac{2B}{B-3.4}n$ words (which approaches $2n$ – the optimal value given the model used – as B increases). The rebalancing cost is amortized logarithmic per update.

It is worth mentioning key compression techniques like those found in prefix B-trees [8] as an approach to improve fanout and thus potentially lower tree height. This and similar key compression techniques represent an orthogonal line of research which we do not pursue.

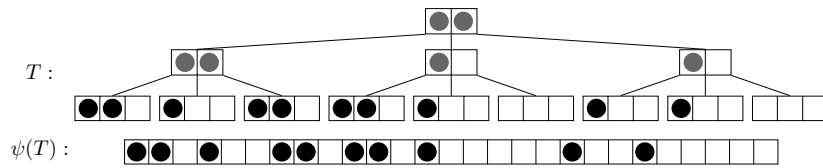
In contrast to B-trees, there for binary trees is a much stronger tradition for focusing on the height. In particular, there is a body of work [23, 3, 1, 2, 4, 5, 21, 20, 14, 15] focusing on rebalancing schemes with height bounds close to the optimal value $\lceil \log_2(n) \rceil$. The end result of this line of investigation is the matching upper and lower bounds in [14, 15]. Our results in this paper can be seen as a generalization of those methods and results to the case of B-trees – setting $B = 2$ in our bounds gives those of [14, 15]. One core new ingredient is the methods of Section 4.4 which are necessary for the upper bound to almost match the loss of a factor of B in the lower bound in Section 3 compared to the bound in [15].

2 Model

We describe our results for leaf-oriented B-trees, as these are standard in the literature. In leaf-oriented B-trees, internal nodes contain a total of $2B - 1$ fields: B pointers to subtrees and $B - 1$ search keys to guide the search (consistent with B-trees of order B as defined by Knuth [18]). Both keys and pointers can be nil. Since keys separate subtrees, the number of non-nil pointers is equal to the number of non-nil keys plus one. Leaf nodes contain the actual elements, including their search keys. For simplicity, we assume that leaf nodes contain up to B elements.¹ All leaf nodes appear at the same level.

A B-tree of height h can contain up to $N = B^h$ elements. We let T denote a B-tree with n elements and optimal height $h = \lceil \log_B n \rceil$ and we define ϵ by $n = N(1 - \epsilon) = B^h(1 - \epsilon)$. Since the height is optimal, we have $B^{h-1} < n \leq B^h$ and hence $0 \leq \epsilon < (B - 1)/B$. A node is said to be modified by an update operation if any of its fields are changed. Clearly the number of modified nodes is a lower bound on the rebalancing cost of an update operation. Unlike in standard B-trees, we do not impose a lower bound on the number of keys. This only makes our lower bound stronger and it thus applies to standard B-trees. Our upper bounds can easily be adapted to conform to a lower bound on node contents.

¹ In practice, the size of elements relative to the size of keys may vary between data sets (and leaves may also store pointers to elements instead of elements themselves), but it is straight-forward to adapt our statements accordingly.



■ **Figure 4** Illustration of the mapping of a tree T (with $B = 3$) into $\psi(T)$. Grey circles represent keys in internal nodes, black circles represent elements (in leaves).

3 Lower bound

In this section, we prove the following main theorem.

► **Theorem 1.** *For any B-tree T of optimal height $h = \lceil \log_B n \rceil$, there exists an insertion into T such that rebalancing T to optimal height after the insertion will require modifying $\Omega(1/(\epsilon B))$ nodes, where ϵ is given by $n = B^h(1 - \epsilon)$.*

The proof is based on creating a mapping from B-trees into arrays (by suitably generalizing a mapping for binary trees in [14]). The mapping allows us to exploit the existence in any array of a “uniformly dense” position, which will point to an update position in T for which we can prove the lower bound stated in Theorem 1.

3.1 Mapping into array

We create the mapping from elements of T to entries of an array $\psi(T)$ of length B^h as follows. For a full tree ($n = B^h$), this mapping is given by an in-order traversal which maps elements met during the traversal to increasing array entries. For a non-full tree, we embed T into a full tree of the same height before applying the mapping. We use the following specific embedding: keys fill up nodes from the left such that missing keys (and for internal nodes the corresponding missing subtrees) for non-full nodes will be on the right. This embedding and the resulting mapping is illustrated in Figure 4.

The mapping has the property that any subtree of the full tree is mapped to a contiguous interval in $\psi(T)$. In particular, for the full tree, a subtree with root at height h' spans an interval of size $B^{h'}$ in $\psi(T)$. This implies that if such a subtree of T is moved during a rebalancing, (say, if siblings of the parent of the subtree are added or removed in T), the positions in $\psi(T)$ of its nodes will be shifted by a multiple of $B^{h'}$.

We denote intervals of array indices (that is, intervals of integers) as $[a; b]$ and the length $b - a + 1$ of such intervals as $l([a; b])$. The following lemma regarding density of subsets of intervals is from Dietz et al. [13].

► **Lemma 2.** *For any two indices a and b and any $S \subseteq [a; b]$, there is an index $i \in [a; b]$ such that for any indices s and t where $a \leq s \leq i \leq t \leq b$, the following holds:*

$$\frac{|S \cap [s; t]|}{l([s; t])} \leq 2 \frac{|S|}{l([a; b])}.$$

Consider the subset $S = \{j \in [a; b] \mid \psi(T)[j] \text{ is empty}\}$ where a and b are the endpoints of the array $\psi(T)$. Applying Lemma 2 with this S , a , and b shows the existence of an index i where any interval around i in $\psi(T)$ will have a density of “holes” (empty array entries) of at most two times the global density of holes in $\psi(T)$, i.e. of at most 2ϵ . For this i , setting $s = t = i$ and $s = i - 1, t = i + 1$ in Lemma 2 shows that $\psi(T)[i]$ and at least one of its direct neighbors are non-empty if $\epsilon < 1/3$. We insert a new element in the tree with a key x lying between $\psi(T)[i]$ and this neighbor. The rest of the proof assumes w.l.o.g. that $\psi(T)[i] < x < \psi(T)[i + 1]$.

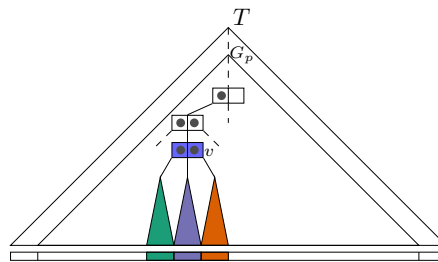
3.2 Counting node changes

Consider any rebalancing operations ensuing from the insertion of x into the original tree T and let T' be the tree after rebalancing. Let all nodes in T modified by the rebalancing be colored blue and let unmodified nodes be colored white. We apply a recursive splitting procedure on all blue nodes layer for layer in a top-down manner. This procedure will maintain a set of *parts*. Each part is a tuple which contains a connected subgraph of T and a contiguous subset of the elements in $\psi(T)$ which we call a segment. As the subgraph in a part is connected, it has a unique highest node which will be called the root of the part.

Initially, the set of parts contains a single tuple consisting of T and a segment containing all elements of $\psi(T)$. Each splitting step on a blue node splits a part into new parts containing subsets of the original part.

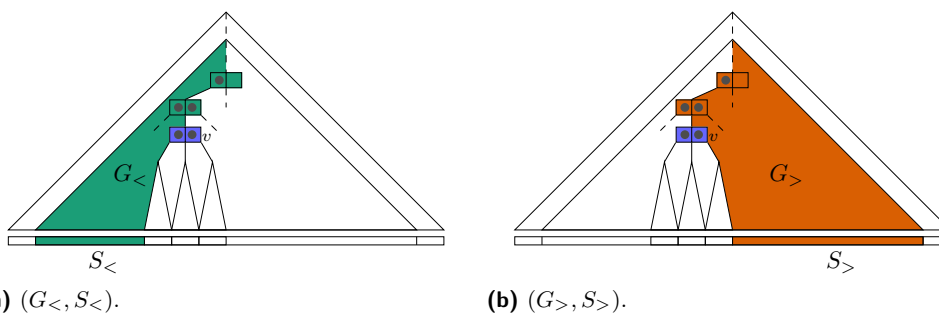
We now describe a splitting step on a blue node v . Let T_v be the subtree of v and let S_v be the elements corresponding to the elements contained in the leaves of T_v mapped into $\psi(T)$. Let (G_p, S_p) be the existing part (where p is the root of G_p) containing v and its subtree. The existing part (G_p, S_p) is replaced by the following new parts.

For each subtree under v , add a new part consisting of the subtree and all elements in the subtree in $\psi(T)$. Refer to Figure 5 for an illustration of these subtree parts. When splitting



■ **Figure 5** Illustrating of a general case of a splitting step on the node v . The subtree parts created in this step (both subgraphs and segments) have been highlighted.

blue leaf nodes, we create a part for each element stored in the node. Such a part will consist of an empty subgraph and a singleton segment containing the element. Generally, G_p may be a proper superset of T_v (see Figure 5). In this case, we create up to two boundary parts for $S_p \setminus S_v$. The segment $S_<$ for the left boundary part $(G_<, S_<)$ consists of all elements in S_p left of S_v . The subgraph $G_<$ consists of all nodes on the path from v to p and all nodes in G_p to the left of this path. Creating the right boundary part is done similarly. See Figure 6 for examples of these boundary parts. For later use, we note that this splitting procedure results in $O(B)$ segments per blue node.



■ **Figure 6** Illustration of the boundary parts of Figure 5.

35:8 On Optimal Balance in B-Trees

The following simple invariants for the splitting procedure are easy to verify: 1) When processing layer k , no part will have a subgraph containing a blue node on any layer $k' > k$. 2) For each part (G, S) where $|S| > 1$, elements of S are in leaves of G . 3) Until the leaves are reached, the process does not create any parts with empty subgraphs.

After the splitting procedure, there will by 1) be no blue nodes within connected subgraphs of the remaining parts. This can be used to show that such subgraphs are not moved up or down.

► **Lemma 3.** *After splitting is done, for parts (G, S) with $|S| > 1$, G will appear in T and T' with the same height.*

Proof. By 3), G will be a non-empty subgraph consisting of white nodes. Hence, this subgraph must appear again in T' . Due to 2), G contains some leaves, hence they appear on the same height in both T and T' (all leaves appear on the same level, so G cannot have moved vertically between T and T'). ◀

We now focus on the segments of parts as the construction will impose restrictions on their ability to move around from $\psi(T)$ to $\psi(T')$. The previous lemma implies that segments can only be *shifted* when going from $\psi(T)$ to $\psi(T')$ (meaning that elements appearing in a segment will have the same relative positions to each other in both arrays). How far a segment is shifted can be bounded from below by its size.

► **Lemma 4.** *A segment containing s elements which has different position in $\psi(T)$ compared to $\psi(T')$ has moved by a distance of $\Omega(s)$.*

Proof. For segments of at most one element, the statement is obvious. Consider a part (G, S) with $s = |S| > 1$ and let v be the topmost node of G . By Lemma 3, G will be found in T' , too, with v having the same height in both trees. As discussed in Section 3.1, nodes on a given height can only appear in specific positions. When v moves, it translates the elements of S by (at least) a multiple of the size (in number of elements in leaves) of a full subtree under v . By (2), this size is at least s . ◀

Proof of Theorem 1. Suppose that the newly inserted element x lies between two segments, S_{l-1} and S_l . As no holes were available, at least one of S_{l-1} and S_l must have moved in $\psi(T')$ compared to $\psi(T)$. Suppose w.l.o.g. that S_l has moved to the right (potentially along with some segments to the right of S_l). Let j be the index of the right-most element in the last consecutive segment S_r right of i which has moved to the right and let $\ell = l([i; j])$. Due to the choice of i , at most $2\epsilon\ell$ of the entries in $[i; j]$ are empty, so there are at least $(1 - 2\epsilon)\ell$ elements in this interval of $\psi(T)$. All of these elements are in the segments S_l, S_{l+1}, \dots, S_r by choice of S_r . Segments can only be translated if there is room in the array (available holes). Since E_{r+1} did not move to the right, this implies that none of the segments S_l, S_{l+1}, \dots, S_r can have moved more than $2\epsilon\ell$, so by Lemma 4, they can then only contain $O(2\epsilon\ell)$ elements each. This means that the number of segments S_l, S_{l+1}, \dots, S_r must be $\Omega\left(\frac{(1-2\epsilon)\ell}{2\epsilon\ell}\right) = \Omega\left(\frac{1}{\epsilon}\right)$. As each blue node accounts for $O(B)$ segments, this implies the existence of $\Omega(1/(\epsilon B))$ blue nodes. ◀

4 Upper bound

In this section, we present a rebalancing scheme for performing insertions into a B-tree containing $B^h(1 - \epsilon)$ elements while maintaining optimal height h . The basic scheme is building on ideas from a rebalancing scheme for binary search tree in [15]. It may also be viewed as a (highly non-trivial) extension of the overflow technique analyzed in [19]. Adding ideas from density keeping algorithms [17], we arrive at the final scheme.

For convenience of notation in the proof, we introduce the parameter $k = 1/\epsilon$. Also, ϵ from now on denotes a value fixed over a sequence of updates (it will be used in such a way that it is never more than a constant factor from its previous meaning, defined by $n = B^h(1 - \epsilon)$).

► **Theorem 5.** *For any ϵ with $0 < \epsilon < (B - 1)/B$ there exists a rebalancing scheme for maintaining optimal height h in a B-tree while its size ranges between $(1 - \epsilon)B^h$ and $(1 - \epsilon/2)B^h$ which has an amortized rebalancing cost per update of*

$$O\left(\frac{k \log^2(\min\{k, B\})}{B}\right),$$

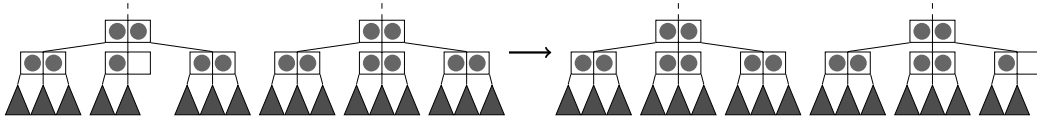
where $k = 1/\epsilon$.

In Section 4.1, we describe the initial setup of the scheme, and in Section 4.2 we describe its rebalancing operations. In Section 4.3, we show how these rebalancing operations can maintain the structure at an amortized rebalancing cost of $O(k)$ node updates per update. In Section 4.4, we combine the scheme with density keeping methods in order to lower the amortized rebalancing cost to $O(k \log^2(\min\{k, B\})/B)$. We focus on insertions, since rebalancing after deletions can be handled by simply running the operations in reverse. We assume $n = (1 - \epsilon)B^h$ initially.

4.1 Layout

The main mechanism employed in the structure is the distribution and redistribution of *holes* in the tree. A hole in a node is simply a missing entry – for leaves, this means that an element field is NIL, for internal nodes this means that a search key field and a corresponding pointer field are NIL. We define the *weight* of a hole in a node of height h' to be $B^{h'}$, i.e., the capacity of a subtree of height h' . This is the number of elements missing from the full tree due to this hole.

Since we initially have $n = (1 - \epsilon)B^h$, we must initialize the tree such that the sum of weights of all holes in the tree is ϵB^h . We call this sum our weight budget. On each level of the tree, we divide the nodes into horizontal *groups* of contiguous nodes. It will be an invariant of the rebalancing scheme that each group contains between 0 and B holes. Initially, this value is B . The bigger the groups, the more sparse the holes will be. On the leaf level we set the group size (the number of nodes in a group) to $\Theta(k)$. On the levels above, we double the group size for each new level. If we conceptually consider this to happen in the full tree of degree B , the B holes per group will on the leaf level correspond to a constant fraction of the weight budget. By tuning the exact group size on the leaf level, we make this fraction be at most $1/8$. While the weight of a hole increases by a factor B per level, the width of the layers in the full tree decreases by the same factor. Thus, doubling the group size means that the total weight of holes in a level decreases by a factor of two for each level. Hence, the level sums in the full tree form a geometrically decreasing series with total sum at most twice the weight of the leaf level, i.e., at most $1/4$ of the weight budget. Some level



■ **Figure 7** Illustration of horizontally sliding a hole within a (sub)tree.

h_{\max} will be the last where the group size does not exceed the total number of nodes on that level. On this level, we place $3/4$ of the total weight budget (arbitrarily positioned on the level). We will refer to this as the *reservoir*.

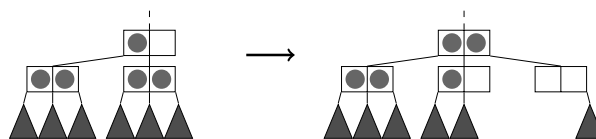
The actual initial layout is built top-down: Above h_{\max} no weight is placed and all nodes have degree B . At level h_{\max} , $3/4$ of the total weight budget is placed, which compared to the full tree removes nodes (by removing entire subtrees) on the levels below. The weight on the lower levels (which removes further nodes of the tree during the top-down process) is given by the group sizes defined above and the rule that each group has B holes. Due to the top-down removal of nodes in the tree, the resulting tree will be thinner than the full tree, hence the level sums in the actual tree produced will be smaller than the $1/4$ of the total budget. Hence, after this top-down procedure, there is some budget of left. This is distributed evenly as holes on the leaf level (these holes are for simplicity of argumentation in proofs considered inactive, i.e., they will not take part in the rebalancing process described below and will not be considered in the invariant that groups have at most B holes). Once the shape of the tree has been produced, it is filled with elements and search keys in a bottom-up fashion. Assuming the elements given in sorted order (for instance during a global rebuild), the above process can be done at linear cost $O(n/B)$.

The rebalancing scheme we describe below works until there are no more holes in the reservoir. By the invariant on the number of holes per group in layers below the reservoir, the total weight outside of the reservoir will never exceed the fraction $1/4$, so emptying the reservoir requires a number of insertions proportional to the initial weight in the reservoir. Hence, the scheme will last at least until size $(1 - \epsilon/2)B^h$, as required for Theorem 5.

4.2 Rebalancing operations

Two basic operations will be used in the rebalancing scheme to move holes around within the tree: *horizontal sliding* and *vertical redistribution*. As the names imply, horizontal sliding will move a hole from one location to another (within a group) on the same level while vertical redistribution moves a hole from one level to another.

The horizontal sliding procedure is illustrated in Figure 7. To efficiently access the nodes to be slid on the sliding level, we maintain horizontal level-pointers between nodes which are neighbors on the level. As groups do not necessarily align with subtree boundaries, neighbors on the level may have a lowest common ancestor which is further away than the sliding distance. Since this ancestor is among the nodes whose keys should be changed due to the slide (to maintain search tree order), we maintain pointers to these ancestors for all pairs of neighboring nodes on a level which are not siblings (these pointers and the level-pointers are not shown in Figure 7). When sliding, subtrees below the slide level must be moved along with the keys to maintain search tree order of the keys (as shown in Figure 7). In that process, the ancestor pointers between the edges of these subtrees may have to be updated, which for each subtree is a number of pointers proportional to its height. Thus, sliding a hole from another node in the current group on a level at height i will require accessing $O(i2^i k)$ nodes, since the group size is $2^i k$.



■ **Figure 8** Illustrating the vertical redistribution. A hole is moved down one layer, thus allowing the creation of a new node on said layer. This corresponds directly to splitting in standard B-trees.

A vertical redistribution transforms one hole on a level $l + 1$ into B holes on level l . It is illustrated in Figure 8. This operation is the same as the split operation in standard B-trees. To perform a vertical redistribution, one clearly needs to access less nodes than for a slide at the same level.

4.3 Insertion

We now describe how to insert a new element with a given key. As in standard B-trees, we first search for the key in the tree to find a leaf node, v . If v contains a hole, we simply add the element to v as in a standard B-tree and no further work is done. Otherwise, a hole is first moved to v to make room for the new element. This is done by searching the group of v for a hole to slide to v . If none exists, we ask for the parent of v to obtain a hole, which we then can redistribute to v 's level as B holes. If the parent has no hole, the process is repeated recursively. This recursive process, called `REQUEST`, is described as Algorithm 1. The recursion ends at the latest when the reservoir is reached.

■ **Algorithm 1** `REQUEST`.

```

procedure REQUEST( $v$ )
  if any node in  $v$ 's group has a hole then
    slide this hole to  $v$ 
  else
    REQUEST(parent node of  $v$ )
    redistribute a hole from parent
    if  $v$ 's group is too big then
      split  $v$ 's group
    end if
  end if
end procedure

```

Moving down holes from higher levels via a vertical redistribution increases the size of the receiving group by one (see Figure 8). To keep the sizes of groups, and hence the cost of sliding within groups, we split a group in two when a redistribution discovers that the size of the receiving group has doubled compared to its initial group size from Section 4.1. Groups are simply implemented by marking the border nodes of groups, so this splitting is straight-forward to carry out as part of the operation.

As each redistribution provides B holes to a group, Algorithm 1 will only recurse upwards (the **else** case) from a group when B new calls of Algorithm 1 to nodes in the group have been issued since the last recursion upwards from this group. Recall that a horizontal slide on a level at height i has a cost of $O(i2^i k)$ and that vertical redistributions are also covered by this bound. From this follows an amortized bound on the rebalancing cost of:

$$k + \frac{1 \cdot 2k}{B} + \frac{2 \cdot 2^2 k}{B^2} + \dots + \frac{i \cdot 2^i k}{B^i} + \dots + \frac{h_{\max} \cdot 2^{h_{\max}} k}{B^{h_{\max}}} \quad (1)$$

which is $O(k)$ (assuming $B > 2$). As is standard, this can formally be proven by a potential function which amounts to each insertion bringing an amount of “coins” equal to Equation (1). Coins are conceptually stored in groups and will pay for all rebalancing work. When splitting a group, the new group has an additional need for coins not covered by Equation (1). As groups grow slowly, the extra amount compared to Equation (1) is low order in its terms, so just doubling Equation (1) is more than enough.

4.4 Achieving log squared amortized rebalancing

We now describe how to modify the rebalancing procedure such that the amortized cost per insertion becomes:

$$O\left(\frac{k \log^2(\min\{k, B\})}{B}\right).$$

Note that the amortized cost of rebalancing on all non-leaf levels is already as low as $O\left(\frac{k}{B}\right)$. This can be seen by excluding the first term in Equation (1). Since $k = \Theta(1/\epsilon)$, that value matches the lower bound, so we only need to lower the cost of rebalancing on the leaf level.

The overall plan is to distribute holes more evenly within leaf groups, both when constructing the tree initially and when vertically redistributing holes from the next higher level. We will use a density keeping scheme [17] on the nodes within each leaf group. Concretely we will use the version described in section 3.1 of [9] (to which we refer for full details). The scheme maintains a distribution of holes in a binary search tree by enforcing density (number of keys relative to the maximum for a given height) bounds on the levels of the tree. In this scheme, each insertion will require redistribution within an interval of size $O(\log^2(n)/\tau_1)$ (amortized) where n is the size of the array and τ_1 is upper limit on the density for the array (τ_1 can be set to $1/2$ here).

We apply this scheme to each group of k leaves such that each leaf node within a group is treated like a leaf node in the binary tree of the density keeping scheme. The binary tree is implicitly overlaid on the group and not actually constructed. The analysis in [9] requires that redistributing everything in a subtree takes linear time in the size of the subtree. We point out that this requirement is satisfied in this setting as the number of nodes involved in rebuilding an interval on the leaf level will be dominated by the distance on the leaf level (as described regarding sliding in Section 4.3).

For $B \geq k$, we must ensure that a node will sustain $\Theta(B/k)$ insertions per request for more holes to achieve the desired amortized cost. To this end, we specify how many holes a node gets from its neighbors via a request (slide) and how many holes a node must have in order to be able to give some holes away (this specifies whether the density keeping scheme should consider the node full). We split the B/k holes in each node into two equal portions: one to handle insertions into the node and one to service a request from another node. This means that a node issues a request only after $\Theta(B/k)$ keys have been inserted into it and that a node is given $\Theta(B/k)$ holes when it issues a request for holes. A node can only help another node (potentially itself) once – this is also the case in the setting of [9] as each node has capacity one. The amortized number of insertions between each request within the group is now $\Omega(k/B)$ and hence the amortized cost is $O\left(\frac{k \log^2(k)}{B}\right)$.

For $k > B$, not all nodes can get a hole per vertical redistribution. Instead, we apply the density keeping scheme to subgroups of k/B nodes where each such subgroup gets one hole after a vertical redistribution. The distance to a hole within a subgroup is $O(k/B)$ and as there will be B subgroups per actual group, using the density keeping scheme will mean that insertions will have an amortized cost of $O\left(\frac{k \log^2(B)}{B}\right)$.

It is important to note that, in both cases, $\Theta(B)$ of the holes are used before a vertical redistribution is requested (this is necessary to cover the cost of global rebuilding).

To handle deletions, the operations can be performed in reverse, as mentioned earlier. This implies moving holes upwards in the tree (equivalent to merge in standard B-trees) when the number of holes in a group grows sufficiently big.

5 Global rebalancing scheme

We here describe how repeated global rebuilding can be used with the scheme presented in Section 4 to achieve a global rebalancing scheme (i.e., not bound to a specific range of tree sizes as in Theorem 5).

Immediately before performing a global rebuild, the tree contains $n = N(1 - \epsilon)$ elements for some ϵ , where N is the next power of B . We set up the system from Section 4 by performing a global rebuild at linear cost (i.e., $O(n/B)$ node updates). We use this set-up while $N(1 - 2\epsilon) \leq n \leq N(1 - \epsilon/2)$. Once one of these bounds have been reached, we rebuild again, updating ϵ by increasing or decreasing it by a factor of two. There will be at least $\Theta(N\epsilon)$ updates between each rebuilding. Thus, the amortized cost incurred by the rebuilding process per update will be $O(1/(B\epsilon))$, which does not increase the upper bound from Section 4.

Applying this scheme during insertions (with ϵ successively decreasing by factors of two along the way) allows us to maintain optimal height until the tree is arbitrarily full and the update cost hence is arbitrarily close to $\Theta(n/B)$. In practice, one would presumably choose to allow suboptimal height when the complexity of rebalancing exceeds a certain rebalancing budget. Figure 1 illustrates how the complexity (lower bound) behaves as n approaches powers of B and how this complexity lower bound relates to a logarithmic budget. Suppose we want to limit the update complexity to a given budget $f(n)$. We can do that by keeping optimal height from $n = B^{h-1}$ up to the point where the budget is exceeded and from that point up to $n = B^h$ allowing the height to be optimal plus one. Specifically, we define ϵ' by $k \log^2(\min\{k, B\})/B = f(n)$ and $k = 1/\epsilon'$. For most $f(n)$, this has for $B^{h-1} < n \leq B^h$ one solution ϵ' for each h , similarly to Figure 1. We assume this to be the case for the $f(n)$ in question. We adjust the scheme above in the following way:

- If n exceeds $B^h(1 - \epsilon'/2)$, rebuild the tree with the height incremented by one and use $\epsilon = 1/2$.
- If n falls below $B^h(1 - \epsilon')$, rebuild the tree with the height decremented by one and use $\epsilon = \epsilon'$.

This gives the result below, where ϵ' is defined as above.

► **Theorem 6.** *There exists a rebalancing scheme such that, given a rebalancing budget $f(n)$, a B-tree can be maintained with perfect height from B^{h-1} to $(1 - \epsilon')B^h$ and with perfect height plus one up to B^h .*

We highlight two interesting special cases. The first case is $f(n) = \Theta(\log_B(n))$, which is a natural choice since it allocates the same cost for rebalancing as for searching time. For this choice, $\epsilon' = o(1)$, leading to:

► **Corollary 7.** *There exists a rebalancing scheme that given a rebalancing budget of $\Theta(\log_B(n))$ maintains a B-tree which has optimal height for all but a vanishing fraction of values of n . For the remaining values of n the height is optimal plus one.*

A comparison of the result of Corollary 7 to optimal height and to the height bound obtained by standard B-trees is given in Figure 2.

The second case is $f(n) = \Theta(1/B)$, for which we have $\epsilon' = \Theta(1)$, leading to:

► **Corollary 8.** *There exists a rebalancing scheme that given a rebalancing budget of $\Theta(1/B)$ maintains a B-tree with optimal height plus one.*

6 Storage utilization

We here consider the connection between the parameter ϵ and the storage utilization.

Our upper bound rebalancing scheme directly implies excellent storage utilization. When we lay out a tree with a given k this tree will have a storage utilization of at least $(k-1)/k = 1 - \epsilon$. This is trivially true for the leaf level as each group consisting of a multiple of k leaves has at most one empty node worth of holes (refer to Section 4.1). As holes are more sparse on higher levels of the tree, these levels have higher storage utilization than the leaves. The total weight of holes in the reservoir is at most a constant times the total weight of holes in the leaves. Since the number of nodes allocated in the reservoir is bounded by the number of allocated leaves and the weight per hole in the reservoir will be greater (by a factor of a power of B), the storage utilization among nodes in the reservoir is at least that of the leaves, too.

The scheme in Section 5 changes ϵ and thus the guaranteed storage utilization while growing a tree. To get a consistent high storage utilization while growing a tree, one can instead choose a desired low ϵ' (lower than needed initially for $n = (1 - \epsilon)N$), lay out the holes among the leaves and internal layers according to this ϵ' , and leave the extra free storage this would yield in the reservoir. By the same arguments as before, the storage utilization on the leaf and inner levels of the tree would then be $1 - \epsilon'$. On the reservoir level, one could compact the nodes such that at most one node would be non-full.

Our lower bound results do not translate directly into a statement about storage utilization. This result is about the amount of room (weight) in a tree of a particular height – the actual number of nodes allocated within the tree is not considered. As a counter-example to implications for storage utilization, take for instance a B-tree with all nodes completely full except the root which has only a tiny fraction of the B possible children. This tree will have excellent storage utilization – it will approach 1 for increasing n – while ϵ will be large as the number of keys could be increased massively without increasing tree height.

References

- 1 Arne Andersson. Optimal bounds on the dictionary problem. In G. Goos, J. Hartmanis, D. Barstow, W. Brauer, P. Brinch Hansen, D. Gries, D. Luckham, C. Moler, A. Pnueli, G. Seegmüller, J. Stoer, N. Wirth, and Hristo Djidjev, editors, *Optimal Algorithms*, volume 401, pages 106–114. Springer Berlin Heidelberg, Berlin, Heidelberg, 1989. doi:10.1007/3-540-51859-2_10.
- 2 Arne Andersson. *Efficient Search Trees*. PhD Thesis, Department of Computer Science, Lund University, Sweden, 1990.
- 3 Arne Andersson, Christian Icking, Rolf Klein, and Thomas Ottmann. Binary search trees of almost optimal height. *Acta Informatica*, 28(2):165–178, February 1990. doi:10.1007/BF01237235.
- 4 Arne Andersson and Tony W. Lai. Fast updating of well-balanced trees. In John R. Gilbert and Rolf Karlsson, editors, *SWAT 90*, pages 111–121. Springer Berlin Heidelberg, 1990.
- 5 Arne Andersson and Tony W. Lai. Comparison-efficient and write-optimal searching and sorting. In Wen-Lian Hsu and R. C. T. Lee, editors, *ISA '91 Algorithms*, pages 273–282. Springer Berlin Heidelberg, 1991.

- 6 David M. Arnow and Aaron M. Tenenbaum. An empirical comparison of B-trees, compact B-trees and multiway trees. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, page 33. ACM Press, 1984. doi:10.1145/602259.602265.
- 7 R. Bayer and E. M. McCreight. Organization and maintenance of large ordered indexes. *Acta Informatica*, 1(3):173–189, September 1972. doi:10.1007/BF00288683.
- 8 Rudolf Bayer and Karl Unterauer. Prefix B-trees. *ACM Transactions on Database Systems*, 2(1):11–26, March 1977. doi:10.1145/320521.320530.
- 9 Gerth Stølting Brodal, Rolf Fagerberg, and Riko Jacob. Cache Oblivious Search Trees via Binary Trees of Small Height. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '02*, pages 39–48, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics. URL: <http://dl.acm.org/citation.cfm?id=545381.545386>.
- 10 Trevor Brown. B-slack Trees: Space Efficient B-Trees. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Alfred Kobsa, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Demetri Terzopoulos, Doug Tygar, Gerhard Weikum, R. Ravi, and Inge Li Gørtz, editors, *Algorithm Theory – SWAT 2014*, volume 8503, pages 122–133. Springer International Publishing, Cham, 2014. doi:10.1007/978-3-319-08404-6_11.
- 11 Trevor Brown. B-slack trees: Highly Space Efficient B-trees. *arXiv:1712.05020 [cs]*, December 2017. arXiv:1712.05020.
- 12 Douglas Comer. Ubiquitous B-Tree. *ACM Comput. Surv.*, 11(2):121–137, June 1979. doi:10.1145/356770.356776.
- 13 Paul F. Dietz, Joel I. Seiferas, and Ju Zhang. A tight lower bound for on-line monotonic list labeling. In Erik M. Schmidt and Sven Skyum, editors, *Algorithm Theory — SWAT '94*, pages 131–142. Springer Berlin Heidelberg, 1994.
- 14 Rolf Fagerberg. Binary search trees: How low can you go? In G. Goos, J. Hartmanis, J. Leeuwen, Rolf Karlsson, and Andrzej Lingas, editors, *Algorithm Theory — SWAT'96*, volume 1097, pages 428–439. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996. doi:10.1007/3-540-61422-2_151.
- 15 Rolf Fagerberg. The complexity of rebalancing a binary search tree. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 72–83. Springer, 1999.
- 16 Scott Huddleston and Kurt Mehlhorn. Robust balancing in B-trees. In *Theoretical Computer Science*, pages 234–244. Springer, 1981.
- 17 Alon Itai, Alan G. Konheim, and Michael Rodeh. A sparse table implementation of priority queues. In Shimon Even and Oded Kariv, editors, *Automata, Languages and Programming*, pages 417–431. Springer Berlin Heidelberg, 1981.
- 18 Donald E. Knuth. *The Art of Computer Programming, Volume 3: (2Nd Ed.) Sorting and Searching*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.
- 19 Klaus Küspert. Storage utilization in B*-trees with a generalized overflow technique. *Acta Informatica*, 19(1), April 1983. doi:10.1007/BF00263927.
- 20 Tony W. Lai and Derick Wood. Updating almost complete trees or one level makes all the difference. In Christian Choffrut and Thomas Lengauer, editors, *STACS 90*, pages 188–194. Springer Berlin Heidelberg, 1990.
- 21 Tony Wen Hsun Lai. *Efficient Maintenance of Binary Search Trees*. PhD Thesis, University of Waterloo, Waterloo, Ont., Canada, Canada, 1990.
- 22 David Maier and Sharon C. Salveter. Hysterical B-trees. *Information Processing Letters*, 12(4):199–202, August 1981. doi:10.1016/0020-0190(81)90101-0.
- 23 H.A. Maurer, Th. Ottmann, and H.-W. Six. Implementing dictionaries using binary trees of very small height. *Information Processing Letters*, 5(1):11–14, May 1976. doi:10.1016/0020-0190(76)90094-6.

35:16 On Optimal Balance in B-Trees

- 24 Arnold L. Rosenberg and Lawrence Snyder. Compact B-trees. In *Proceedings of the 1979 ACM SIGMOD international conference on Management of data*, page 43. ACM Press, 1979. doi:10.1145/582095.582102.
- 25 Balasubramaniam Srinivasan. An Adaptive Overflow Technique to Defer Splitting in B-trees. *The Computer Journal*, 34(5):397–405, 1991. doi:10.1093/comjnl/34.5.397.
- 26 Andrew Chi-Chih Yao. On random 2–3 trees. *Acta Informatica*, 9(2):159–170, June 1978. doi:10.1007/BF00289075.

How Does Object Fatness Impact the Complexity of Packing in d Dimensions?

Sándor Kisfaludi-Bak

Max Planck Institut für Informatik, Saarbrücken, Germany
sandor.kisfaludi-bak@mpi-inf.mpg.de

Dániel Marx

Max Planck Institut für Informatik, Saarbrücken, Germany
dmarx@cs.bme.hu

Tom C. van der Zanden

Department of Data Analytics and Digitalisation, Maastricht University, The Netherlands
T.vanderZanden@maastrichtuniversity.nl

Abstract

Packing is a classical problem where one is given a set of subsets of Euclidean space called objects, and the goal is to find a maximum size subset of objects that are pairwise non-intersecting. The problem is also known as the Independent Set problem on the intersection graph defined by the objects. Although the problem is NP-complete, there are several subexponential algorithms in the literature. One of the key assumptions of such algorithms has been that the objects are fat, with a few exceptions in two dimensions; for example, the packing problem of a set of polygons in the plane surprisingly admits a subexponential algorithm. In this paper we give tight running time bounds for packing similarly-sized non-fat objects in higher dimensions.

We propose an alternative and very weak measure of fatness called the stabbing number, and show that the packing problem in Euclidean space of constant dimension $d \geq 3$ for a family of similarly sized objects with stabbing number α can be solved in $2^{O(n^{1-1/d}\alpha)}$ time. We prove that even in the case of axis-parallel boxes of fixed shape, there is no $2^{o(n^{1-1/d}\alpha)}$ algorithm under ETH. This result smoothly bridges the whole range of having constant-fat objects on one extreme ($\alpha = 1$) and a subexponential algorithm of the usual running time, and having very “skinny” objects on the other extreme ($\alpha = n^{1/d}$), where we cannot hope to improve upon the brute force running time of $2^{O(n)}$, and thereby characterizes the impact of fatness on the complexity of packing in case of similarly sized objects. We also study the same problem when parameterized by the solution size k , and give a $n^{O(k^{1-1/d}\alpha)}$ algorithm, with an almost matching lower bound: there is no algorithm with running time of the form $f(k)n^{o(k^{1-1/d}\alpha/\log k)}$ under ETH. One of our main tools in these reductions is a new wiring theorem that may be of independent interest.

2012 ACM Subject Classification Theory of computation \rightarrow Computational geometry

Keywords and phrases Geometric intersection graph, Independent Set, Object fatness

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.36

Related Version A full version of the paper is available at <http://arxiv.org/abs/1909.12044>.

Funding *Sándor Kisfaludi-Bak*: Supported by NWO Gravitation Grant Networks (No. 024.002.003).
Dániel Marx: Supported by ERC Consolidator Grant SYSTEMATICGRAPH (No. 725978).

1 Introduction

Many well-known NP-hard problems (e.g. INDEPENDENT SET, HAMILTON CYCLE, DOMINATING SET) can be solved in time $2^{O(\sqrt{n})}$ when restricted to planar graphs, while only $2^{O(n)}$ algorithms are known for general graphs [11–16, 18, 24, 28, 30]. This beneficial effect of planarity is known as the “square root phenomenon,” and can be exploited also in the context of 2-dimensional geometric problems where the problem is defined on various intersection



© Sándor Kisfaludi-Bak, Dániel Marx, and Tom C. van der Zanden;
licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 36; pp. 36:1–36:18

Leibniz International Proceedings in Informatics



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

36:2 How Does Object Fatness Impact Packing?

graphs in \mathbb{R}^2 [3, 4, 17, 25]. In particular, consider the geometric packing problem where, given a set of polygons in \mathbb{R}^2 , the task is to find a subset of k pairwise disjoint polygons. This problem can be solved in time $n^{O(\sqrt{k})}$ [25], which – when expressed only as a function of the input – gives an $n^{O(\sqrt{n})} = 2^{O(\sqrt{n} \log n)}$ algorithm for finding a maximum size disjoint subset.

Can these 2-dimensional subexponential algorithms be generalized to higher dimensions? It seems that the natural generalization is to aim for $2^{O(n^{1-1/d})}$, or in case of parameterized problems, either $2^{O(k^{1-1/d})} \cdot n^{O(1)}$ or $n^{O(k^{1-1/d})}$ time algorithms in d -dimensions: the literature contains upper and lower bounds of this form (although sometimes with extra logarithmic factors in the exponent) [9, 26, 29]. However, all of these algorithms have various restrictions on the object family on which the intersection graph is based: there is no known analogue of the $n^{O(\sqrt{k})}$ time algorithm of Marx and Pilipczuk [25] in higher dimensions with the same generality of objects. There is a good reason for this: it is easy to see that any n -vertex graph can be expressed as the intersection graph of 3-dimensional simple polyhedra. Thus a subexponential algorithm for 3-dimensional objects without any severe restriction would give a subexponential algorithm for INDEPENDENT SET on general graphs, violating standard complexity-theoretic assumptions.

What could be reasonable restrictions on the objects that allow running times of the form, e.g., $2^{O(n^{1-1/d})}$? One of the most common restrictions is to study a set $F \subseteq 2^{\mathbb{R}^d}$ of *fat objects*, where for each object $o \in F$ the ratio $\text{radius}(B_{\text{in}}(o)) / \text{radius}(B_{\text{out}}(o))$ is at least some fixed positive constant. (We denote by $\text{radius}(B_{\text{in}})$ and $\text{radius}(B_{\text{out}})$ the radius of the inscribed and circumscribed ball respectively.) Another common restriction is to have *similarly sized* objects, that is, a family F where the ratio of the largest and smallest object diameter is at most some absolute constant. Many results concern only *unit disk graphs*, where F consists of unit disks in the plane: unit disks are both fat and similarly sized. The focus of our paper is to explore the role of fatness in the context of packing problems and to understand when and to what extent fatness decreases the complexity of the problem. We observe that fatness is a crucial requirement for subexponential algorithms in higher dimensions, and this prompts us to explore in a quantitative way how fatness influences the running time. For this purpose, we introduce a parameter α describing the fatness of the objects and give upper and lower bounds taking into account this parameter as well.

More precisely, we introduce the notion of the *stabbing number*, which can be regarded as an alternative measure of fatness. This slightly extends a similar definition by Chan [6]. We say that an object o is stabbed by a point p if $p \in o$. A family of objects $F \subseteq 2^{\mathbb{R}^d}$ is α -stabbed if for any $r \in \mathbb{R}$, the subset of F -objects o of diameter $\text{diam}(o) \in [r/2, r]$ contained in any ball of radius r can be stabbed by α^d points. The stabbing number of F is defined as $\inf_{\alpha \in [1, \infty)} \{F \text{ is } \alpha\text{-stabbed}\}$. Note that a set of n objects in d -dimensions has stabbing number at most $n^{1/d}$. The stabbing number is closely related to the inverse of a common measure of fatness. This relationship is explored in Section 2.

By adapting a separator theorem from [9], we can give an algorithm where the running time smoothly goes from $2^{O(n^{1-1/d})}$ to $2^{O(n)}$ as the stabbing number goes from $O(1)$ to the maximum possible $n^{1/d}$.

► **Theorem 1.** *Let $\alpha \in [1, \infty)$ and $2 \leq d \in \mathbb{N}$ be fixed constants. There is an algorithm that solves INDEPENDENT SET for intersection graphs of similarly sized α -stabbed objects in \mathbb{R}^d running in time $2^{O(n^{1-1/d} \alpha)}$.*

As mentioned, the stabbing number is at most $n^{1/d}$, and this algorithm runs in subexponential time whenever the stabbing number is better than this trivial upper bound, that is, whenever $\alpha = o(n^{1/d})$ holds.

In order to have definite answers to the best running times achievable, we also need a lower bound framework. A popular starting point in the past decades is the Exponential Time Hypothesis (ETH) [21], which posits that there exists a constant $\gamma > 0$ such that there is no $2^{\gamma n}$ algorithm for the 3-SAT problem. Classical NP-hardness reductions automatically yield quantitative lower bounds on the running time under ETH. If enough care is taken to ensure that the constructed instance is sufficiently small, then one can find lower bounds that match the best known algorithms [8]. For the INDEPENDENT SET problem, a lower bound of $2^{\Omega(n)}$ is a consequence of classical reductions under ETH.

A standard way to explore the impact of a parameter such as fatness is to give an algorithm where the parameter appears in the running time, together with a matching lower bound. However, the notion of “matching lower bound” needs to be defined precisely if we are expressing the running time as a function of two parameters, the size n of the instance and the stabbing number α of the objects.

A recent example of such an algorithm and lower bound involving two parameters is the paper by Biró *et al.* [5], where it is shown that the coloring problem of unit disk graphs with $\ell = n^\lambda$ colors can be solved in $2^{O(\sqrt{n\ell} \log n)}$ time, where $\lambda \in [0, 1]$ is a fixed constant, and they also exclude algorithms of running time $2^{o(\sqrt{n\ell})}$ under ETH. This is interesting since this smoothly bridges the gap between a standard “square root phenomenon” algorithm ($\ell = O(1)$) on one extreme and the brute force $2^{O(n)}$ on the other ($\ell = n^{1-o(1)}$). Our results show a similar behavior in the context of fatness and the packing problem: the running time of Theorem 1 is optimal, with the running time smoothly going from $2^{O(n^{1-1/d})}$ time in the case of $\alpha = O(1)$ to the trivial $2^{O(n)}$ time of brute force when $\alpha = n^{1/d}$.

Let $\mathcal{G}(d, L)$ denote the set of intersection graphs in \mathbb{R}^d where each object is an axis-parallel box whose side lengths form the multiset $\{1, \dots, 1, L\}$. Let us call such an axis-parallel box *canonical*. As usual, n denotes the number of objects (the number of vertices in the graph).

For example, it is easy to see that $1 \times 1 \times L$ boxes have stabbing number $O(L^{2/3})$. Any collection of $1 \times 1 \times L$ boxes of the same orientation can be stabbed by the lattice generated by the vertices of such a box, which has $O(L^2)$ points in a ball of radius $O(L)$. By taking the same lattice for the two other orientations, we obtain a complete stabbing set of size $O(L^2)$ inside a ball of radius $O(L)$ for all axis-parallel boxes of this shape. In general for $d \geq 3$, the stabbing number for canonical boxes is $\alpha = O(L^{1-1/d})$, so in particular, for $L = 1$ we have $\alpha = O(1)$, and for $L \geq n^{1/(d-1)}$ we have $\alpha = O(n^{1/d})$. In our main contribution, we show that this very restricted set of non-fat objects is sufficient to prove the desired lower bound.

► **Theorem 2.** *Let $d \geq 3$ be fixed. Then there is a constant $\gamma > 0$ such that for all $\alpha \in [1, n^{1/d}]$ it holds that INDEPENDENT SET on intersection graphs of d -dimensional canonical axis-parallel boxes of stabbing number α has no algorithm running in time $2^{\gamma n^{1-1/d} \alpha}$, unless ETH fails.*

An immediate corollary is that the $2^{O(n)}$ time brute-force algorithm cannot be improved, even for the intersection graph of axis-parallel boxes. This Corollary 3 can also be derived from a simpler construction by Chlebík and Chlebíková [7].

► **Corollary 3.** *Let $3 \leq d \in \mathbb{N}$ be fixed. Then INDEPENDENT SET on intersection graphs of axis-parallel boxes in d -dimensions has no algorithm running in time $2^{o(n)}$, unless ETH fails.*

In unit ball graphs, there is a lower bound of $2^{\Omega(n^{1-1/d})}$ under ETH, which of course carries over to intersection graphs of fat objects [9]. This latter reduction is based on establishing efficient routing constructions (called the “Cube Wiring theorem”) in the d -dimensional Euclidean grid. The crucial insight of the present paper is that tight lower bounds for *nonfat*

36:4 How Does Object Fatness Impact Packing?

objects can be obtained via INDEPENDENT SET on induced subgraphs of the d -dimensional *blown-up* grid cube, where each vertex is replaced by a clique of t vertices, fully connected to the adjacent cliques in all d directions. First we establish a lower bound for INDEPENDENT SET on subgraphs of such cubes (even for subgraphs of maximum degree 3), using and extending the Cube Wiring theorem [9]. Unlike for unit balls, it now seems difficult to realize every such subgraph G as intersection graph of appropriate boxes. Instead, we realize a graph G' that is obtained from G by some number of double subdivisions (subdividing some edge twice). As every double subdivision is known to increase the size of the maximum independent set by exactly 1, switching to G' does not cause a problem in the reduction.

The key insight of the reduction (in 3-dimensions) is that if $t = L^2$, then t vertices can be represented with $1 \times 1 \times L$ size boxes arranged in an $L \times L$ grid, occupying $O(L) \times O(L) \times O(L)$ space. Each t -clique of the blown-up cube is represented by such arrangements of boxes. The main challenge that we have to overcome is that the subgraph G may contain an arbitrary matching between two adjacent t -cliques. Given two sets of $1 \times 1 \times L$ size boxes arranged in two $L \times L$ grids, it seems unclear whether such arbitrary connections can be realized while staying in an $O(L) \times O(L) \times O(L)$ region of space. However, we show that this is possible, as the $L \times L$ grid arrangement allows easy reordering within the rows or within the columns, and it is known that any permutation of a grid can be obtained as doing a permutation first within the rows, then within the columns, and finally one more time within the rows. Thus with some effort, it is possible to build gadgets representing $L \times L$ vertices in an $O(L) \times O(L) \times O(L)$ region of space that allows arbitrary matchings to be realized with the adjacent gadgets.

The idea is similar in higher dimensions $d > 3$. We reduce from the INDEPENDENT SET problem on a subgraph of the blow-up of a d -dimensional grid where each vertex is blown-up into a clique of L^{d-1} vertices. Each gadget now contains L^{d-1} boxes of size $1 \times 1 \times \dots \times 1 \times L$ arranged in a grid. In order to implement arbitrary matchings between adjacent gadgets, we decompose every permutation of the $(d-1)$ -dimensional grid into $O(d)$ simpler permutations that are easy to realize in d -dimensional space.

We also study the complexity of packing in the context of parameterized algorithms: the question is how much one can improve the brute force $n^{O(k)}$ algorithm for finding k independent objects. We present a counterpart of Theorem 1 in this setting.

► **Theorem 4.** *Let $\alpha \in [1, \infty)$ and $2 \leq d \in \mathbb{N}$. There is a parameterized algorithm that solves independent set for intersection graphs of similarly sized α -stabbed objects in \mathbb{R}^d running in time $n^{O(k^{1-1/d}\alpha)}$, where the parameter k is the size of the maximum independent set.*

If one regards the parameterized algorithm's running time in terms of the instance size only, the result would be a $2^{O(n^{1-1/d}(\log n)^\alpha)}$ algorithm, which is slower than the running time $2^{O(n^{1-1/d}\alpha)}$ provided by the latter algorithm. The parameterized algorithm is based on a separator theorem by Miller *et al.* [27].

Finally, we sketch how the lower bound construction of Theorem 2 can be adapted to a parameterized setting, and obtain the following theorem:

► **Theorem 5.** *Let $3 \leq d \in \mathbb{N}$ be fixed. Then there is a constant $\gamma > 0$ such that for all $\alpha \in [1, n^{1/d}]$ it holds that deciding if there is an independent set of size k in intersection graphs of d -dimensional canonical axis-parallel boxes of stabbing number α has no $f(k)n^{\gamma k^{1-1/d}\alpha/\log k}$ algorithm for any computable function f , unless ETH fails.*

The crucial difference is that we are reducing from the PARTITIONED SUBGRAPH ISOMORPHISM problem instead of INDEPENDENT SET, which means that instead of choosing or not choosing a box (representing choosing or not choosing a vertex in the INDEPENDENT

SET problem), the solution needs to choose one of n very similar boxes (representing the choice of one of n vertices in a class of the partition). The overall structure of the reduction (e.g., routing in the blown-up d -dimensional grid) is similar to the proof of Theorem 2, and it can be found in the full version [23] along with other missing proofs.

2 The relationship between the stabbing number and fatness

In the usual definition of fatness, an object $o \subset \mathbb{R}^d$ is α -fat if there exists a ball of radius ρ_{in} contained in o and a ball of radius ρ_{out} that contains o , where $\rho_{\text{in}}/\rho_{\text{out}} = \alpha$. For a fixed constant α this is a useful definition and unifies many other similar notions in case of convex objects, i.e., it holds that a set of convex objects that is constant-fat for this notion of fatness are constant-fat for more restrictive definitions and vice versa. For our purposes however this definition is not fine-grained enough in the following sense. The fatness of a $1 \times 1 \times n$ box in three dimensions would be $\Theta(n)$, just as the fatness of a $1 \times n \times n$ box. As it will be apparent in what follows, we need a fatness definition according to which $1 \times n \times n$ boxes are much more fat than $1 \times 1 \times n$ boxes. For this purpose, we use the following weaker definition of fatness, that tracks the volume compared to a circumscribed ball more closely. (Note that constant-fat objects are also weakly constant-fat.)

► **Definition 6 (Weakly α -fat).** *A measurable object $o \subseteq \mathbb{R}^d$ is α -fat for some $\alpha \in [1, \infty)$ if $\text{Vol}(o)/\text{Vol}(B) \leq \alpha^d$, where $\text{Vol}(o)$ and $\text{Vol}(B)$ denotes the volume of o and the volume of its circumscribed ball B respectively.*

An object o is *strongly α -fat* if for any ball B centered inside o we have $\text{Vol}(B \cap o)/\text{Vol}(B) \geq \alpha^d$. In case of convex objects, weak fatness coincides with strong fatness up to constant factors, see [31].

The next theorem shows that the inverse of the weak fatness of an object family is related to the stabbing number. In a sense, this means that the stabbing number is a further weakening of weak fatness. Note that in our setting, the stabbing number will be polynomial in n (i.e., $\alpha = n^\lambda$ for some constant λ), so the $\log n$ term is insignificant.

► **Theorem 7.** *Let d be a fixed constant. Then the stabbing number of any family of n weakly $(1/\alpha)$ -fat (measurable) objects in \mathbb{R}^d is $O(\alpha \log^{1/d} n)$.*

Proof. Consider a family F of weakly $1/\alpha$ -fat objects. Let B be a ball of radius δ , and let F_B be the set of objects contained in B of diameter at least $\delta/2$. It is sufficient to show that we can stab F_B with $O(\alpha^d \log n)$ points. Pick $k = \lfloor (4\alpha)^d (\log n + 1) \rfloor$ points p_1, \dots, p_k independently uniformly at random in B . For any given object o , its volume is at least $\text{Vol}(B)/(4\alpha)^d$, so the probability that a given p_i is not in o is at most $1 - 1/(4\alpha)^d$. Since the k points are chosen independently, the probability that a given object o is unstabbed is at most $\left(1 - \frac{1}{(4\alpha)^d}\right)^k$. By the union bound, the probability that there is an unstabbed object is at most

$$n \left(1 - \frac{1}{(4\alpha)^d}\right)^k = n \left(1 - \frac{1}{(4\alpha)^d}\right)^{\lfloor (4\alpha)^d (\log n + 1) \rfloor} < n(1/e)^{\log n + 1} < 1.$$

Consequently, there exists an outcome where all objects are stabbed. ◀

We conclude this section with the following theorem, which shows an even stronger connection between fatness and stabbing in case of convex objects. The theorem uses the existence of the John ellipsoid [22] and the ε -net theorem [20].

36:6 How Does Object Fatness Impact Packing?

► **Theorem 8.** *Let d be a fixed constant. Then the stabbing number of any family of n weakly $(1/\alpha)$ -fat convex objects in \mathbb{R}^d is $O(\alpha \log^{1/d} \alpha)$.*

Proof. Consider a family F of weakly $1/\alpha$ -fat convex objects. Let B be a ball of radius δ , and let F_B be the set of objects contained in B of diameter at least $\delta/2$. It is sufficient to show that we can stab F_B with $O(\alpha^d \log \alpha)$ points. For any given object o , its volume is at least $\text{Vol}(B)/(4\alpha)^d$. Every convex object $o \in F_B$ contains an ellipsoid $\ell(o) \subseteq o$ such that $\text{Vol}(o)/\text{Vol}(\ell(o)) \leq d^d$ [22]. Since the VC-dimension of ellipsoids in \mathbb{R}^d is $O(d^2)$ [2], the ϵ -net theorem [20] implies that the ellipsoids $\ell(o)$ ($o \in F_B$) can be stabbed by $O(\frac{d^2}{1/(4\alpha)^d} \log \frac{d^2}{1/(4\alpha)^d}) = O(\alpha^d \log \alpha)$ points. Since the ellipsoids are contained in their respective objects, this point set also stabs all objects in F_B . ◀

3 Algorithms

We require very little from the objects that we use in our algorithms. It is necessary that we can decide in polynomial time whether a point is contained in an object, whether two objects intersect, and whether an object intersects some given sphere, ball, and empty or dense hypercube. Let us assume that such operations are possible from now on.

To prove Theorem 4, we use the following separator theorem, due to Miller et al. [27]. The *ply* of a set of objects in \mathbb{R}^d is the largest number p such that there exists a point $x \in \mathbb{R}^d$ which is contained in p objects.

► **Theorem 9** (Miller et al. [27]). *Let $\Gamma = \{B_1, \dots, B_n\}$ be a collection of n closed balls in \mathbb{R}^d with ply at most p . Then there exists a sphere S whose boundary intersects at most $O(p^{1/d} n^{1-1/d})$ balls, and the number of balls in Γ disjoint from S that fall inside and outside S are both at most $\frac{d+1}{d+2}n$.*

Proof sketch. Consider the set of balls B made up by the circumscribed balls of the objects in a maximum independent set. We claim that the ply of this set is $O(\alpha^d)$. To prove the claim, let S be a subset of the independent set whose circumscribed balls overlap at a point $x \in \mathbb{R}^d$. Since the objects are similarly sized, S must lie within a ball centered at x whose radius is at most a constant times the diameter of the largest object. Thus, S can be stabbed by $O(\alpha^d)$ points. However, as S forms an independent set, each point can only stab at most one object from S . Therefore, $|S| = O(\alpha^d)$. By Theorem 9 the ball set B has a sphere separator intersecting $O((\alpha^d)^{1/d} k^{1-1/d}) = O(k^{1-1/d} \alpha)$ balls. We proceed by guessing such a sphere: a discretization argument shows that there are $\text{poly}(n)$ distinct guesses for this sphere separator. We also guess the set of objects in the independent set that intersect the sphere, and remove all other objects intersecting the sphere or the guessed objects, and recurse on the remaining objects inside S and on the remaining objects outside S . The resulting running time is $n^{O(k^{1-1/d} \alpha)}$. ◀

For arbitrary size objects that are $O(1)$ -fat in some stronger sense (or just $O(1)$ -stabbed), we can apply the above scheme of guessing a separating sphere or hypercube, and use one of the many separator theorems designed for objects of small ply. See [6, 19, 29]. One can also apply [9] since in case of ply 1, the weights are constants; although the theorem is stated for the usual notion of fatness, the proof itself uses only the stabbing number. We get the following theorem.

► **Theorem 10.** *Let $2 \leq d \in \mathbb{N}$. There is a parameterized algorithm that solves INDEPENDENT SET for intersection graphs of $O(1)$ -stabbed objects in \mathbb{R}^d running in time $n^{O(k^{1-1/d})}$, where the parameter k is the size of the maximum independent set.*

The algorithm for Theorem 1 is an adaptation of the INDEPENDENT SET algorithm for fat objects from [9], based on weighted cliques, and its proof is deferred to the full version [23].

4 Wiring in a blowup of the Euclidean Cube

Our wiring theorem relies on the folklore observation that can be informally stated the following way: an $n \times m$ matrix can be sorted by first permuting the elements within each row, then permuting the elements within each column, and then permuting the elements in each row again. Note that the permutations are independent of each other, and they are not sorting steps; the permutations required are quite specialized. We state the lemma in a more group-theoretic setting. Let $\text{Sym}(X)$ denote the symmetric group on the set X .

► **Lemma 11** (Lemma 4 of [1]). *Let A and B be two finite sets. Then $\text{Sym}(A \times B) = G_A G_B G_A$, where G_A is the subgroup of $\text{Sym}(A \times B)$ consisting of permutations π where $\pi(a, b) \in A \times \{b\}$ for all $(a, b) \in A \times B$, and G_B is the subgroup of $\text{Sym}(A \times B)$ consisting of permutations π where $\pi(a, b) \in \{a\} \times B$ for all $(a, b) \in A \times B$.*

► **Corollary 12.** *Let $2 \leq d \in \mathbb{N}$ and let A_1, A_2, \dots, A_d be finite sets. Then $\Gamma \stackrel{\text{def}}{=} \text{Sym}(A_1 \times A_2 \times \dots \times A_d)$ is of the form $\Gamma = G_1 G_2 \dots G_{d-1} G_d G_{d-1} G_{d-2} \dots G_1$, where G_i is the subgroup of Γ consisting of permutations π where $\pi(a_1, \dots, a_i, \dots, a_d) \in \{a_1\} \times \dots \times \{a_{i-1}\} \times A_i \times \{a_{i+1}\} \times \dots \times \{a_d\}$ for all $(a_1, \dots, a_d) \in \Gamma$.*

Proof. We use induction on d ; for $d = 2$, the statement is equivalent to Lemma 11. Let $d \geq 3$. We can write Γ as $\text{Sym}((A_1 \times \dots \times A_{d-1}) \times A_d)$, so by induction (for $d = 2$), we have that $\Gamma = G_1 \times G_{A_2 \times \dots \times A_d} \times G_1$. By induction, we also have that $G_{A_2 \times \dots \times A_d} = G_2 \dots G_{d-1} G_d G_{d-1} G_{d-2} \dots G_2$, therefore $\Gamma = G_1 G_2 \dots G_{d-1} G_d G_{d-1} G_{d-2} \dots G_1$. ◀

For an integer n , let $[n] = \{1, \dots, n\}$. Let $\mathcal{EC}^d(n)$ be the d -dimensional Euclidean grid graph whose vertices are $[n]^d$, and $x, y \in V(G)$ are connected if and only if they are at distance 1 in \mathbb{R}^d . For $x \in \mathbb{Z}^d$ and $S \subset \mathbb{Z}^d$, we use the shorthand $x + S \stackrel{\text{def}}{=} \{x + y \mid y \in S\}$. Let $\mathcal{BEC}^d(n, t)$ denote the t -fold blowup of $\mathcal{EC}^d(n)$, where all vertices of $\mathcal{EC}^d(n)$ are exchanged with a clique of size t , and vertices in neighboring cliques are connected. More precisely,

$$\begin{aligned} V(\mathcal{BEC}^d(n, t)) &= [n]^d \times [t] \\ E(\mathcal{BEC}^d(n, t)) &= \{(x, i)(y, j) \mid x = y \vee (x, y) \in E(\mathcal{EC}^d(n))\}. \end{aligned}$$

Our second key ingredient is the Euclidean Cube Wiring theorem.

► **Theorem 13** (Theorem 21 in [9]). *Let $3 \leq d \in \mathbb{Z}$. There exists a constant c dependent only on the dimension such that any matching M between $P = [n]^{d-1} \times \{1\}$ and $Q \stackrel{\text{def}}{=} [n]^{d-1} \times \{cn\}$ can be embedded in $\mathcal{EC}^d(cn)$, that is, there is a set of vertex disjoint paths connecting p and q in $\mathcal{EC}^d(cn)$ for all $pq \in M$.*

► **Theorem 14** (Blown-up Cube Wiring). *Let $3 \leq d \in \mathbb{Z}$, and let n, t be positive integers. We consider two opposing facets of the blown-up cube $\mathcal{C} \stackrel{\text{def}}{=} \mathcal{BEC}^d(cn, t)$ (where $c \in \mathbb{Z}_+$ depends only on d):*

$$\begin{aligned} P &\stackrel{\text{def}}{=} ([n]^{d-1} \times \{1\}) \times [t] \\ Q &\stackrel{\text{def}}{=} ([n]^{d-1} \times \{cn\}) \times [t] \end{aligned}$$

Any matching M between P and Q can be embedded in \mathcal{C} , that is, there is a constant integer c dependent only on the dimension d such that for any matching M there is a set of vertex disjoint paths connecting p and q in $\mathcal{BEC}^d(cn, t)$ for all $pq \in M$.

Proof. Without loss of generality, suppose that M is a perfect matching between P and Q (this can be ensured by adding dummy edges to M if necessary). Let $c = c' + 2$ where c' is a constant such that cube wiring can be done in height $h = c'n$. Let $A = [n]^{d-1}$ and let $B = [t]$. The matching M can be regarded as a permutation π of $A \times B$, where $\pi(a, b) = (a', b')$ if $((a, b)(a', b')) \in M$.

By Lemma 11, there exists a permutation $\pi_A \in G_A$ and $\pi_B, \pi'_B \in G_B$ such that $\pi = \pi'_B \pi_A \pi_B$, where G_A and G_B are defined as in Lemma 11. We can think of both π_B and π'_B as the union of n^{d-1} distinct permutations of $[t]$. We can realize π_B using one matching: for all $(x, i) \in A \times B$, we add the edge $((x, 1), i)((x, 2), j)$ to M_B , where $\pi_B(x, i) = (x, j)$. As a result, M_B is a perfect matching between P and the next layer of the blown-up cube, $P' \stackrel{\text{def}}{=} ([n]^{d-1} \times \{2\}) \times [t]$. Similarly, for all $(x, i) \in A \times B$, let M'_B contain the edge $((x, cn - 1), i)((x, cn), j)$, where $\pi'_B(x, i) = (x, j)$; this matches $Q' \stackrel{\text{def}}{=} ([n]^{d-1} \times \{cn - 1\}) \times [t]$ to Q . Finally, by the Cube Wiring Theorem (Theorem 13), for each $i \in [t]$, there are vertex disjoint paths from $P'_i \stackrel{\text{def}}{=} ([n]^{d-1} \times \{2\}) \times \{i\}$ to $Q'_i \stackrel{\text{def}}{=} ([n]^{d-1} \times \{cn - 1\}) \times \{i\}$ that realizes the matching

$$M_A^i \stackrel{\text{def}}{=} \{(x, i)(y, i) \mid x \in [n]^{d-1} \text{ and } \pi_A(x, i) = (y, i)\}.$$

For each $i \in [t]$, these wirings are vertex disjoint since they are contained in vertex disjoint Euclidean grid hypercubes. The matchings M_A^i for $i \in [t]$ together with the matchings M_B and M'_B realize the matching M . \blacktriangleleft

5 Lower bounds for packing isometric axis-parallel boxes

Our first lower bound shows that the running time of the algorithm in Theorem 1 is tight under ETH.

Overview of the proof of Theorem 2. Our proof is a reduction from (3,3)-SAT, the satisfiability problem of CNF formulas where clauses have size at most three and each variable occurs at most three times. Such formulas have the property that if they have n variables, then they have $O(n)$ clauses. The problem has no $2^{o(n)}$ algorithm under ETH [10].

The proof has two steps; the first step is a reduction from (3,3)-SAT to INDEPENDENT SET in certain subgraphs of the blown-up Euclidean cube, and the second step is to show that these subgraphs can essentially be realized with axis-parallel boxes. Throughout the proof, we consider the dimension d to be a constant.

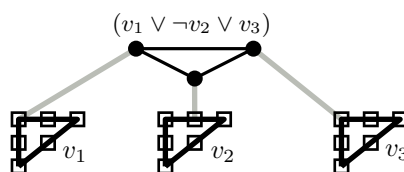
The *incidence graph* of a (3,3)-CNF formula ϕ is a graph where vertices correspond to clauses and variables of ϕ , and a variable and clause vertex are connected if and only if the variable occurs in the clause.

5.1 Independent Set in subgraphs of the blown-up Euclidean cube

A simple and generic lower bound construction for Independent Set

We give a generic reduction from (3,3)-SAT to INDEPENDENT SET, which serves as a skeleton for the more geometric type of reduction we will do later.

Consider the incidence graph of ϕ . Replace each variable vertex v with a cycle of length 6, consisting of vertices v^1, \dots, v^6 , where the edges formerly incident to v are now connected to distinct cycle vertices v^2, v^4 or v^6 for positive literals and to v^1, v^3 or v^5 for negative literals (see Figure 1). We replace each clause vertex w that corresponds to a clause of exactly 3 literals with a cycle of length three, and connect the formerly incident edges to distinct



■ **Figure 1** The graph G_ϕ for $\phi = (v_1 v_2 v_3)$.

vertices of the triangle. For clauses that have exactly two literals, the gadget is a single edge, and we connect the formerly incident edges to distinct endpoints of the edge. We can eliminate clauses of size 1 in a preprocessing step. Let G'_ϕ be the resulting graph.

An independent set can contain at most 3 vertices of a variable cycle of length 6, and at most 1 vertex per clause gadget. Observe that a formula with ν variables and γ clauses has an independent set of size $3\nu + \gamma$ if and only if the original formula is satisfiable.

Let G be a graph and let uv be an edge of G . A *double subdivision* of uv is replacing uv with a path of length 3, i.e., we add the new vertices w and w' , remove the edge uv and add the edges $uw, ww', w'v$. A graph that can be obtained from G by some sequence of double subdivisions is called an *even subdivision* of G . Observe that a double subdivision increases the size of the maximum independent set by one, so G has an independent set of size k if and only if its even subdivision G' has an independent set of size $k + \frac{|V(G')| - |V(G)|}{2}$.

Embedding G'_ϕ into a blown-up cube

In a blown-up cube $\mathcal{BEC}^d(n, t)$, we call a clique corresponding to $x \in [n]^d$ the *cell* of x or simply a cell, that is, the cell of x is defined as the set of vertices $\{x\} \times [t] \subset V(\mathcal{BEC}^d(n, t))$.

The following is a tight lower bound for INDEPENDENT SET inside the blown-up Euclidean cube.

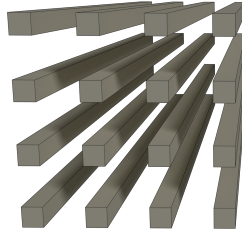
► **Theorem 15.** *For any fixed constant $d \geq 3$, there exists a $\gamma > 0$ such that for any $t \geq 2$ there is no $2^{\gamma n^{1-1/d} t^{1/d}}$ algorithm for INDEPENDENT SET for subgraphs of the blown-up cube $\mathcal{C} \stackrel{\text{def}}{=} \mathcal{BEC}^d((n/t)^{1/d}, t)$ under ETH. The lower bound holds even if the subgraph G has maximum degree three, and the neighbors of each vertex in G lie in distinct cells.*

Proof sketch. Given a (3,3)-SAT formula ϕ , we show that we construct a subgraph of a blown-up cube with the required properties that is also an even subdivision of G'_ϕ . If ϕ has \bar{n} literals, then we create a subgraph G that has $n = c \cdot \bar{n}^{\frac{d}{d-1}} / t^{\frac{1}{d-1}}$ vertices; a simple computation shows that this is sufficient. The variable cycles become cycles of length 6, and they are placed densely within cells that lie in some facet of \mathcal{C} . Similarly, for clauses of size two and three, we associate an edge or a triangle in the cells of the opposing facet of \mathcal{C} . Using Theorem 14, we can construct wires that for each literal connect the relevant vertex of the variable cycle to the relevant vertex of the clause cycle. If the resulting wire has even length, then we add an extra edge to its end that connects to the clause cycle. The resulting embedding has the desired properties. ◀

5.2 Detailed construction and gadgetry

Having established our lower bound for blown-up Euclidean cubes, we now need to construct a set of canonical boxes whose intersection graph is an even subdivision of a given subgraph with maximum degree three where the neighbors of each vertex lie in distinct cells.

36:10 How Does Object Fatness Impact Packing?



■ **Figure 2** A basic brick.

► **Theorem 16.** *Let $d \geq 3$ and $L \geq 16$ be fixed, and let G be a subgraph of the blown-up cube $\mathcal{C} = \mathcal{BEC}^d(s, (L/8)^{d-1})$ of maximum degree three, where the neighbors of each vertex lie in distinct cells. Then G has an even subdivision G' that can be realized using boxes of size $1 \times \dots \times 1 \times L$. Moreover, given G , the boxes of G' can be constructed in $O(|V(\mathcal{C})|)$ time, and $|V(G')| = O(|V(G)|)$.*

We consider $d = 3$ first; later on, we show how the construction can be generalized to higher dimensions. We need to define a set of boxes whose intersection graph is an even subdivision of G . The idea is to create a generic *module* that is able to represent a subgraph of G induced by any cell; these modules will take up $O(L) \times O(L) \times O(L)$ space. The modules are arranged into a larger cube of side length $O(sL)$ to make up the final construction.

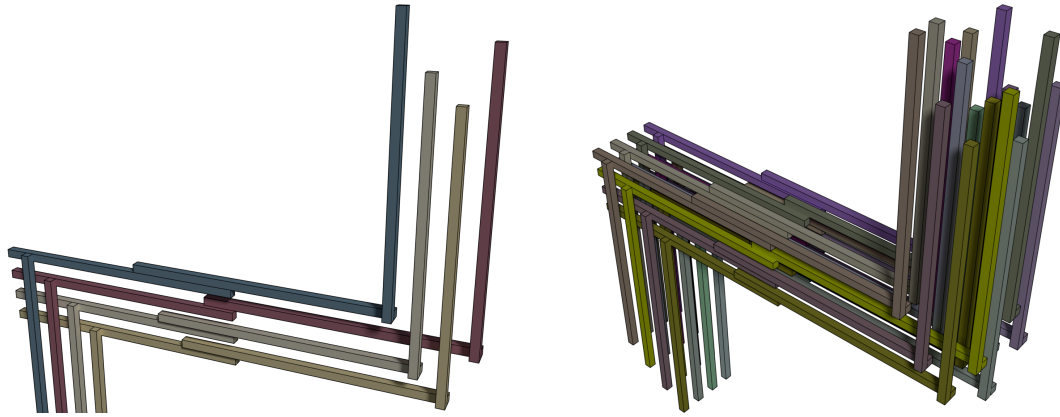
Due to space constraints, we concentrate on giving a picture of the overall construction for $d = 3$, and on presenting our most intricate gadget that is capable of realizing so-called parallel matchings. The rest of the gadgetry and other details are deferred to Appendix A.

Modules and bricks

We index the vertices in a cell by a pair from $[L/8]^2$. The starting object in our reduction is a set of $(L/8)^2$ disjoint boxes parallel to the same axis, arranged loosely in an $L/8 \times L/8$ grid structure called a *brick*. See Figure 2 for an example. Loosely speaking, each box of each brick within the cell's module can be associated with a vertex of the cell; for a brick B , we can refer to a box corresponding to vertex (i, j) of the cell as $B(i, j)$.

Let X be the set of cells within \mathcal{C} : $X \stackrel{\text{def}}{=} \{\{x\} \times [L/8]^2 \mid x \in [cn]^d\}$. The wiring within each cell $x \in X$ will be represented by $O(1)$ bricks, and these bricks will fit in an $O(L)$ side length module.

The position of a brick can be specified by defining its axis (along which the side length of the boxes is L), and for each box (i, j) within the brick, defining the coordinates of its lexicographically smallest corner (or *lexmin corner* for short). For example, consider the brick B with axis x_3 where box $B(i, j)$ has coordinates $(3i, 3j, 0)$. (See Figure 2.) This brick and all bricks isometric to this are called *basic bricks*. Most bricks can be thought of as a perturbation of a basic brick, where we apply shifts to each box. The eventual module that we create will consist of several bricks, which together will represent an even subdivision of the sparse graph G restricted to a given cell. Note that no single brick can be said to represent the set of vertices in a cell. When defining our gadgetry, it is convenient to talk about these bricks, even though in the final construction we only need a certain subset of the boxes within each brick. We can remove the unwanted boxes from each brick at a later stage.



■ **Figure 3** Left: First column of a parallel matching gadget for the permutation $\pi_1(1) = 1, \pi_1(2) = 4, \pi_1(3) = 2, \pi_1(4) = 3$. Boxes of each color induce paths; boxes of different color are disjoint. Right: A full parallel matching gadget.

The parallel matching gadget

A parallel matching gadget is capable of realizing a matching between two cells where the endpoints of each matching edge differ only on a fixed coordinate, so for $d = 3$, all edges are of the type $((x, (i, j)), (x', (i', j)))$ or all edges are of the type $((x, (i, j)), (x', (i, j')))$ for some cells x and x' . We call a matching with this property a *parallel matching*. Parallel matchings can be decomposed into matchings on disjoint cliques, where each clique contains vertices that share all of their coordinates except one. In the remainder of this gadget's description, we will omit the cells x and x' from the coordinate lists.

Suppose that each matching edge is of the form $((i, j), (i', j))$. Let $\pi_j(i)$ denote the first coordinate of the pair of (i, j) , that is, suppose that the matching edges are $((i, j), (\pi_j(i), j))$, $i \in I_j$ for some sets $I_j \subseteq [L/8]$. Instead of realizing these matchings, we first extend them to permutations π_j on each clique $[L/8] \times \{j\}$. A permutation can be thought of as a perfect matching between two copies of a set; by removing the unwanted vertices (removing the unwanted boxes) we can get to a representation of the matching, i.e., a set of vertex disjoint paths that connect box (i, j) in the starting brick to box $(\pi_j(i), j)$ in the target brick.

In every brick, each box is translated individually, where the translation vector's component along the brick's axis must be an integer $k \in 3 \cdot \{-L/8, \dots, L/8\}$, and along the other axes it must be of the form k/L for some $k \in \{-L/8, \dots, L/8\}$. For a brick B , its box of index (i, j) is denoted by $B(i, j)$, and recall that the position of a box is defined by its lexmin corner and the axis of the brick.

We give the coordinates of each box in each brick of the parallel matching gadget below. Let us take the matching edges where $j = 1$ first. We start with the first column of the brick ($j = 1$), where the coordinates of $B^{(1)}(i, 1)$ are $(3i, 3 + i/L, -3i)$. See the left hand side of Figure 3 that illustrates the idea behind the gadget. The coordinates for $B^{(1)}(i, j)$ are $(3i, 3j + i/L, -3i)$. The first column of the next brick $B^{(2)}$ has axis x_1 and the coordinates of $B^{(2)}(i, 1)$ are $(0, 4 + i/L, L - 1 - 3i)$, that is, these boxes touch the previously defined boxes of $B^{(1)}$ from "behind" in Figure 3. In general, $B^{(2)}(i, j)$ has coordinates $(0, 3j + 1 + i/L, L - 3i)$. The next brick $B^{(3)}$ also has axis x_1 , and the coordinates for $B^{(3)}(i, j)$ are $(L/2 + 3\pi_j(i), 3j + 1 + \pi_j(i)/L, L - 3i)$, that is, we change the box perturbations along the first and second coordinate. Finally, the last brick $B^{(4)}$ has axis x_3 and the coordinates are $(3L/2 + 3\pi_j(i), 3j - \pi_j(i)/L, L - 3i)$, i.e., they are placed "in front of"

36:12 How Does Object Fatness Impact Packing?

the bricks of $B^{(3)}$ in Figure 3. This can be rewritten as $B^{(4)}(i', j)$ having coordinates $(3L/2 + 3i', 3j - i'/L, L - 3\pi_j^{-1}(i'))$. Notice that in the final brick, we indeed have the desired ordering, i.e., the ordering of the boxes along the x_1 axis is as required. It is routine to check that the intersection graph induced each column of this parallel matching gadget consists of vertex disjoint paths of length four. Different columns are also disjoint since projecting the boxes of column j onto the x_2 axis results in a subset of the open interval $(3j - 0.5, 3j + 2.5)$.

Using several parallel matching gadgets, by Lemma 11 we are capable of representing arbitrary matchings between two neighboring cells or within a single cell in $O(L) \times O(L) \times O(L)$ space. Further detailed gadgetry describing how branching gadgets are made (capable of representing a collection of degree 3 vertices), and how everything can be fit into modules of side length $O(L)$ are described in Appendix A. Using Theorem 16, it is easy to prove Theorem 2.

Proof of Theorem 2. Set $L \stackrel{\text{def}}{=} \max(16, \alpha^{\frac{d}{d-1}})$. This choice of L implies that any family of canonical boxes of size $1 \times 1 \times L$ are $O(\alpha)$ -stabbed. Furthermore, set $t = (L/8)^{d-1}$. The proof is by reduction from INDEPENDENT SET on subgraphs of the blown-up cube $\mathcal{C} \stackrel{\text{def}}{=} \mathcal{BEC}^d((\bar{n}/t)^{1/d}, t)$, where the subgraph G has maximum degree three, and the neighbors of each vertex in G lie in distinct cells. By Theorem 15, there is no $\gamma > 0$ for which a $2^{\gamma n^{1-1/d} t^{1/d}}$ algorithm exists for this problem under ETH.

Let G be a subgraph of \mathcal{C} as described above. By Theorem 16, we can realize an odd subdivision G' of G using boxes of size $1 \times \dots \times 1 \times L$, with $O(\bar{n})$ vertices in $\text{poly}(\bar{n})$ time. If for any $\gamma > 0$ there is an algorithm for INDEPENDENT SET on α -stabbed canonical boxes with running time $2^{\gamma n^{1-1/d} \alpha}$, then this translates into $2^{\gamma n^{1-1/d} L^{1-1/d}}$ algorithms for all $\gamma > 0$. This can be composed with our construction to get $2^{\gamma \bar{n}^{(1-1/d)} t^{1/d}}$ algorithms for all $\gamma > 0$ for INDEPENDENT SET on the described subgraphs of \mathcal{C} , which contradicts ETH according to Theorem 15. \blacktriangleleft

6 Conclusion

We have explored the impact of the stabbing number on the complexity of packing. We have seen that subexponential packing algorithms are possible for similarly sized objects if the stabbing number is $o(n^{1/d})$. The subexponential algorithms could be derived from powerful separator theorems, while the lower bounds required custom wiring results and non-trivial geometric gadgetry. We propose two open problems for future research.

- What is the precise impact of the stabbing number on the complexity of packing if objects are not similarly sized? One can get a subexponential algorithm by an adaptation of the separator in [9], but it yields an algorithm whose dependence on α is much weaker: it has α^d in the exponent instead of α . Is this algorithm optimal?
- Is there a subexponential algorithm for the DOMINATING SET problem in intersection graphs of α -stabbed similarly sized objects? Or even for n axis-parallel $1 \times n^\varepsilon$ and $n^\varepsilon \times 1$ boxes in two dimensions?

References

- 1 Miklós Abért. Symmetric groups as products of abelian subgroups. *Bulletin of the London Mathematical Society*, 34(4):451–456, 2002.
- 2 Yohji Akama and Kei Irie. VC dimension of ellipsoids. *CoRR*, abs/1109.4347, 2011. [arXiv:1109.4347](https://arxiv.org/abs/1109.4347).

- 3 Jochen Alber and Jirí Fiala. Geometric separation and exact solutions for the parameterized independent set problem on disk graphs. *Journal of Algorithms*, 52(2):134–151, 2004. doi:10.1016/j.jalgor.2003.10.001.
- 4 Julien Baste and Dimitrios M. Thilikos. Contraction-Bidimensionality of Geometric Intersection Graphs. In *IPEC 2017*, volume 89 of *LIPICs*, pages 5:1–5:13, 2018. doi:10.4230/LIPICs.IPEC.2017.5.
- 5 Csaba Biró, Édouard Bonnet, Dániel Marx, Tillmann Miltzow, and Paweł Rzażewski. Fine-grained complexity of coloring unit disks and balls. *JoCG*, 9(2):47–80, 2018. doi:10.20382/jocg.v9i2a4.
- 6 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.
- 7 Miroslav Chlebík and Janka Chlebíková. Approximation hardness of optimization problems in intersection graphs of d -dimensional boxes. In *Proceedings of SODA 2005*, pages 267–276. SIAM, 2005. URL: <http://dl.acm.org/citation.cfm?id=1070432.1070470>.
- 8 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 9 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 STOC 2018*, pages 574–586, 2018. doi:10.1145/3188745.3188854.
- 10 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. *CoRR*, abs/1803.10633, 2018. arXiv:1803.10633.
- 11 Erik D. Demaine, Fedor V. Fomin, Mohammad Taghi Hajiaghayi, and Dimitrios M. Thilikos. Fixed-parameter algorithms for (k, r) -Center in planar graphs and map graphs. *ACM Transactions on Algorithms*, 1(1):33–47, 2005.
- 12 Erik D. Demaine, Fedor V. Fomin, Mohammad Taghi Hajiaghayi, and Dimitrios M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and H -minor-free graphs. *Journal of the ACM*, 52(6):866–893, 2005. doi:10.1145/1101821.1101823.
- 13 Frederic Dorn, Fedor V. Fomin, and Dimitrios M. Thilikos. Subexponential parameterized algorithms. *Computer Science Review*, 2(1):29–39, 2008.
- 14 Frederic Dorn, Fedor V. Fomin, and Dimitrios M. Thilikos. Catalan structures and dynamic programming in H -minor-free graphs. *J. Comput. Syst. Sci.*, 78(5):1606–1622, 2012.
- 15 Frederic Dorn, Eelko Penninkx, Hans L. Bodlaender, and Fedor V. Fomin. Efficient Exact Algorithms on Planar Graphs: Exploiting Sphere Cut Decompositions. *Algorithmica*, 58(3):790–810, 2010.
- 16 Fedor V. Fomin, Daniel Lokshtanov, Venkatesh Raman, and Saket Saurabh. Subexponential algorithms for partial cover problems. *Inf. Process. Lett.*, 111(16):814–818, 2011.
- 17 Fedor V. Fomin, Daniel Lokshtanov, and Saket Saurabh. Bidimensionality and geometric graphs. In *Proceedings of SODA 2012*, pages 1563–1575. SIAM, 2012. URL: <http://portal.acm.org/citation.cfm?id=2095240&CFID=63838676&CFTOKEN=79617016>.
- 18 Fedor V. Fomin and Dimitrios M. Thilikos. Dominating Sets in Planar Graphs: Branch-Width and Exponential Speed-Up. *SIAM J. Comput.*, 36(2):281–309, 2006.
- 19 Sarel 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.
- 20 David Haussler and Emo Welzl. ε -nets and simplex range queries. *Discrete & Computational Geometry*, 2(2):127–151, June 1987. doi:10.1007/BF02187876.
- 21 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.
- 22 Fritz John. *Extremum Problems with Inequalities as Subsidiary Conditions*, pages 197–215. Springer Basel, Basel, 2014. doi:10.1007/978-3-0348-0439-4_9.

- 23 Sándor Kisfaludi-Bak, Dániel Marx, and Tom C. van der Zanden. How does object fatness impact the complexity of packing in d dimensions? *CoRR*, abs/1909.12044, September 2019. [arXiv:1909.12044](https://arxiv.org/abs/1909.12044).
- 24 Philip N. Klein and Dániel Marx. A subexponential parameterized algorithm for Subset TSP on planar graphs. In *SODA 2014 Proc.*, pages 1812–1830, 2014.
- 25 Dániel Marx and Michal Pilipczuk. Optimal Parameterized Algorithms for Planar Facility Location Problems Using Voronoi Diagrams. In *Proceedings of ESA 2015*, volume 9294 of *LNCS*, pages 865–877. Springer, 2015. doi:10.1007/978-3-662-48350-3_72.
- 26 Dániel Marx and Anastasios Sidiropoulos. The limited blessing of low dimensionality: when $1 - 1/d$ is the best possible exponent for d -dimensional geometric problems. In *Proceedings of SoCG 2014*, pages 67–76. ACM, 2014. doi:10.1145/2582112.2582124.
- 27 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.
- 28 Marcin Pilipczuk, Michał Pilipczuk, Piotr Sankowski, and Erik Jan van Leeuwen. Subexponential-Time Parameterized Algorithm for Steiner Tree on Planar Graphs. In *STACS 2013 Proc.*, pages 353–364, 2013.
- 29 Warren D. Smith and Nicholas C. Wormald. Geometric Separator Theorems & Applications. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science, FOCS 1998*, pages 232–243. IEEE Computer Society, 1998. doi:10.1109/SFCS.1998.743449.
- 30 Dimitrios M. Thilikos. Fast Sub-exponential Algorithms and Compactness in Planar Graphs. In *ESA 2011 Proc.*, pages 358–369, 2011.
- 31 A. Frank van der Stappen, Dan Halperin, and Mark H. Overmars. The Complexity of the Free Space for a Robot Moving Amidst Fat Obstacles. *Comput. Geom.*, 3:353–373, 1993. doi:10.1016/0925-7721(93)90007-S.

A Gadgetry and further construction details for the proof of Theorem 16

Parity Fix, adjustment, bridge, and elbow gadgets

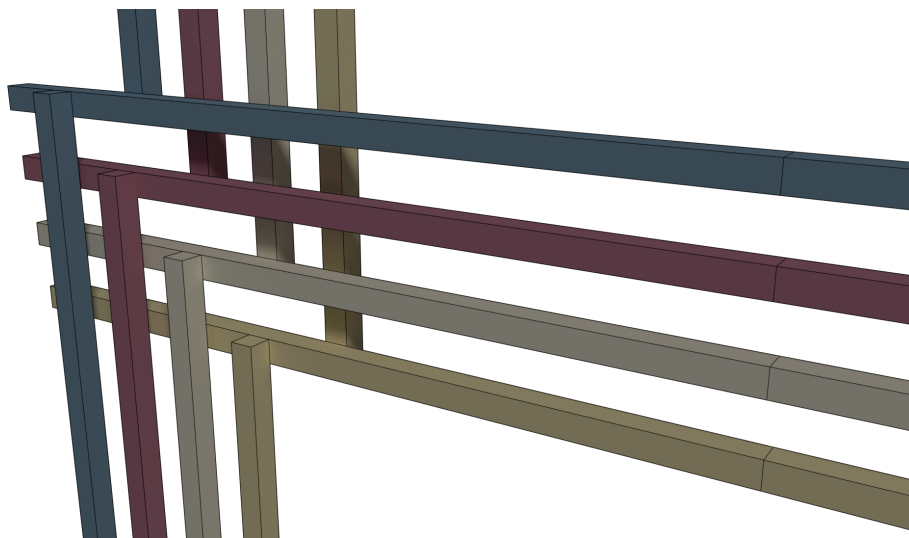
The parity fix gadget is introduced so that we can ensure that each of the subdivisions that we create are even subdivisions. The gadget induces a path of length 3 or 4 depending on our needs, but occupies the same space in both cases. More precisely, the parity fix gadget contains three or four boxes, depending on the parity we need. The union of the boxes is a larger box of size $3L \times 1 \times 1$; it is easy to see that within that space we can realize both a path of length three and four using $L \times 1 \times 1$ boxes: one can cover the larger box by placing their lexmin corners at equal length intervals.

We can bridge distance along the axis of a basic brick by putting basic bricks next to each other, where each box intersects only the box of the same index from the previous and following brick. This creates a set of $(L/8)^2$ vertex disjoint paths in the intersection graph. We call this a *bridge gadget*.

Using two bricks of the same axis, we can in one step get rid of a perturbation (or introduce one). Let B be a normal brick with axis x_3 that is a perturbation of the basic brick. We introduce the basic brick B' that is the translate of the basic brick with the vector $(0, 1, L/2)$. Notice that box $B(i, j)$ intersects $B'(i, j)$ and no other boxes. Moreover, we could even introduce arbitrary perturbations along the x_1 axis in B' and along the x_3 axis within both B and B' without changing the intersection graph induced by B and B' . We call a pair of normal bricks that are a translated and rotated version of these an *adjustment gadget*.



■ **Figure 4** An elbow.



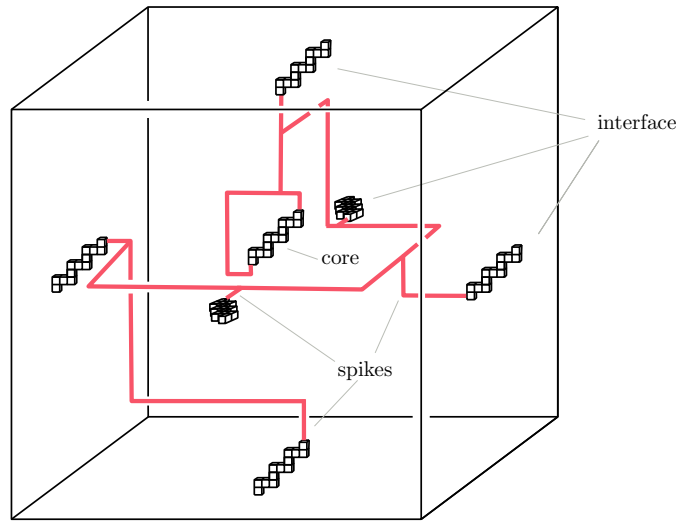
■ **Figure 5** The first “column” of a branching gadget.

Next, we introduce a way to change brick axis using an “elbow”. Consider a brick B that is a perturbation of the basic brick, where box (i, j) has coordinates $(3i, 3j, -3i)$. The brick B' has axis x_1 and the coordinates for $B'(i, j)$ are $(3i, 3j, L - 3i)$ (see Figure 4). Notice that using these *elbow gadgets* and adjustment gadgets together, one can route from any brick to any other brick at distance $\Theta(L)$ in $O(1)$ steps.

Realizing an arbitrary matching of a biclique or clique

We can regard a general matching M induced by two neighboring cells as a permutation of $[L/8]^2$, which can be written as the product of three special permutations by Corollary 12 that correspond to parallel matchings; i.e., the matching M is realizable as the succession of three parallel matchings. This means that each edge of M becomes a path of length three, so by using three parallel matching gadgets in succession we can represent M . We add a parity fix gadget to each box at the beginning of each wire, which will be useful later to ensure that each edge has been subdivided an even number of times. As a result, we have realized M using $O(1)$ bricks and $O(L) \times O(L) \times O(L)$ space. This collection of boxes is called a *general matching gadget*. A general matching gadget has a first and a last brick where it connects to the rest of the construction, we call these bricks endbricks.

36:16 How Does Object Fatness Impact Packing?



■ **Figure 6** A module with general matching gadgets of the interface and the core, with the simplified image of a brick-tree (in red).

If the goal is to realize a matching within a cell with vertex set V_x , then we can just create two copies of V_x (denoted by V'_x and V''_x), with a complete bipartite graph between them. For a matching edge $v_i v_j \in \binom{V_x}{2}$, we identify it with the edge $v'_i v''_j$. Then we realize the matching of this biclique using a general matching gadget.

The branching gadget

The *branching gadget* creates for all indices in $[L/8]^2$ a disjoint copy of a star on 4 vertices (that is, a vertex of degree 3 with its neighborhood of 3 isolated vertices). This gadget contains four bricks, and realizes $(L/8)^2$ disjoint stars. We use the first two bricks ($B^{(1)}$ and $B^{(2)}$) of the parallel matching gadget. The third brick B' is a translate of the first brick $B^{(1)}$ with the vector $(3, 2, L-1)$, i.e., the coordinates of $B'(i, j)$ are $(3i+3, 3j+2+i/L, L-1-3i)$. The final brick B'' is the translate of $B^{(2)}$ by the vector $(L, 0, 0)$. See Figure 5 for a rendering of the first “column” of the four bricks. Vertices corresponding to $B^{(2)}$ have degree three, and their neighbors are the boxes of the same index in $B^{(1)}$, B' and B'' .

Constructing a module

Our goal is to define modules of side length $O(L)$ that are capable of representing the role played by cells. The modules together must be able to represent a subgraph of \mathcal{C} of maximum degree three, where the neighbors of any vertex lie in distinct cells.

For all pairs of neighboring modules, we introduce a general matching gadget to represent the matching required by G between the two neighboring cells. These gadgets form the *interface*. Moreover, in the middle of each module, we add another general matching gadget to represent the matching within the cell; this gadget is the *core* of the module. See Figure 6. Finally, within each module, we tie the endbricks of the core and the endbricks of the interface falling inside the module together with a *brick-tree*. The brick-tree is a collection of $(L/8)^2$ isomorphic and disjoint trees, realized as a collection of branching, elbow, adjustment and bridge gadgets. Each tree (i, j) has maximum degree three, and its leaves are the boxes of index (i, j) in the interface and in the core.

First, we show that such a construction is sufficient to represent an even subdivision of an arbitrary subgraph G , and later we show how the brick-tree can be constructed. Let G be a subgraph with the desired properties, and let x be a particular cell. For each edge uv induced by x , we fix an arbitrary orientation, and realize the acquired matching so that the source vertex of the arcs are in one end of the core and the targets are in the other. Since the neighbors of any vertex lie in different cells, all indices of $[L/8]^2$ appear at most once, either as a source of an arc, as a target of an arc, or not at all. Then we realize the arcs using the core's general matching gadget of the module. For each index $\mathbf{i} \in [L/8]^2$, the edges incident to vertex \mathbf{i} of x can be assigned to a subtree T of the tree corresponding to index \mathbf{i} , where T has at most three leaves, at most one of which is adjacent to a box of the core, and other leaves are adjacent to boxes in distinct endbricks of the interface. There is a unique minimal subtree T that induces the desired (at most three) leaves; we can map a vertex $v \in V(G)$ of degree three to the degree three vertex of T . If V has a smaller degree, then it can be mapped to an arbitrary non-leaf vertex of T .

To construct a brick-tree in \mathbb{R}^3 , consider first a Euclidean grid cube of size $O(1)$. We can use this small cube as a model of our module: in general, an edge of this cube represents a brick. We have some edges already occupied by the general matching gadgets corresponding to the interface and the core. By choosing a cube large enough, we can ensure that these vertices are distant in the ℓ_1 norm. It is easy to see that if the cube is large enough (we allow its size to depend only on d), then there is a subtree of the grid of maximum degree three, where the leaves are some distant prescribed vertices. Such a tree can be constructed for example by mimicking a Hamiltonian path of the inscribed octahedron of the module, and adding to it small "spikes" that go to the endbrick of the interfaces. At the end of the path, we extend it towards the center of the cube, where we add another branching for the two endbricks of the core. The branching points in the brick-tree are branching gadgets, the turns are elbow gadgets, and straight segments are bridges and adjustments.

Finalizing the construction in \mathbb{R}^3

By packing the modules in a side length $O(sL)$ Euclidean cube, and removing unused boxes from each module according to the given subgraph, we get our final construction for three dimensions. For each edge, we have it represented by a sequence of $O(1)$ boxes passing through a single general matching gadget. Using the parity fix gadget inside the general matching gadget, we can ensure that the path representing the edge has an odd number of internal vertices. Therefore, the final construction has $O(|V(G)|)$ boxes, and each edge of G is represented with a path of odd length, that is, the graph induced by the boxes is an even subdivision of G .

The construction in higher dimensions

It is surprisingly easy to adapt our three-dimensional construction to the d -dimensional case. This time, we need to realize a subgraph of $\mathcal{C} = \mathcal{BEC}^d(s, (L/8)^{d-1})$.

The basic brick in d dimensions contains $(L/8)^{d-1}$ boxes, indexed by $[L/8]^{d-1}$, where the lexicographically minimal corner of box \mathbf{i} is $(0, 3\mathbf{i})$. For normal bricks, we allow perturbations of the form $3k$ ($|k| \in [L/8]$) along the axis of the brick, and k/L ($|k| \in [L/8]$) in all other directions. The parity fix, adjustment, and elbow gadgets can be defined analogously. The parallel matching gadget is also straightforward: the task here is to represent a parallel matching, where each edge is of the form $(\mathbf{i}, \mathbf{i}') \in [L/8]^{d-1} \times [L/8]^{d-1}$, where \mathbf{i}, \mathbf{i}' differ only on

36:18 How Does Object Fatness Impact Packing?

the t -th coordinate for some fixed $t \in [d-1]$. As previously, we can extend this to $(L/8)^{d-2}$ permutations, where for each $\iota \in [L/8]^{d-2}$, we have a permutation π_ι over the ‘‘column’’ ι , i.e., over the set

$$\{(i_1, \dots, i_{d-1}) \mid i_t \in [L/8] \text{ and } (i_1, \dots, i_{t-1}, i_{t+1}, \dots, i_{d-1}) = \iota\}.$$

Such a permutation can be represented as described before: we replace the role played by the x_1 axis with x_t , the role of x_2 with $x_{t+1 \bmod (d-1)}$ and x_3 with x_d . Along all other axes, we introduce no perturbations to the boxes. The column gadget corresponding to column $\iota = (i_1, \dots, i_{t-1}, i_{t+1}, \dots, i_{d-1})$ can be covered by¹

$$\begin{aligned} & [3i_1, 3i_1 + 1] \times \dots \times [3i_{t-1}, 3i_{t-1} + 1] \\ & \quad \times [-L/2, 3L/2] \times (3i_{t+1} - 0.5, 3i_{t+1} + 2.5) \\ & \quad \times [3i_{t+2}, 3i_{t+2} + 1] \times \dots \times [3i_{d-1}, 3i_{d-1} + 1] \times [0, \frac{3}{2}L]. \end{aligned}$$

These sets are clearly disjoint for distinct values of ι .

A general matching M is regarded as a permutation of $[L/8]^{d-1}$, which can be written as the product of $2(d-1) - 1$ special permutations by Corollary 12 that correspond to parallel matchings; therefore, M is realizable as the succession of $2d - 3$ parallel matchings. As a result, we can realize M with $O(d) = O(1)$ bricks and $O(L) \times \dots \times O(L)$ space. As before, we add parity fix gadgets to each box of one of the endbricks.

To realize a brick-tree, we can again trace a Hamiltonian path of the graph given by the dimension 1 faces of the cross-polytope inside the module, and add spikes to it to reach the endbricks of the interface and extend it to the two endbricks of the core. Note that the cross-polytope does have a Hamiltonian path, we can use e.g.

$$(1, 0, \dots, 0); (0, 1, 0, \dots, 0) \dots (0, \dots, 0, 1); (-1, 0, \dots, 0); (0, -1, 0, \dots, 0) \dots (0, \dots, 0, -1).$$

The finalizing steps are again analogous to the 3-dimensional case. This concludes the proof of Theorem 16.

¹ The formula is only accurate for the case $t \leq d-2$. If $t = d-1$, the role of x_{t+1} and x_1 should be switched.

On One-Round Discrete Voronoi Games

Mark de Berg

Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands
m.t.d.berg@tue.nl

Sándor Kisfaludi-Bak

Max Planck Institut für Informatik, Saarbrücken, Germany
sandor.kisfaludi-bak@mpi-inf.mpg.de

Mehran Mehr

Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands
mehran.mehr@gamil.com

Abstract

Let V be a multiset of n points in \mathbb{R}^d , which we call voters, and let $k \geq 1$ and $\ell \geq 1$ be two given constants. We consider the following game, where two players \mathcal{P} and \mathcal{Q} compete over the voters in V : First, player \mathcal{P} selects a set P of k points in \mathbb{R}^d , and then player \mathcal{Q} selects a set Q of ℓ points in \mathbb{R}^d . Player \mathcal{P} wins a voter $v \in V$ iff $\text{dist}(v, P) \leq \text{dist}(v, Q)$, where $\text{dist}(v, P) := \min_{p \in P} \text{dist}(v, p)$ and $\text{dist}(v, Q)$ is defined similarly. Player \mathcal{P} wins the game if he wins at least half the voters. The algorithmic problem we study is the following: given V , k , and ℓ , how efficiently can we decide if player \mathcal{P} has a winning strategy, that is, if \mathcal{P} can select his k points such that he wins the game no matter where \mathcal{Q} places her points.

Banik et al. devised a singly-exponential algorithm for the game in \mathbb{R}^1 , for the case $k = \ell$. We improve their result by presenting the first polynomial-time algorithm for the game in \mathbb{R}^1 . Our algorithm can handle arbitrary values of k and ℓ . We also show that if $d \geq 2$, deciding if player \mathcal{P} has a winning strategy is Σ_2^P -hard when k and ℓ are part of the input. Finally, we prove that for any dimension d , the problem is contained in the complexity class $\exists\forall\mathbb{R}$, and we give an algorithm that works in polynomial time for fixed k and ℓ .

2012 ACM Subject Classification Theory of computation \rightarrow Computational geometry; Theory of computation \rightarrow Algorithmic game theory; Theory of computation \rightarrow Problems, reductions and completeness

Keywords and phrases competitive facility location, plurality point

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.37

Related Version A full version of the paper is available at <https://arxiv.org/abs/1902.09234>.

Funding MdB, SKB, and MM are supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 024.002.003, 024.002.003, and 022.005.025, respectively.

1 Introduction

Voronoi games, as introduced by Ahn et al. [1], can be viewed as competitive facility-location problems in which two players \mathcal{P} and \mathcal{Q} want to place their facilities in order to maximize their market area. The Voronoi game of Ahn et al. is played in a bounded region $R \subset \mathbb{R}^2$, and the facilities of the players are modeled as points in this region. Each player gets the same number, k , of facilities, which they have to place alternatingly. The market area of \mathcal{P} (and similarly of \mathcal{Q}) is now given by the area of the region of all points $q \in R$ whose closest facility was placed by \mathcal{P} , that is, it is the total area of the Voronoi cells of \mathcal{P} 's facilities in the Voronoi diagram of the facilities of \mathcal{P} and \mathcal{Q} . Ahn et al. proved that for $k > 1$ and when the region R is a circle or a segment, the second player can win the game by a payoff of $1/2 + \varepsilon$, for some $\varepsilon > 0$, where the first player can ensure ε is arbitrarily small.



© Mark de Berg, Sándor Kisfaludi-Bak, and Mehran Mehr;
licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 37; pp. 37:1–37:17

Leibniz International Proceedings in Informatics



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

The one-round Voronoi game introduced by Cheong et al. [8] is similar to the Voronoi game of Ahn et al., except that the first player must first place all his k facilities, after which the second player places all her k facilities. They considered the problem where R is a square, and they showed that when k is large enough the second player can always win a fraction $1/2 + \alpha$ of the area of R for some $\alpha > 0$. Fekete and Meijer [11] considered the problem on a rectangle R of aspect ratio $\rho \leq 1$. They showed that the first player wins more than half the area of R , unless $k \geq 3$ and $\rho > \sqrt{2}/n$, or $k = 2$ and $\rho > \sqrt{3}/2$. They also showed that if R is a polygon with holes, then computing the locations of the facilities for the second player that maximize the area she wins, against a given set of facilities of the first player is NP-hard.

One-round discrete Voronoi games. In this paper we are interested in *discrete (Euclidean) one-round Voronoi games*, where the players do not compete for area but for a discrete set of points. That is, instead of the region R one is given a set V of n points in a geometric space, and a point $v \in V$ is won by the player owning the facility closest to v . (Another discrete variant of Voronoi games is played on graphs [16, 18] but we restrict our attention to the geometric variant.) Formally, the problem we study is defined as follows.

Let V be a multiset of n points in \mathbb{R}^d , which we call *voters* from now on, and let $k \geq 1$ and $\ell \geq 1$ be two integers. The one-round discrete Voronoi game defined by the triple $\langle V, k, \ell \rangle$ is a single-turn game played between two players \mathcal{P} and \mathcal{Q} . First, player \mathcal{P} places a set P of k points in \mathbb{R}^d , then player \mathcal{Q} places a set Q of ℓ points in \mathbb{R}^d . (These points may coincide with the voters in V .) We call the set P the *strategy of \mathcal{P}* and the set Q the *strategy of \mathcal{Q}* . Player \mathcal{P} wins a voter $v \in V$ if $\text{dist}(v, P) \leq \text{dist}(v, Q)$, where $\text{dist}(v, P)$ and $\text{dist}(v, Q)$ denote the minimum distance between a voter v and the sets P and Q , respectively. Note that this definition favors player \mathcal{P} , since in case of a tie a voter is won by \mathcal{P} . We now define $V[P \succeq Q] := \{v \in V : \text{dist}(v, P) \leq \text{dist}(v, Q)\}$ to be the multiset of voters won by player \mathcal{P} when he uses strategy P and player \mathcal{Q} uses strategy Q . Player \mathcal{P} wins the game $\langle V, k, \ell \rangle$ if he wins at least half the voters in V , that is, when $|V[P \succeq Q]| \geq n/2$; otherwise \mathcal{Q} wins the game. Here $|V[P \succeq Q]|$ denotes the size of the multiset $V[P \succeq Q]$ (counting multiplicities). We now define $\Gamma_{k,\ell}(V)$ as the maximum number of voters that can be won by player \mathcal{P} against an optimal opponent:

$$\Gamma_{k,\ell}(V) := \max_{P \subset \mathbb{R}^d, |P|=k} \min_{Q \subset \mathbb{R}^d, |Q|=\ell} |V[P \succeq Q]|.$$

For a given multiset V of voters, we want to decide if¹ $\Gamma_{k,\ell}(V) \geq n/2$. In other words, we are interested in determining for a given game $\langle V, k, \ell \rangle$ if \mathcal{P} has a *winning strategy*, which is a set of k points such that \mathcal{P} wins the game no matter where \mathcal{Q} places her points.

An important special case, which has already been studied in spatial voting theory for a long time, is when $k = \ell = 1$ [14]. Here the coordinates of a point in V represent the preference of the voter on certain topics, and the point played by \mathcal{Q} represents a certain proposal. If the point played by \mathcal{P} wins against all possible points played by \mathcal{Q} , then the \mathcal{P} 's proposal will win the vote against any other proposal. Note that in the problem definition we gave above, voters at equal distance from P and Q are won by \mathcal{P} , and \mathcal{P} has to win at least half the voters. This is the definition typically used in papers of Voronoi games [3, 4, 5, 6]. In voting theory other variants are studied as well, for instance where points at equal distance

¹ One can also require that $\Gamma_{k,\ell}(V) > n/2$; with some small modifications, all the results in this paper can be applied to the case with strict inequality as well.

to \mathcal{P} and \mathcal{Q} are not won by either of them, and \mathcal{P} wins the game if he wins more voters than \mathcal{Q} ; see the paper by McKelvey and Wendell [14] who use the term *majority points* for the former variant and the term *plurality points* for the latter variant.

Previous work. Besides algorithmic problems concerning the one-round discrete Voronoi game one can also consider combinatorial problems. In particular, one can ask for bounds on $\Gamma_{k,\ell}(V)$ as a function of n , k , and ℓ . It is known that for any set V in \mathbb{R}^2 and $k = \ell = 1$ we have $\lfloor n/3 \rfloor \leq \Gamma_{1,1}(V) \leq \lceil n/2 \rceil$. This result is based on known bounds for maximum Tukey depth, where the lower bound can be proven using Helly's theorem. It is also known [6] that there is a constant c such that $k = c\ell$ points suffice for \mathcal{P} to win, that is, $\Gamma_{c\ell,\ell}(V) \geq n/2$ for any V .

In this paper we focus on the algorithmic problem of computing $\Gamma_{k,\ell}(V)$ for given V , k , and ℓ . The problem of deciding if $\Gamma_{k,\ell}(V) \geq n/2$ was studied for the case $k = \ell = 1$ by Wu et al. [19] and Lin et al. [13], and later by De Berg et al. [9] who solve this problem in $O(n \log n)$ time in any fixed dimension d . Their algorithms work when V is a set (not a multiset) and for plurality points instead of majority points. Other algorithmic results are for the setting where the players already placed all but one of their points, and one wants to compute the best locations for the last point of \mathcal{P} and of \mathcal{Q} . Banik et al. [5] gave algorithms that find the best location for \mathcal{P} in $O(n^8)$ time and for \mathcal{Q} in $O(n^2)$ time. For the two-round variant of the problem, with $k = \ell = 2$, polynomial algorithms for finding the optimal strategies of both players are also known [4].

Our work is inspired by the paper of Banik et al. [3] on computing $\Gamma_{k,\ell}(V)$ in \mathbb{R}^1 . They considered the case of arbitrarily large k and ℓ , but where $k = \ell$ (and V is a set instead of a multiset). For this case they showed that depending on the set V either \mathcal{P} or \mathcal{Q} can win the game, and they presented an algorithm to compute $\Gamma_{k,\ell}(V)$ in time $O(n^{k-\lambda_k})$, where $0 < \lambda_k < 1$ is a constant dependent only on k . This raises the question: is the problem NP-hard when k is part of the input?

Our results. We answer the question above negatively, by presenting an algorithm that computes $\Gamma_{k,\ell}(V)$ in \mathbb{R}^1 in polynomial time. Our algorithm works when V is a multiset, and it does not require k and ℓ to be equal. Our algorithm computes $\Gamma_{k,\ell}(V)$ and finds a strategy for \mathcal{P} that wins this many voters in time $O(kn^4)$. The algorithm can be extended to the case when the voters are weighted, requiring only a slight increase in running time.

The algorithm by Banik et al. [3] discretizes the problem, by defining a finite set of potential locations for \mathcal{P} to place his points. However, to ensure an optimal strategy for \mathcal{P} , the set of potential locations has exponential size. To overcome this problem we need several new ideas. First of all, we partition the possible strategies into various classes – the concept of thresholds introduced later plays this role – such that for each class we can anticipate the behavior of the optimal strategy for \mathcal{Q} . To compute the best strategy within a certain class we use dynamic programming, in a non-standard (and, unfortunately, rather complicated) way. The subproblems in our dynamic program are for smaller point sets and smaller values of k and ℓ (actually we will need several other parameters) where the goal of \mathcal{P} will be to push his rightmost point as far to the right as possible to win a certain number of points. One complication in the dynamic program is that it is unclear which small subproblems I' can be used to solve a given subproblem I . The opposite direction – determining for I' which larger I may use I' in their solution – is easier, so we use a sweep approach: when the solution to some I' is determined, we update the solution to larger subproblems I that can use I' .

After establishing that we can compute $\Gamma_{k,\ell}(V)$ in polynomial time in \mathbb{R}^1 , we turn to the higher-dimensional problem. We show (in the full version) that deciding if \mathcal{P} has a winning strategy is Σ_2^P -hard in \mathbb{R}^2 . We also show that for fixed k and ℓ this problem can be solved in polynomial time. Our solution combines algebraic methods [7] with a result of Paterson and Zwick [15] that one can construct a polynomial-size boolean circuits that implements the majority function. The latter result is essential to avoid the appearance of n in the exponent. As a byproduct of the algebraic method, we show that the problem is contained in the complexity class $\exists\forall\mathbb{R}$; see [10] for more information on this complexity class.

2 A Polynomial-Time Algorithm for $d = 1$

In this section, we present a polynomial-time algorithm for the 1-dimensional discrete Voronoi game. Our algorithm will employ dynamic programming, and it will be convenient to use n , k , and ℓ as variables in the dynamic program. From now on, we therefore use n^* for the size of the original multiset V , and k^* and ℓ^* for the initial number of points that can be played by \mathcal{P} and \mathcal{Q} , respectively.

2.1 Notation and Basic Properties

We denote the given multiset of voters by $V := \{v_1, \dots, v_{n^*}\}$, where we assume the voters are numbered from left to right. We also always number the points in the strategies $P := \{p_1, \dots, p_{k^*}\}$ and $Q := \{q_1, \dots, q_{\ell^*}\}$ from left to right. For brevity we make no distinction between a point and its value (that is, its x -coordinate), so that we can for example write $p_1 < q_1$ to indicate that the leftmost point of P is located to the left of the leftmost point of Q .

For a given game $\langle V, k, \ell \rangle$, we say that a strategy P of player \mathcal{P} *realizes* a gain γ if $|V[P \succeq Q]| \geq \gamma$ for any strategy Q of player \mathcal{Q} . Furthermore, we say that a strategy P is *optimal* if it realizes $\Gamma_{k,\ell}(V)$, the maximum possible gain for \mathcal{P} , and we say a strategy Q is *optimal* against a given strategy P if $|V[P \succeq Q]| \leq |V[P \succeq Q']|$ for any strategy Q' .

Trivial, reasonable, and canonical strategies for \mathcal{P} . For $0 \leq n \leq n^*$, define $V_n := \{v_1, \dots, v_n\}$ to be the leftmost n points in V . Suppose we want to compute $\Gamma_{k,\ell}(V_n)$ for some $1 \leq k \leq n$ and $0 \leq \ell \leq n$. The *trivial strategy* of player \mathcal{P} is to place his points at the k points of V_n with the highest multiplicities – here we consider the multiset V_n as a set of distinct points, each with a multiplicity corresponding to the number of times it occurs in V_n – with ties broken arbitrarily. Let $\|V_n\|$ denote the number of distinct points in V_n . Then the trivial strategy is optimal when $k \geq \|V_n\|$ and also when $\ell \geq 2k$: in the former case \mathcal{P} wins all voters with the trivial strategy, and in the latter case \mathcal{Q} can always win all voters not coinciding with a point in P (namely by surrounding each $p_i \in P$ by two points very close to p_i) so the trivial strategy is optimal for \mathcal{P} . Hence, from now on we consider subproblems with $k < \|V_n\|$ and $\ell < 2k$.

We can without loss of generality restrict our attention to strategies for \mathcal{P} that place at most one point in each half-open interval of the form $(v_i, v_{i+1}]$ with $v_i \neq v_{i+1}$, where $0 \leq i \leq n$, $v_0 := -\infty$, and $v_{n^*+1} := \infty$. Indeed, placing more than two points inside an interval $(v_i, v_{i+1}]$ is clearly not useful, and if two points are placed in some interval $(v_i, v_{i+1}]$ then we can always move the leftmost point onto v_i . (If v_i is already occupied by a point in P , then we can just put the point on any unoccupied voter; under our assumption that $k < \|V_n\|$ an unoccupied voter always exists.) We will call a strategy for \mathcal{P} satisfying the property above *reasonable*.

► **Observation 1** (Banik et al. [3]). *Assuming $k < \|V_n\|$ there exist an optimal strategy for \mathcal{P} that is reasonable and has $p_1 \in V$ (that is, p_1 coincides with a voter).*

We can define an ordering on strategies of the same size by sorting them in lexicographical order. More precisely, we say that a strategy $P = \{p_1, \dots, p_k\}$ is *greater than* a strategy $P' = \{p'_1, \dots, p'_k\}$, denoted by $P \succ P'$, if $\langle p_1, \dots, p_k \rangle >_{\text{lex}} \langle p'_1, \dots, p'_k \rangle$, where $>_{\text{lex}}$ denotes the lexicographical order. Using this ordering, the largest reasonable strategy P that is optimal – namely, that realizes $\Gamma_{k,\ell}(V_n)$ – is called the *canonical strategy* of \mathcal{P} .

α -gains, β -gains, and gain sequences. Consider a strategy $P := \{p_1, \dots, p_k\}$. It will be convenient to add two extra points to P , namely $p_0 := -\infty$ and $p_{k+1} := \infty$; this clearly does not influence the outcome of the game. The strategy P thus induces $k + 1$ open intervals of the form (p_i, p_{i+1}) where player \mathcal{Q} may place her points. It is easy to see that there exists an optimal strategy for \mathcal{Q} with the following property: \mathcal{Q} contains at most two points in each interval (p_i, p_{i+1}) with $1 \leq i \leq k - 1$, and at most one point in (p_0, p_1) and at most one point in (p_k, p_{k+1}) . From now on we restrict our attention to strategies for \mathcal{Q} with this property.

Now suppose that x and y are consecutive points (with $x < y$) in some strategy P , where x could be $-\infty$ and y could be ∞ . As just argued, \mathcal{Q} either places zero, one, or two points inside (x, y) . When \mathcal{Q} places zero points, then she obviously does not win any of the voters in $V_n \cap (x, y)$. The maximum number of voters \mathcal{Q} can win from $V_n \cap (x, y)$ by placing a single point is the maximum number of voters in (x, y) that can be covered by an open interval of length $(y - x)/2$, as by placing her point in any $q \in (x, y)$, \mathcal{Q} wins all (and only) the voters in the open interval $((x + q)/2, (q + y)/2)$ of length $(y - x)/2$; see Banik et al. [3]. We call this value the α -gain of \mathcal{Q} in (x, y) and denote it by $\text{gain}_\alpha(V_n, x, y)$. By placing two points inside (x, y) , one immediately to the right of x and one immediately to the left of y , player \mathcal{Q} will win all voters $V_n \cap (x, y)$. Thus the extra number of voters won by the second point in (x, y) as compared to just placing a single point is equal to $|V_n \cap (x, y)| - \text{gain}_\alpha(V_n, x, y)$. We call this quantity the β -gain of \mathcal{Q} in (x, y) and denote it by $\text{gain}_\beta(V_n, x, y)$. Note that for intervals (x, ∞) we have $\text{gain}_\alpha(x, \infty) = |V_n \cap (x, \infty)|$ and $\text{gain}_\beta(x, \infty) = 0$; a similar statement holds for $(-\infty, y)$.

The following observation follows from the fact that $\text{gain}_\alpha(V_n, x, y)$ equals the maximum number of voters in (x, y) that can be covered by an open interval of length $(y - x)/2$.

► **Observation 2** (Banik et al. [3]). *For any x, y we have $\text{gain}_\alpha(V_n, x, y) \geq \text{gain}_\beta(V_n, x, y)$.*

Let $P := \{p_0, p_1, \dots, p_k, p_{k+1}\}$ be a given strategy for \mathcal{P} , where by convention $p_0 = -\infty$ and $p_{k+1} = \infty$. Consider $\{\text{gain}_\alpha(V_n, p_i, p_{i+1}) : 0 \leq i \leq k\} \cup \{\text{gain}_\beta(V_n, p_i, p_{i+1}) : 0 \leq i \leq k\}$, the multiset of all α -gains and β -gains defined by the intervals (p_i, p_{i+1}) . Sort this sequence in non-increasing order, using the following tie-breaking rules if two gains are equal:

- Gains from the interval (p_i, p_{i+1}) have precedence over gains from intervals (p_j, p_{j+1}) when $i < j$.
- if both gains are for the same interval (p_i, p_{i+1}) then the α -gain precedes the β -gain.

We call the resulting sorted sequence the *gain sequence* induced by P on V_n . We denote this sequence by $\Sigma_{\text{gain}}(V_n, P)$ or, when P and V_n are clear from the context, by Σ_{gain} .

The canonical strategy of \mathcal{Q} and sequence representations. Given the multiset V_n , a strategy P and value ℓ , player \mathcal{Q} can compute an optimal strategy as follows. First she computes the gain sequence $\Sigma_{\text{gain}}(V_n, P)$ and chooses the first ℓ gains in $\Sigma_{\text{gain}}(V_n, P)$. Then for each $0 \leq i \leq k$ she proceeds as follows. When $\text{gain}_\alpha(V_n, p_i, p_{i+1})$ and $\text{gain}_\beta(V_n, p_i, p_{i+1})$ are both chosen, she places two points in (p_i, p_{i+1}) that win all voters in (p_i, p_{i+1}) ; when only

$\text{gain}_\alpha(V_n, p_i, p_{i+1})$ is chosen, she places one point in (p_i, p_{i+1}) that win $\text{gain}_\alpha(V_n, p_i, p_{i+1})$ voters. (By Observation 2 and the tie-breaking rules, when $\text{gain}_\beta(V_n, p_i, p_{i+1})$ is chosen it is always the case that $\text{gain}_\alpha(V_n, p_i, p_{i+1})$ is also chosen.) The resulting strategy Q is optimal as highest possible gains are chosen and is called the *canonical strategy* of Q with ℓ points against P on V_n .

From now on we restrict the strategies of player Q to canonical strategies. In a canonical strategy, player Q places at most two points in any interval induced by a strategy $P = \{p_0, \dots, p_{k+1}\}$, and when we know that Q places a single point (and similarly when she places two points) then we also know where to place the point(s). Hence, we can represent a canonical strategy Q , for given V_n and P , by a sequence $M(V, P, Q) := \langle m_0, \dots, m_k \rangle$ where $m_i \in \{0, 1, 2\}$ indicates how many points Q plays in the interval (p_i, p_{i+1}) . We call $M(V, P, Q)$ the *sequence representation* of the strategy Q against P on V_n . We denote the sequence representation of the canonical strategy of Q with ℓ points against P on V_n by $M(V, P, \ell)$.

► **Observation 3.** *The canonical strategy of Q with ℓ points against P is the optimal strategy Q with ℓ points against P which has lexicographically maximal sequence representation.*

2.2 The Subproblems for a Dynamic-Programming Solution

For clarity, in the rest of Section 2 we assume the multiset of voters V does not have repetitive entries, i.e we have a set of voters, and not a multiset. While all the results are easily extendible to multisets, dealing with them adds unnecessary complexity to the text.

Our goal is to develop a dynamic-programming algorithm to compute $\Gamma_{k^*, \ell^*}(V)$. Before we can define the subproblems on which the dynamic program is based, we need to introduce the concept of *thresholds*, which is a crucial ingredient in the subproblems.

Strict and loose thresholds. Recall that in an arbitrary gain sequence $\Sigma_{\text{gain}}(V_n, P) = \langle \tau_1, \dots, \tau_{2k+2} \rangle$, each τ_j is the α -gain or β -gain of some interval (p_i, p_{i+1}) , and that these gains are sorted in non-increasing order. We call any integer value $\tau \in [\tau_{\ell+1}, \tau_\ell]$ an ℓ -*threshold* for Q induced by P on V_n , or simply a *threshold* if ℓ is clear from the context. We implicitly assume $\tau_0 := n$ so that talking about 0-threshold is also meaningful. Note that when $\tau_\ell \geq \tau > \tau_{\ell+1}$ then the canonical strategy for Q chooses all gains larger than τ and no gains smaller or equal to τ . Hence, we call τ a *strict* threshold if $\tau_\ell \geq \tau > \tau_{\ell+1}$. On the other hand, when $\tau = \tau_{\ell+1}$ then gains of value τ may or may not be chosen by the canonical strategy of Q . (Note that in this case for gains of value τ to be picked, we would actually need $\tau_\ell = \tau = \tau_{\ell+1}$.) In this case we call τ a *loose* threshold.

The idea will be to guess the threshold τ in an optimal solution and then use the fact that fixing the threshold τ helps us to limit the strategies for \mathcal{P} and anticipate the behavior of Q . Let P_{opt} be the canonical strategy realizing $\Gamma_{k^*, \ell^*}(V)$. We call any ℓ^* -threshold of P_{opt} an *optimal threshold*. We devise an algorithm that gets a value τ as input and computes $\Gamma_{k^*, \ell^*}(V)$ correctly if τ is an optimal threshold, and computes a value not greater than $\Gamma_{k^*, \ell^*}(V)$, otherwise.

Clearly we only need to consider values of τ that are at most n^* . In fact, since each α -gain or β -gain in a given gain sequence corresponds to a unique subset of voters, the ℓ^* -th largest gain can be at most n^*/ℓ^* , so we only need to consider τ -values up to $\lfloor n^*/\ell^* \rfloor$. Observe that when there exists an optimal strategy that induces an ℓ^* -threshold equal to zero, then Q can win all voters not explicitly covered by P . In this case the trivial strategy is optimal for \mathcal{P} . Our global algorithm is now as follows.

1. For all thresholds $\tau \in \{1, \dots, \lfloor n^*/\ell^* \rfloor\}$, compute an upper bound on the number of voters \mathcal{P} can win with a strategy that has an ℓ^* -threshold τ . For the run where τ is an optimal threshold, the algorithm will return $\Gamma_{k^*, \ell^*}(V)$.
2. Compute the number of voters \mathcal{P} wins in the game $\langle V, k^*, \ell^* \rangle$ by the trivial strategy.
3. Report the best of all solutions found.

The subproblems for a fixed threshold τ . From now on we consider a fixed threshold value $\tau \in \{1, \dots, \lfloor n^*/\ell^* \rfloor\}$. The subproblems in our dynamic-programming algorithm for the game $\langle V, k^*, \ell^* \rangle$ have several parameters.

- A parameter $n \in \{0, \dots, n^*\}$, specifying that the subproblem is on the voter set V_n .
- Parameters $k, \ell \in \{0, \dots, n\}$, specifying that \mathcal{P} can use $k + 1$ points and \mathcal{Q} can use ℓ points.
- A parameter $\gamma \in \{0, \dots, n\}$, specifying the number of voters \mathcal{P} must win.
- A parameter $\delta \in \{\text{strict}, \text{loose}\}$, specifying the strictness of the fixed ℓ -threshold τ .

Intuitively, the subproblem specified by a tuple $\langle n, k, \ell, \gamma, \delta \rangle$ asks for a strategy P where \mathcal{P} wins at least γ voters from V_n and such that P that induces an ℓ -threshold of strictness δ , against an opponent \mathcal{Q} using ℓ points. Player \mathcal{P} may use $k + 1$ points and his objective will be to push his last point, p_{k+1} as far to the right as possible. The value of the solution to such a subproblem, which we denote by $X_{\max}(n, k, \ell, \gamma, \delta)$, will indicate how far to the right we can push p_{k+1} . Ultimately we will be interested in solutions where \mathcal{P} can push p_{k^*+1} all the way to $+\infty$, which means he can actually win γ voters by placing only k^* points.

To formally define $X_{\max}(n, k, \ell, \gamma, \delta)$, we need two final pieces of notation. Let $x \in \mathbb{R} \cup \{-\infty\}$, let $n \in \{1, \dots, n^*\}$, and let a, b be integers. For convenience, define $v_{n^*+1} := \infty$. Now we define the (a, b) -span of x to v_{n+1} , denoted by $\text{span}(x, n, a, b)$, as

$$\text{span}(x, n, a, b) := \begin{cases} \text{the maximum real value } y \in (v_n, v_{n+1}] \text{ such that} & \text{if } x \neq -\infty \text{ and } y \text{ exists} \\ \text{gain}_\alpha(V, x, y) = a \text{ and gain}_\beta(V, x, y) = b & \\ -\infty & \text{otherwise.} \end{cases}$$

If we let $a := \text{gain}_\alpha(V_n, x, y)$ and $b := \text{gain}_\beta(V_n, x, y)$, then player \mathcal{Q} wins either 0, a , or $a + b$ points depending on whether she plays 0, 1, or 2 points inside the interval. It will therefore be convenient to introduce the notation \oplus_j for $j \in \{0, 1, 2\}$, which is defined as

$$a \oplus_0 b := 0, \quad a \oplus_1 b := a, \quad a \oplus_2 b := a + b.$$

We assume the precedence of these operators are higher than addition.

► **Definition 4.** For parameters $n \in \{0, \dots, n^*\}$, $k, \ell, \gamma \in \{0, \dots, n\}$, and $\delta \in \{\text{strict}, \text{loose}\}$, we define the value $X_{\max}(n, k, \ell, \gamma, \delta)$ and what it means when a strategy P realizes this, as follows.

- For $k = 0$, we call it an elementary subproblem, and define $X_{\max}(n, k, \ell, \gamma, \delta) = v_{n+1}$ if
 1. $\{v_{n+1}\}$ wins at least γ voters from V_n , and
 2. $\{v_{n+1}\}$ induces an ℓ -threshold τ with strictness δ on V_n ,
and we define $X_{\max}(n, k, \ell, \gamma, \delta) = -\infty$ otherwise. In the former case we say that $P := \{v_{n+1}\}$ realizes $X_{\max}(n, k, \ell, \gamma, \delta)$.
- For $k > 0$, we call it a non-elementary subproblem, and $X_{\max}(n, k, \ell, \gamma, \delta)$ is defined to be equal to the maximum real value $y \in (v_n, v_{n+1}]$ such that there exists a strategy $P := P' \cup \{y\}$ with $P' = \{p_1, \dots, p_k\}$, integer values n', a, b with $0 \leq n' < n$ and $0 \leq a, b \leq n$, an integer $j \in \{0, 1, 2\}$, and a $\delta' \in \{\text{strict}, \text{loose}\}$ satisfying the following conditions:

1. P wins at least γ voters from V_n ,
 2. P induces an ℓ -threshold τ with strictness δ on V_n ,
 3. $\text{span}(p_k, n, a, b) = y$,
 4. P' realizes $X_{\max}(n', k-1, \ell-j, \gamma-n+n'+a \oplus_j b, \delta')$,
 5. Let $M(V_{n'}, P', \ell-j) := \langle m'_0, \dots, m'_k \rangle$ and $M(V_n, P, \ell) := \langle m_0, \dots, m_{k+1} \rangle$. Then $m'_i = m_i$ for all $0 \leq i < k$.
- When a set P satisfying the conditions exists, we say that P realizes $X_{\max}(n, k, \ell, \gamma, \delta)$. We define $X_{\max}(n, k, \ell, \gamma, \delta) = -\infty$ if no such P exists.

For example, given voters $V = \{1, \dots, 6\}$, $k^* = 3$, $\ell^* = 3$, and $n^* = 6$, $X_{\max}(3, 0, 1, 2, \text{strict}) = -\infty$ because \mathcal{P} cannot win voters without placing any points and $X_{\max}(3, 2, 1, 2, \text{strict}) = 4$ because player \mathcal{P} can easily win two of the first three voters by placing his first two points on them and then push his third point to the far right to position 4; note that $\tau = 1$ is an strict induced 1-threshold in this case.

Intuitively, each prefix of a canonical strategy of \mathcal{P} is a realizing strategy to some of these subproblems, which is consistent with Definition 4 as realizing strategies to subproblems are defined to be a realizing strategy to a smaller subproblem plus the last point of \mathcal{P} pushed to the rightmost possible position, where \mathcal{Q} 's response would be the same except she has the chance to place $j \in \{0, 1, 2\}$ without violating the conditions mentioned in the definition.

By induction we can show that if the parameters n, k, ℓ are not in a certain range, namely if one of the conditions $\ell < 2(k+1)$ or $k \leq \|V_n\|$ is violated, then $X_{\max}(n, k, \ell, \gamma, \delta) = -\infty$. The next lemma shows we can compute $\Gamma_{k^*, \ell^*}(V)$ from the solutions to our subproblems.

► **Lemma 5.** *Let $V = \{v_1, \dots, v_{n^*}\}$ be a set of n^* voters in \mathbb{R}^1 . Let $0 \leq k^* \leq n^*$ and $1 \leq \ell^* \leq n^*$ be two integers such that $\ell^* < 2k^*$ and $k^* < \|V\|$, and let τ be a fixed threshold. Then*

$$\Gamma_{k^*, \ell^*}(V) \geq \text{the maximum value of } \gamma \text{ with } 0 \leq \gamma \leq n^* \text{ for which there exist a } \delta \in \{\text{loose, strict}\} \text{ such that } X_{\max}(n^*, k^*, \ell^*, \gamma, \delta) = \infty. \quad (1)$$

Moreover, for an optimal threshold $\tau_{\text{opt}} > 0$, the inequality changes to equality.

► **Remark.** Usually in dynamic programming, subproblems have a clean non-recursive definition – the recursion only comes in when a recursive formula is given to compute the value of an optimal solution. Our approach is more complicated: Definition 4 above gives a recursive subproblem definition (and Lemma 5 shows how to use it), however, using this recursive formula to compute solutions is not feasible and Lemma 7 below will then give a different recursive formula to actually compute the solutions to the subproblems.

2.3 Computing Solutions to Subproblems

The solution to an elementary subproblem follows fairly easily from the definitions, and can be computed in constant time; see the full version.

By definition, in order to obtain a strategy P realizing the solution to a non-elementary subproblem $I = \langle n, k, \ell, \gamma, \delta \rangle$ of size k , we need a solution to a smaller subproblem $I' = \langle n', k-1, \ell', \gamma', \delta' \rangle$ of size $k-1$ and add one point $y \in (v_n, v_{n+1}]$ to the strategy $P' = \{p_1, \dots, p_k\}$ realizing I' . Thus by adding y , we extend the solution to I' to get a solution to I . To find the “right” subproblem I' , we guess some values for $n', a, b, j \in \{0, 1, 2\}$, and $\delta' \in \{\text{strict, loose}\}$; these values are enough to specify I' . We note that there are just a polynomial number of cases and therefore a polynomial number of values for the value $y \in (v_n, v_{n+1}]$ which we want to maximize. Namely, there are $O(n)$ choices for the values n', a , and b , three

choices for j , and two choices for δ' . This makes $O(n^3)$ different cases to be considered for each subproblem I , in total. However, not all those subproblems can be extended to I . In the following definition, we list all the triples (δ', j, δ) that can guarantee the extendibility of I' to I .

Let a and b be the α -gain and β -gain of the interval (p_k, y) in a strategy $P = \{p_1, \dots, p_k, y\}$ with threshold τ . We define the following sets of triples:

$$\Delta(\tau, a, b) := \begin{cases} \{(\text{loose}, 2, \text{loose}), (\text{strict}, 2, \text{strict})\} & \text{if } a > \tau \wedge b > \tau \\ \{(\text{loose}, 1, \text{loose}), (\text{strict}, 1, \text{loose}), (\text{strict}, 2, \text{strict})\} & \text{if } a > \tau \wedge b = \tau \\ \{(\text{loose}, 1, \text{loose}), (\text{strict}, 1, \text{strict})\} & \text{if } a > \tau \wedge b < \tau \\ \{(\text{loose}, 0, \text{loose}), (\text{strict}, 0, \text{loose}), (\text{strict}, 1, \text{loose}), (\text{strict}, 2, \text{strict})\} & \text{if } a = \tau \wedge b = \tau \\ \{(\text{loose}, 0, \text{loose}), (\text{strict}, 0, \text{loose}), (\text{strict}, 1, \text{strict})\} & \text{if } a = \tau \wedge b < \tau \\ \{(\text{loose}, 0, \text{loose}), (\text{strict}, 0, \text{strict})\} & \text{if } a < \tau \wedge b < \tau. \end{cases}$$

► **Lemma 6.** *Let $P' = \{p_1, \dots, p_k\}$ and $P := P' \cup \{y\}$, be two reasonable strategies on $V_{n'}$ and V_n , where $n' = \operatorname{argmax}_{1 \leq i \leq n^*} v_i < p_k$, $n = \operatorname{argmax}_{1 \leq i \leq n^*} v_i < y$, and $y \in (v_n, v_{n+1}]$. Let $a = \operatorname{gain}_\alpha(V_n, p_k, y)$ and $b = \operatorname{gain}_\beta(V_n, p_k, y)$, and assume $\tau > 0$ is an $(\ell - j)$ -threshold of strictness δ' for \mathcal{Q} induced by P' on $V_{n'}$, where $j \in \{0, 1, 2\}$. Then, there exists a triple $(\delta', j, \delta) \in \Delta(\tau, a, b)$ if and only if*

1. P induces an ℓ -threshold τ with strictness δ on V_n .
2. Let $M(V_{n'}, P', \ell - j) := \langle m'_0, \dots, m'_k \rangle$ and $M(V_n, P, \ell) := \langle m_0, \dots, m_{k+1} \rangle$. Then $m'_i = m_i$ for all $0 \leq i < k$.

Moreover, this triple is unique if it exists.

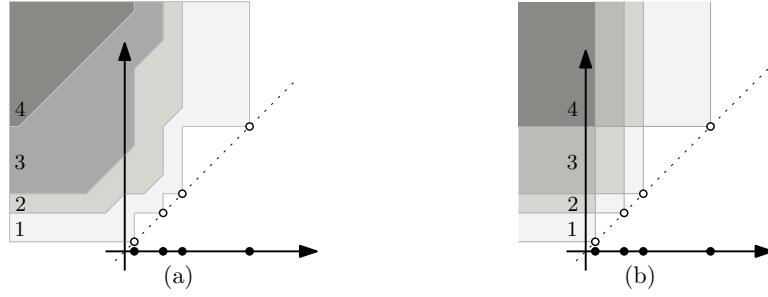
► **Lemma 7.** *For a non-elementary subproblem $I = \langle n, k, \ell, \gamma, \delta \rangle$, we have*

$$X_{\max}(n, k, \ell, \gamma, \delta) = \max_{0 \leq n' < n} \max_{0 \leq a \leq n} \max_{0 \leq b \leq n} \max_{(\delta', j, \delta) \in \Delta(\tau, a, b)} \operatorname{span}(X_{\max}(n', k - 1, \ell - j, \gamma - n + n' + a \oplus_j b, \delta'), n, a, b).$$

If we can compute the span function efficiently, we can compute all the solutions by dynamic programming and solve the problem. However, a solution based on a trivial dynamic programming will have running time $\lfloor n^*/\ell^* \rfloor \cdot O(k^* \ell^* (n^*)^2) \cdot O((n^*)^3 f(n^*)) = O(k^* (n^*)^6 f(n^*))$, where $\lfloor n^*/\ell^* \rfloor$ is the total number of choices for the threshold, $O(k^* \ell^* (n^*)^2)$ is the number of subproblems for each threshold, and $O((n^*)^3 f(n^*))$ is the time needed to solve each subproblem where $f(n)$ is the time needed to compute the span function. This algorithm is quite slow. More importantly it is not easy to compute the span function. In the following, we introduce some new concepts to compute the span function and also get a better running time.

2.4 Computing the span Function Using Gain Maps

Before we give the algorithm we introduce the *gain map*, which we need to compute the span function. Consider an arbitrary strategy P of \mathcal{P} on V , and recall that such a strategy induces open intervals of the form (p_i, p_{i+1}) where \mathcal{Q} can place her points. We can represent any interval (x, y) that may arise in this manner as a point (x, y) in the plane. Thus the locus of all possible intervals is the region $R := \{(x, y) : x < y\}$. We will define two subdivisions of this region, the A -map and the B -map, and the gain map will then be the overlay of the A -map and the B -map.



■ **Figure 1** **a)** A -map of $V = \{1, 4, 6, 13\}$ with the corresponding α -gain for each region. **b)** B -map of V with the corresponding β -gain for each region.

The A -map is the subdivision of R into regions A^t and B^t , for $0 \leq t \leq n^*$, defined as $A^t := \{(x, y) : \text{gain}_\alpha(V, x, y) = t\}$ and $B^t := \{(x, y) : \text{gain}_\alpha(V, x, y) + \text{gain}_\beta(V, x, y) = t\}$. In other words, A^t is the locus of all intervals (x, y) such that, if (x, y) is an interval induced by P , then \mathcal{Q} can win t voters (but no more than t) from $V \cap (x, y)$ by placing a single point in (x, y) . To construct the A -map, let $A^{\geq t}$ denote the locus of all intervals (x, y) such that $\text{gain}_\alpha(V, x, y) \geq t$. Note that $A^t = A^{\geq t} \setminus A^{\geq t+1}$. For $1 \leq i \leq n^* - t + 1$, let $V_i^t := \{v_i, \dots, v_{i+t-1}\}$ and define

$$A_i^{\geq t} := \{(x, y) : V_i^t \subset (x, y) \text{ and } \mathcal{Q} \text{ can win } V_i^t \text{ by placing a single point in } (x, y)\}.$$

Thus $A_i^{\geq t} = \{(x, y) : x < v_i \text{ and } y > v_{i+t-1} \text{ and } y > x + 2(v_{i+t-1} - v_i)\}$. Here the conditions $x < v_i$ and $y > v_{i+t-1}$ are needed to guarantee that $V_i^t \subset (x, y)$. The condition $y > x + 2(v_{i+t-1} - v_i)$ implies that V_i^t can be covered with an interval of length $(y - x)/2$, which is necessary and sufficient for \mathcal{Q} to be able to win all these voters. Note that each region $A_i^{\geq t}$ is the intersection of three halfplanes, bounded by a vertical, a horizontal and a diagonal line, respectively.

Since \mathcal{Q} can win at least t voters in inside (x, y) with a single point if she can win at least t consecutive voters with a single point, we have $A^{\geq t} = \bigcup_{i=1}^{n^*-t+1} A_i^{\geq t}$. Thus $A^{\geq t}$ is a polygonal region, bounded from below and from the right by a a polyline consisting of horizontal, vertical, and diagonal segments, and the regions A^t are sandwiched between such polylines; see Figure 1a. We call the polylines that form the boundary between consecutive regions A^t *boundary polylines*.

The B -map can be constructed in a similar, but easier manner. Indeed, B^t is the locus of all intervals such that \mathcal{Q} can win t voters (but no more) from $V \cap (x, y)$, and this is the case if and only if $|V \cap (x, y)| = t$. Hence, B^t is the union of the rectangular regions $[v_i, v_{i+1}] \times (v_{i+t}, v_{i+t+1}]$ (intersected with R), for $0 \leq i \leq n^* - t$, where $v_0 := -\infty$ and $v_{n^*+1} := \infty$, as shown in Figure 1b.

As mentioned, by overlaying the A - and B -map, we get the *gain map*. For any given region on this map, the corresponding intervals have equal α -gain and equal β -gain.

► **Lemma 8.** *The complexity of the gain-map is $O((n^*)^2)$.*

Proof. The boundary polylines in the A -map are xy -monotone and comprised of vertical, horizontal, and diagonal lines. The B -map is essentially a grid of size $O((n^*)^2)$ defined by the lines $x = v_i$ and $y = v_i$, for $1 \leq i \leq n^*$. Since each of these lines intersects any xy -monotone polyline at most once – in a point or in a vertical segment – the complexity of the gain map is also $O((n^*)^2)$. ◀

Using the gain map, we can compute the values $\text{span}(x_0, n, a, b)$ for a given $x_0 \in \mathbb{R}$ and for all triples n, a, b satisfying $1 \leq n \leq n^*$, and $0 \leq a \leq n^*$ and $0 \leq b \leq n^*$, as follows. First, we compute the intersection points of the vertical line $x = x_0$ with (the edges of) the gain map, sorted by increasing y -coordinates. (If this line intersects the gain map in a vertical segment, we take the topmost endpoint of the segment.) Let $(x_0, y_1), \dots, (x_0, y_z)$ denote this sorted sequence of intersection points, where $z \leq 2n^*$ denotes the number of intersections. Let a_i and b_i denote the α -gain and β -gain of the interval corresponding to the point (x_0, y_i) , and let a_{z+1} and b_{z+1} denote the α -gain and β -gain of the unbounded region intersected by the line $x = x_0$. Define $n_i = \operatorname{argmax}_n v_n < y_i$. Then we have

$$\text{span}(x_0, n, a, b) = \begin{cases} y_i & \text{if } a = a_i \text{ and } b = b_i, \text{ and } n = n_i, \text{ for some } 1 \leq i \leq z \\ +\infty & \text{if } a = a_{z+1} \text{ and } b = b_{z+1} \text{ and } n = n^* \\ -\infty & \text{for all other triples } n, a, b \end{cases} \quad (2)$$

Our algorithm presented below moves a sweep line from left to right over the gain map. During the sweep we maintain the intersections of the sweep line with the gain map. It will be convenient to maintain the intersections with the A -map and the B -map separately. We will do so using two sequences, $A(x_0)$ and $B(x_0)$.

- The sequence $A(x_0)$ is the sequence of all diagonal or horizontal edges in the A -map that are intersected by the line $x = x_0$, ordered from bottom to top along the line. (More precisely, the sequence contains (at most) one edge for any boundary polyline. When the sweep line reaches the endpoint of such an edge, the edge will be removed and it will be replaced by the next non-vertical edge of that boundary polyline, if it exists.)
- The sequence $B(x_0)$ is the sequence of the y -coordinates of the horizontal segments in the B -map intersected by the line $x = x_0$, ordered from bottom to top along the line.

The number of intersections of the A -map, and also of the B -map, with the line $x = x_0$ is equal to $n^* - n_0 + 1$, where $n_0 = \operatorname{argmin}_n v_n > x_0$. Hence, the sequences $A(x_0)$ and $B(x_0)$ have length $n^* - n_0 + 1 \leq n^* + 1$.

If we have the sequences $A(x_0)$ and $B(x_0)$ available then, using Equation (2), we can easily find all triples n, a, b such that $\text{span}(x_0, n, a, b) \neq -\infty$ (and the corresponding y -values) by iterating over the two sequences. We can summarize the results of this section as follows.

► **Observation 9.** *Given the sequences $A(x_0)$ and $B(x_0)$, we can compute all the values $\text{span}(x_0, n, a, b)$ with $1 \leq n \leq n^*$ and $0 \leq a, b \leq n$ that are not equal to $-\infty$ in $O(n^*)$ time in total.*

This observation, together with Lemma 7 forms the basis of our dynamic-programming algorithm.

2.5 The Sweep-Line Based Dynamic-Programming Algorithm

We will use a sweep-line approach, moving a vertical line from left to right over the gain map. We will maintain a table X , indexed by subproblems, such that when the sweep line is at position x_0 , then $X[I]$ holds the best solution known so far for subproblem I , where the effect of all the subproblems with solution smaller than x_0 have been taken into account. When our sweep reaches a subproblem I' , then we check which later subproblems I can use I' in their solution, and we update the solutions to these subproblems.

Recall that the algorithm works with a fixed threshold value $\tau \in \{1, \dots, \lfloor n^*/\ell^* \rfloor\}$ and that its goal is to compute the values $X_{\max}(n^*, k^*, \ell^*, \gamma, \delta)$ for all $0 \leq \gamma \leq n^*$ and $\delta \in \{\text{strict}, \text{loose}\}$. Our algorithm maintains the following data structures.

- $A[0..n^*]$ is an array that stores the sequence $A(x_0)$, where x_0 is the current position of the sweep line and $A[i]$ contains the i -th element in the sequence. When the i -th element does not exist then $A[i] = \text{NIL}$.
- Similarly, $B[0..n^*]$ is an array that stores the sequence $B(x_0)$.
- X : This is a table with an entry for each subproblem $I = \langle n, k, \ell, \gamma, \delta \rangle$ with $0 \leq n \leq n^*$, and $0 \leq k \leq k^*$ and $0 \leq \ell \leq \ell^*$, and $0 \leq \gamma \leq n^*$ and $\delta \in \{\text{strict}, \text{loose}\}$. When the sweep line is at position x_0 , then $X[I]$ holds the best solution known so far for subproblem I , where the effect of all the subproblems with solution smaller than x_0 have been taken into account. More precisely, in the right-hand side of the equation in Lemma 7 we have taken the maximum value over all subproblems $I' = \langle n', k-1, \ell-j, \gamma-n+n'+a \oplus_j b, \delta' \rangle$ with $X_{\max}(I') < x_0$. In the beginning of the algorithm the entries for elementary subproblems are computed in constant time and all other entries have value $-\infty$.
- E : This is the event queue, which will contain four types of events, as explained below. The event queue E is a min-priority queue on the x -value of the events. There are four types of events, as listed next, and when events have the same x -value then the first event type (in the list below) has higher priority, that is, will be handled first. When two events of the same type have equal x -value then their order is arbitrary. Note that events with the same x -value are not degenerate cases – this is inherent to the structure of the algorithms, as many events take place at x -coordinates corresponding to voters.

A -map events, denoted by $e_A(a, s, s')$: At an A -map event, the edge s of the A -map ends – thus the x -value of an A -map event is the x -coordinate of the right endpoint of s – and the array A must be updated by replacing it with the edge s' . Here s' is the next non-vertical edge along the boundary polyline that s is part of, where $s' = \text{NIL}$ if s is the last non-vertical edge of the boundary polyline. The value a indicates that the edge s is on the boundary polyline between A^a and A^{a+1} . In other words s (and s' , if it exist) are the a -th intersection point, $0 \leq a < n^*$, with the A -map along the current sweep line, and so we must update the entry $A[a]$ by setting $A[a] \leftarrow s'$.

B -map events, denoted by $e_B(v_n)$: At a B -map event, a horizontal edge of the B -map ends. This happens when the sweep line reaches a voter v_n – that is, when $x_0 = x_n$ – and so the x -value of this event is v_n . The bottommost intersection of the sweep line with the B -map now disappears (see Figure 1b), and so we must update B by shifting all other intersection points one position down in B and setting $B[n^* - n] \leftarrow \text{NIL}$.

Subproblem events, denoted by $e_X(n', k', \ell', \gamma', \delta')$: At a subproblem event the solution to the subproblem given by $I' = \langle n', k', \ell', \gamma', \delta' \rangle$ is known and the x -value of this event is equal to $X_{\max}(I')$. Handling the subproblem event for I' entails deciding which later subproblems I can use I' in their solution and how they can use it, using the sets $\Delta(\tau, a, b)$, and updating the solutions to these subproblems.

In the beginning of the algorithm all the events associated with elementary subproblems are known. The events associated with non-elementary subproblems are added to the event queue when handling an update event $e_E(v_n)$, as discussed next.

Update events, denoted by $e_E(v_n)$: At the update event happening at x -value v_n , all subproblem events of size n are added to the event queue E . These are simply the subproblems $\langle n, k, \ell, \gamma, \delta \rangle$ for all $k, \ell, \gamma \in \{0, \dots, n\}$ and $\delta \in \{\text{strict}, \text{loose}\}$. The reason we could not add them at the start of the algorithm was that the x -value of such a subproblem I was now known yet. However, when we reach v_n then $X_{\max}(I)$ is determined, so we can add the event to E with $X_{\max}(I)$ as its x -value.

The pseudocode below summarizes the algorithm.

■ **Algorithm 1** COMPUTESOLUTIONS(τ, V, k^*, ℓ^*).

```

1 for  $i \leftarrow 0$  to  $n^* - 1$  do
2    $A[i] \leftarrow (v_i, v_{i+1}) - (v_{i+1}, v_{i+1}); \quad B[i] \leftarrow v_{i+1}$            ▷ define  $v_0 := v_1 - 1$ 
3  $A[n^*] \leftarrow \text{NIL}; \quad B[n^*] \leftarrow \text{NIL}$ 
4 Initialize  $X$  by the solutions to elementary subproblems
5 Initialize  $E$  by all map events, update events, and elementary subproblem events
6 while  $E$  is not empty do
7    $e \leftarrow \text{extractMin}(E); \quad x_0 \leftarrow x\text{-value of } e$ 
8   switch  $e$  do
9     case  $e_A(a, s, s')$  do
10       $A[a] \leftarrow s'$ 
11     case  $e_E(v_n)$  do
12       $B[n^* - n] \leftarrow \text{NIL}$ 
13      for  $i \leftarrow 0$  to  $n^* - n - 1$  do
14         $B[i] \leftarrow v_{n+i+1}$ 
15     case  $e_X(n', k', \ell', \gamma', \delta')$  do
16      for all span( $x_0, n, a, b$ ) =  $y$  where  $y \neq -\infty$  do
17        for all  $(\delta', j, \delta) \in \Delta(\tau, a, b)$  do
18           $I \leftarrow \langle n, k' + 1, \ell' + j, \gamma' + n - n' - \text{gain}^j(a, b), \delta \rangle$ 
19           $X(I) \leftarrow \max(X(I), y)$ 
20     case  $e_E(v_n)$  do
21      Add all the events for subproblems of size  $n$  to  $E$ , as explained above

```

► **Lemma 10.** *Algorithm 1 correctly computes the solutions for subproblems $\langle n, k, \ell, \gamma, \delta \rangle$ for the given value τ , for all $n, k, \ell, \gamma, \delta$ with $0 \leq n \leq n^*$, and $0 \leq k, \ell, \gamma \leq n$, and $\delta \in \{\text{strict}, \text{loose}\}$, and $\ell < 2(k + 1)$. The running time of the algorithm is $O(k^* \ell^* (n^*)^3)$.*

Proof. We handle the A -map and B -map events before a subproblem event so that A and B data structures are up-to-date when we want to compute the span function on handling a subproblem event. We also handle a subproblem event before an update event so that when we want to add a new subproblem event to the event queue on handling an update event, its entry in table X has the correct value. The correctness of the algorithm now follows from the discussion and lemmas above.

The running time is dominated by the handling of the subproblem events. By Observation 9, the algorithm handles each subproblem in $O(n^*)$ time, plus $O(\log n^*)$ for operations on the event queue, and there are $O(k^* \ell^* (n^*)^2)$ subproblems. Hence, the total running time is $O(k^* \ell^* (n^*)^3)$. ◀

By Lemmas 5 and 10, the algorithm described at the beginning of Section 2.2 computes $\Gamma_{k^*, \ell^*}(V)$ correctly. Since this algorithm calls COMPUTESOLUTIONS $\lfloor n^*/\ell^* \rfloor$ times in Step 1, we obtain the following theorem.

► **Theorem 11.** *There exists an algorithm that computes $\Gamma_{k^*, \ell^*}(V)$, and thus solves the one-dimensional case of the one-round discrete Voronoi game, in time $O(k^* (n^*)^4)$.*

► **Remark.** We can also solve the one-dimensional case of the one-round discrete Voronoi game when voters are weighted, i.e. each voter $v \in V$ has an associated weight $\omega(v)$ and the players try to maximize the total weight of the voters they win. In this case, the α -gain and β -gain of an interval is defined as the total weight of voters the second player can win in that interval by placing one point and two points, respectively. The number of possible thresholds is not an integer in range $[0, n^*]$, but the sum of any sequence of consecutive voters define a threshold, which makes a total of $O((n^*)^2)$ different thresholds. The gain map also becomes more complex and in the algorithm we need to spend $O((n^*)^2)$ time (instead of $O(n^*)$) to handle each subproblem event, which results in an algorithm with running time $O(k^* \ell^* (n^*)^5)$.

3 Containment in $\exists\forall\mathbb{R}$ and the Algorithm for $d \geq 2$

We now consider the one-round discrete Voronoi game in the L_p -norm, for some arbitrary p . Then a strategy $P = \{p_1, \dots, p_k\}$ can win a voter $v \in V$ against a strategy $Q = \{q_1, \dots, q_\ell\}$ if and only if the following Boolean expression is satisfied:

$$\text{win}(v) := \bigvee_{i \in [k]} \bigwedge_{j \in [\ell]} (\text{dist}_p(p_i, v))^p \leq (\text{dist}_p(q_j, v))^p,$$

where dist_p is the L_p -distance. This expression has $k\ell$ polynomial inequalities of degree p . The strategy P is winning if and only if the majority of the expressions $\text{win}(v_1), \dots, \text{win}(v_n)$ are true. Having a majority function *Majority* that evaluates to true if at least half of its parameters evaluates to true, player \mathcal{P} has a winning strategy if and only if

$$\begin{aligned} &\exists x_1(p_1), \dots, x_d(p_1), \dots, x_1(p_k), \dots, x_d(p_k) \\ &\forall x_1(q_1), \dots, x_d(q_1), \dots, x_1(q_\ell), \dots, x_d(q_\ell) : \text{Majority}(\text{win}(v_1), \dots, \text{win}(v_n)) \end{aligned}$$

is true, where $x_i(\cdot)$ denotes the i -th coordinate of a point.

Ajtai et al. [2] show that it is possible to construct a sorting network, often called the AKS sorting network, composed of comparison units configured in $c \cdot \log n$ levels, where c is a constant and each level contains exactly $\lfloor n/2 \rfloor$ comparison units. Each comparison unit takes two numbers as input and outputs its input numbers in sorted order. Each output of a comparison unit (except those on the last level) feeds into exactly one input of a comparison unit in the next level, and the input numbers are fed to the inputs of the first level. The outputs of the comparison units in the last level (i.e., the outputs of the network) give the numbers in sorted order.

Using AKS sorting networks we can construct a Boolean formula of size $O(n^c)$ for some constant c that tests if the majority of its n inputs are true as follows. Assuming the boolean value *false* is smaller than the boolean value *true* value, we make an AKS sorting network that sorts n boolean values. This is possible using comparison units that get p and q as input, and output $p \wedge q$ and $p \vee q$. It is not hard to verify that the $\lceil n/2 \rceil$ -th output of the network is equal to the value of the majority function on the input boolean values. By construction, we can write the Boolean formula representing the value of this output as *logical and* (\wedge) and *logical or* (\vee) combination of the input boolean values, and the size of the resulting formula is $O(n^c)$.

Thus we can write $\text{Majority}(\text{win}(v_1), \dots, \text{win}(v_n))$ as a Boolean combination of $O(n^c k \ell)$ polynomial inequalities of degree p , where each quantified block has $k d$ and ℓd variables respectively. Basu et al. [7] gave an efficient algorithm for deciding the truth of quantified

formulas. For our formula this gives an algorithm with $O((n^c k \ell)^{(kd+1)(\ell d+1)} p^{k \ell d^2})$ running time to decide if \mathcal{P} has a winning strategy for a given instance $\langle V, k, \ell \rangle$ of the Voronoi game problem. Note that this is polynomial when k, ℓ and d are constants.

For the L_∞ norm, we can define $F(v)$ as follows:

$$F(v) := \bigvee_{i \in [k]} \bigwedge_{j \in [\ell]} \bigvee_{s' \in [d]} \bigwedge_{s \in [d]} |x_s(p_i) - x_s(v)| \leq |x_{s'}(q_j) - x_{s'}(v)|,$$

By comparing the squared values instead of the absolute values, we have a formula which demonstrates that even with the L_∞ norm, the problem is contained in $\exists \forall \mathbb{R}$ and there exists an algorithm of complexity $O((n^c k \ell d^2)^{(kd+1)(\ell d+1)} 2^{k \ell d^2})$ to solve it.

► **Theorem 12.** *The one-round discrete Voronoi game $\langle V, k, \ell \rangle$ in \mathbb{R}^d with the L_p norm is contained in $\exists \forall \mathbb{R}$. Moreover, for fixed k, ℓ, d there exists an algorithm that solves it in polynomial time.*

De Berg et al. [9] introduced the notion of personalized preferences. More precisely, given a natural number p , assuming each axis defines an aspect of the subject voters are voting for, the voter v_i gives different weights to different axes, and v_i has a weighted L_p distance $(\sum_{j \in [d]} w_{ij} (x_j(p) - x_j(v_i))^p)^{1/p}$ from any point $p \in \mathbb{R}^d$. For the weighted L_∞ distance, v_i is at distance $\max_{j \in [d]} (w_{ij} |x_j(p) - x_j(v_i)|)$ from any point $p \in \mathbb{R}^d$. This approach also works when voters have personalized preferences.

4 Σ_2^P -Hardness for $d \geq 2$

In this section, we present the most important ideas behind our proof that the one-round discrete Voronoi game is Σ_2^P -hard in \mathbb{R}^2 . To prove Σ_2^P -hardness, it suffices to show that deciding if \mathcal{Q} has a winning strategy against every possible strategy of \mathcal{P} is Π_2^P -hard. Our proof will use a reduction from a special case of the quantified Boolean formula problem (QBF), as defined next. Let $S := \{s_1, \dots, s_{n_s}\}$ and $T := \{t_1, \dots, t_{n_t}\}$ be two sets of variables, and let $\bar{S} := \{\bar{s}_1, \dots, \bar{s}_{n_s}\}$ and $\bar{T} := \{\bar{t}_1, \dots, \bar{t}_{n_t}\}$ denote their negations. We consider Boolean formulas B of the form

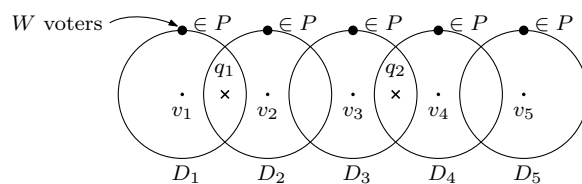
$$B := \forall s_1, \dots, s_{n_s} \exists t_1, \dots, t_{n_t} : c_1 \wedge \dots \wedge c_{n_c}$$

where each clause c_i in $C := \{c_1, \dots, c_{n_c}\}$ is a disjunctive combination of at most three literals from $S \cup \bar{S} \cup T \cup \bar{T}$. Deciding if a formula of this form is true is a Π_2^P -complete problem [17].

Consider the undirected graph $G_B := (N, A)$ representing B , where $N := S \cup T \cup C$ is the set of nodes of G_B and $A := \{(c_i, s_j) : s_j \in c_i \vee \bar{s}_j \in c_i\} \cup \{(c_i, t_j) : t_j \in c_i \vee \bar{t}_j \in c_i\}$ is the set of edges of G_B . Lichtenstein [12] showed how to transform an instance of QBF in polynomial time to an equivalent one whose corresponding graph is planar (and of quadratic size). Thus we may start our reduction from a formula B such that G_B is planar. Our reduction then creates an instance $\langle V, k, \ell \rangle$ of the Voronoi game such that B is true if and only if \mathcal{Q} has a winning strategy.

Define D_i , the *disk of v_i* with respect to a given set P , as the disk with center v_i and radius $\text{dist}(v_i, P)$, that is, D_i is the largest disk centered at v_i that has no point from P in its interior.

► **Observation 13.** *\mathcal{Q} wins a voter v_i against P iff she places a point q in the interior of D_i .*



■ **Figure 2** When all the heavy-weight clusters of W voters are chosen by \mathcal{P} , the best strategy of \mathcal{Q} to win the remaining single voters is to put her points in every other intersection of the disks.

The idea of the construction is that a cluster of W coinciding voters, for a sufficiently large W , forces \mathcal{P} to put a point on top of that cluster. The disk D_i of a voter v_i is then prescribed by cluster closest to v_i . This allows us to create gadgets for the variables s_i , and clause gadgets, consisting of (sets of) disks. Because the graph G_B is planar, we can carry information from the variable gadgets to the clause gadgets using non-crossing chains of disks. This is done in such a way that \mathcal{Q} must either place points in the “even-numbered” intersections or in the “odd-numbered” intersections in a chain, corresponding to the truth settings of the variables t_j ; see Figure 2. An optimal choice of \mathcal{Q} will also carry the bits so that the clauses of B can be checked. The detailed construction is described in the full version.

5 Concluding Remarks

We presented the first polynomial-time algorithm for the one-round discrete Voronoi game in \mathbb{R}^1 . The algorithm is quite intricate, and it would be interesting to see if a simpler (and perhaps faster) algorithm is possible. Finding a lower bound for the 1-dimensional case is also open.

We also showed that the problem is Σ_2^P -hard in \mathbb{R}^2 . Fekete and Meijer [11] conjectured that finding an optimal strategy for the multi-round continuous version of the Voronoi game is PSPACE-complete. We conjecture that in the multi-round version of the discrete version, finding an optimal strategy is PSPACE-hard as well. Note that using the algebraic method presented in this paper, it is easy to show that this problem is contained in PSPACE. While the algebraic method we used is considered a standard technique, it is, as far as we know, the first time this method is combined with polynomial-size boolean formulas for the majority function. We think it should be possible to apply this combination to other problems as well.

References

- 1 Hee-Kap Ahn, Siu-Wing Cheng, Otfried Cheong, Mordecai Golin, and Rene Van Oostrum. Competitive facility location: the Voronoi game. *Theoretical Computer Science*, 310(1-3):457–467, 2004.
- 2 Miklós Ajtai, János Komlós, and Endre Szemerédi. Sorting in $c \log n$ parallel steps. *Combinatorica*, 3(1):1–19, 1983.
- 3 Aritra Banik, Bhaswar B Bhattacharya, and Sandip Das. Optimal strategies for the one-round discrete Voronoi game on a line. *Journal of Combinatorial Optimization*, 26(4):655–669, 2013.
- 4 Aritra Banik, Bhaswar B Bhattacharya, Sandip Das, and Sreeja Das. Two-round discrete Voronoi Game along a line. In *Frontiers in Algorithmics and Algorithmic Aspects in Information and Management*, pages 210–220. Springer, 2013.
- 5 Aritra Banik, Bhaswar B Bhattacharya, Sandip Das, and Satyaki Mukherjee. The discrete Voronoi game in \mathbb{R}^2 . *Computational Geometry*, 63:53–62, 2017.

- 6 Aritra Banik, Jean-Lou De Carufel, Anil Maheshwari, and Michiel Smid. Discrete Voronoi games and epsilon-nets, in two and three dimensions. *Computational Geometry*, 55:41–58, 2016.
- 7 Saugata Basu, Richard Pollack, and Marie-Françoise Roy. On the combinatorial and algebraic complexity of quantifier elimination. *Journal of the ACM*, 43(6):1002–1045, 1996.
- 8 Otfried Cheong, Sarel Har-Peled, Nathan Linial, and Jiří Matoušek. The One-Round Voronoi Game. *Discrete & Computational Geometry*, 31(1):125–138, 2004.
- 9 Mark de Berg, Joachim Gudmundsson, and Mehran Mehr. Faster algorithms for computing plurality points. *ACM Transactions on Algorithms*, 14(3):36:1–36:23, 2018.
- 10 Michael G Dobbins, Linda Kleist, Tillmann Miltzow, and Paweł Rzażewski. $\forall\exists\mathbb{R}$ -Completeness and Area-Universality. In *Proceedings of the 44th International Graph-Theoretic Concepts in Computer Science (WG 2018)*, pages 164–175, 2018.
- 11 Sándor P Fekete and Henk Meijer. The one-round Voronoi game replayed. *Computational Geometry*, 30(2):81–94, 2005.
- 12 David Lichtenstein. Planar formulas and their uses. *SIAM Journal on Computing*, 11(2):329–343, 1982.
- 13 Wei-Yin Lin, Yen-Wei Wu, Hung-Lung Wang, and Kun-Mao Chao. Forming Plurality at Minimum Cost. In *Proceedings of the 9th International Workshop on Algorithms and Computation*, pages 77–88, 2015.
- 14 Richard D McKelvey and Richard E Wendell. Voting equilibria in multidimensional choice spaces. *Mathematics of operations research*, 1(2):144–158, 1976.
- 15 Michael Paterson and Uri Zwick. Shallow circuits and concise formulae for multiple addition and multiplication. *Computational Complexity*, 3(3):262–291, 1993.
- 16 Joachim Spoerhase and H-C Wirth. (r, p) -centroid problems on paths and trees. *Theoretical Computer Science*, 410(47-49):5128–5137, 2009.
- 17 Larry J Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.
- 18 Sachio Teramoto, Erik D Demaine, and Ryuhei Uehara. The Voronoi game on graphs and its complexity. *Journal of Graph Algorithms and Applications*, 15(4):485–501, 2011.
- 19 Yen-Wei Wu, Wei-Yin Lin, Hung-Lung Wang, and Kun-Mao Chao. Computing plurality points and Condorcet points in Euclidean space. In *International Symposium on Algorithms and Computation*, pages 688–698, 2013.

On Explicit Branching Programs for the Rectangular Determinant and Permanent Polynomials

V. Arvind

Institute of Mathematical Sciences (HBNI), Chennai, India
arvind@imsc.res.in

Abhranil Chatterjee

Institute of Mathematical Sciences (HBNI), Chennai, India
abhranilc@imsc.res.in

Rajit Datta

Chennai Mathematical Institute, Chennai, India
rajit@cmi.ac.in

Partha Mukhopadhyay

Chennai Mathematical Institute, Chennai, India
partham@cmi.ac.in

Abstract

We study the arithmetic circuit complexity of some well-known family of polynomials through the lens of parameterized complexity. Our main focus is on the construction of explicit algebraic branching programs (ABP) for determinant and permanent polynomials of the *rectangular* symbolic matrix in both commutative and noncommutative settings. The main results are:

- We show an explicit $O^*\left(\binom{n}{\lfloor k/2 \rfloor}\right)$ -size ABP construction for noncommutative permanent polynomial of $k \times n$ symbolic matrix. We obtain this via an explicit ABP construction of size $O^*\left(\binom{n}{\lfloor k/2 \rfloor}\right)$ for $S_{n,k}^*$, noncommutative symmetrized version of the elementary symmetric polynomial $S_{n,k}$.
- We obtain an explicit $O^*(2^k)$ -size ABP construction for the commutative rectangular determinant polynomial of the $k \times n$ symbolic matrix.
- In contrast, we show that evaluating the rectangular noncommutative determinant over rational matrices is $\#W[1]$ -hard.

2012 ACM Subject Classification Theory of computation \rightarrow Algebraic complexity theory; Theory of computation

Keywords and phrases Determinant, Permanent, Parameterized Complexity, Branching Programs

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.38

1 Introduction

The complexity of arithmetic computations is usually studied in the model of arithmetic circuits and its various restrictions. An *arithmetic circuit* is a directed acyclic graph with each indegree-0 node (called an input gate) labeled by either a variable in $\{x_1, x_2, \dots, x_n\}$ or a scalar from the field \mathbb{F} , and all other nodes (called gates) labeled as either $+$ or \times gate. At a special node (designated the output gate), the circuit computes a multivariate polynomial in $\mathbb{F}[x_1, x_2, \dots, x_n]$. Usually we use the notation $\mathbb{F}[X]$ to denote the polynomial ring $\mathbb{F}[x_1, x_2, \dots, x_n]$.



© V. Arvind, Abhranil Chatterjee, Rajit Datta, and Partha Mukhopadhyay;
licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 38; pp. 38:1–38:13

Leibniz International Proceedings in Informatics



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

Arithmetic computations are also considered in the noncommutative setting. The free noncommutative ring $\mathbb{F}\langle y_1, y_2, \dots, y_n \rangle$ is usually denoted by $\mathbb{F}\langle Y \rangle^1$. In the ring $\mathbb{F}\langle Y \rangle$, monomials are words in Y^* and polynomials in $\mathbb{F}\langle Y \rangle$ are \mathbb{F} -linear combinations of words. We define noncommutative arithmetic circuits essentially as their commutative counterparts. The only difference is that at each product gate in a noncommutative circuit there is a prescribed left to right ordering of its inputs.

A more restricted model than arithmetic circuits are algebraic branching programs. An *algebraic branching program* (ABP) is a directed acyclic graph with one in-degree-0 vertex called *source*, and one out-degree-0 vertex called *sink*. The vertex set of the graph is partitioned into layers $0, 1, \dots, \ell$, with directed edges only between adjacent layers (i to $i + 1$). The source and the sink are at layers zero and ℓ respectively. Each edge is labeled by a linear form over variables x_1, x_2, \dots, x_n . The polynomial computed by the ABP is the sum over all source-to-sink directed paths of the product of linear forms that label the edges of the path. An ABP is *homogeneous* if all edge labels are homogeneous linear forms. ABPs can be defined in both commutative and noncommutative settings.

The main purpose of the current paper is to present new arithmetic complexity upper bound results, in the form of “optimal” algebraic branching programs, for some important polynomials in both the commutative and noncommutative domains. These results are motivated by our recent work on an algebraic approach to designing efficient parameterized algorithms for various combinatorial problems [1].

We now proceed to define the polynomials and explain the results obtained.

The Elementary Symmetric Polynomial

We first recall the definition of k^{th} elementary symmetric polynomial $S_{n,k} \in \mathbb{F}[X]$, over the n variables $X = \{x_1, x_2, \dots, x_n\}$,

$$S_{n,k}(X) = \sum_{S \subseteq [n]: |S|=k} \prod_{i \in S} x_i.$$

It is well-known that $S_{n,k}(X)$ can be computed by an algebraic branching program of size $O(nk)$. In this paper, we consider the noncommutative symmetrized version $S_{n,k}^*$, in the ring $\mathbb{F}\langle Y \rangle$, defined as:

$$S_{n,k}^*(Y) = \sum_{T \subseteq [n]: |T|=k} \sum_{\sigma \in S_k} \prod_{i \in T} y_{\sigma(i)}.$$

The complexity of the polynomial $S_{n,k}^*$ is first considered by Nisan in his seminal work in noncommutative computation [9]. Nisan shows that any ABP for $S_{n,k}^*$ is of size $\Omega\left(\binom{n}{\lfloor k/2 \rfloor}\right)^2$. Furthermore, Nisan also shows the *existence* of ABP of size $O\left(\binom{n}{\lfloor k/2 \rfloor}\right)$ for $S_{n,k}^*$. However, it is not clear how to construct such an ABP in time $O\left(\binom{n}{\lfloor k/2 \rfloor}\right)$. Note that an ABP of size $O^*(n^k)$ for $S_{n,k}^*$ can be directly constructed in $O^*(n^k)$ time by opening up the expression completely³. The main upper bound question is whether we can achieve any constant factor saving of the parameter k in terms of size and run time of the construction. In this paper, we give such an explicit construction. Note that Nisan’s result also rules out any FPT(k)-size ABP for $S_{n,k}^*$. That also justifies the problem from an *exact computation* point of view.

¹ Throughout the paper, we use X to denote the set of commuting variables and Y, Z to denote the set of noncommuting variables.

² We use $\binom{n}{\lfloor r \rfloor}$ to denote $\sum_{i=0}^r \binom{n}{i}$.

³ In this paper we use the notation $O^*(\cdot)$ freely to suppress the terms asymptotically smaller than the main term.

Rectangular Permanent and Rectangular Determinant Polynomial

The next polynomial of interest in the current paper is rectangular permanent polynomial. Given a $k \times n$ rectangular matrix $X = (x_{i,j})_{1 \leq i \leq k, 1 \leq j \leq n}$ of commuting variables and a $k \times n$ rectangular matrix $Y = (y_{i,j})_{1 \leq i \leq k, 1 \leq j \leq n}$ of noncommuting variables, the rectangular permanent polynomial in commutative and noncommutative domains are defined as follows

$$\text{rPer}(X) = \sum_{\sigma \in I_{k,n}} \prod_{i=1}^k x_{i,\sigma(i)}, \quad \text{rPer}(Y) = \sum_{\sigma \in I_{k,n}} \prod_{i=1}^k y_{i,\sigma(i)}.$$

Here, $I_{k,n}$ is the set of all injections from $[k] \rightarrow [n]$. An alternative view is that $\text{rPer}(X) = \sum_{S \subset [n]: |S|=k} \text{Per}(X_S)$ where X_S is the $k \times k$ submatrix where the columns are indexed by the set S . Of course, such a polynomial can be computed in time $O^*(n^k)$ using a circuit of similar size, the main interesting issue is to understand whether the dependence on the parameter k can be improved. It is implicit in the work of Vassilevska and Williams [10] that the $\text{rPer}(X)$ polynomial in the commutative setting can be computed by an algebraic branching program of size $O^*(2^k)$. This problem originates from its connection with combinatorial problems studied in the context of exact algorithm design [10]. In the noncommutative setting, set-multilinearizing $S_{n,k}^*(Y)$ polynomial (i.e. replacing each y_i at position j by $y_{j,i}$), we obtain $\text{rPer}(Y)$ where Y is a $k \times n$ symbolic matrix of noncommuting variables. Using this connection with the explicit construction of $S_{n,k}^*(Y)$ polynomial, we provide an ABP for $\text{rPer}(Y)$ in the *noncommutative setting* of size $O^*\left(\binom{n}{\lfloor k/2 \rfloor}\right)$. The construction time is also similar.

As in the usual commutative case, the noncommutative determinant polynomial of a symbolic matrix $Y = (y_{i,j})_{1 \leq i,j \leq k}$ is defined as follows (the variables in the monomials are ordered from left to right):

$$\text{Det}(Y) = \sum_{\sigma \in S_k} \text{sgn}(\sigma) y_{1,\sigma(1)} \cdots y_{k,\sigma(k)}.$$

Nisan [9] has also shown that any algebraic branching program for the *noncommutative determinant* of a $k \times k$ symbolic matrix must be of size $\Omega(2^k)$. In this paper we give an explicit construction of such an ABP in time $O^*(2^k)$. Here too, the main point is that Nisan has also shown that the lower bound is tight, but we provide an explicit construction.

Moreover, motivated by the result of Vassilevska and Williams [10], we study the complexity of the rectangular determinant polynomial (in commutative domain) defined as follows.

$$\text{rDet}(X) = \sum_{S \in \binom{[n]}{k}} \text{Det}(X_S).$$

We prove that the rectangular determinant polynomial can be computed using $O^*(2^k)$ -size explicit ABP.

Finally, we consider the problem of evaluating the noncommutative rectangular determinant over matrix algebras and show that it is $\#W[1]$ -hard for polynomial dimensional matrices. Hence the noncommutative rectangular determinant is unlikely to have an explicit $O^*(n^{o(k)})$ -size ABP. Recently, we have shown the $\#W[1]$ -hardness of computing noncommutative rectangular permanents over poly-dimensional rational matrices [1]. We note that the noncommutative $n \times n$ determinant over matrix algebras is well-studied, and computing it remains $\#P$ -hard even over 2×2 rational matrices [3, 7, 6]. Our proof technique is based on

Hadamard product of noncommutative polynomials which is also used in [3]. However, the crucial difference is that, to show the #P-hardness of noncommutative determinant, authors in [3] reduce the evaluation of commutative permanent to this case; whereas, #W[1]-the hardness of noncommutative rectangular determinant seems more challenging as commutative rectangular permanent is in FPT. In contrast, we show that the rectangular determinant (and rectangular permanent), whose entries are $r \times r$ matrices over any field, can be computed in time $O^*(2^k r^{2k})$.

Our Results

We first formally define what we mean by *explicit* circuit upper bounds.

► **Definition 1** (Explicit Circuit Upper Bound). *A family $\{f_n\}_{n>0}$ of degree- k polynomials in the commutative ring $\mathbb{F}[x_1, x_2, \dots, x_n]$ (or the noncommutative ring $\mathbb{F}\langle y_1, y_2, \dots, y_n \rangle$) has $q(n, k)$ -explicit upper bounds if there is an $O^*(q(n, k))$ time-bounded algorithm \mathcal{A} that on input $\langle 0^n, k \rangle$ outputs a circuit C_n of size $O^*(q(n, k))$ computing f_n .*

We show the following explicit upper bound results.

► **Theorem 2.**

1. *The family of symmetrized elementary polynomials $\{S_{n,k}^*(Y)\}_{n>0}$ has $\binom{n}{\lfloor k/2 \rfloor}$ -explicit ABPs over any field.*
2. *The noncommutative rectangular permanent family $\{\text{rPer}(Y)\}_{n>0}$, where Y is a $k \times n$ symbolic matrix of variables has $\binom{n}{\lfloor k/2 \rfloor}$ -explicit ABPs.*

► **Remark 3.** We note here that there is an algorithm of run time $O^*(\binom{n}{\lfloor k/2 \rfloor})$ for computing the rectangular permanent over rings and semirings [5]. Our contribution in Theorem 2.2 is that we obtain an $\binom{n}{\lfloor k/2 \rfloor}$ -explicit ABP for it.

► **Theorem 4.**

1. *The family of noncommutative determinants $\{\text{Det}(Y)\}_{k>0}$ has 2^k -explicit ABPs over any field.*
2. *There is a family $\{f_n\}$ of noncommutative degree- k polynomials f_n such that f_n has the same support as $S_{n,k}^*$, and it has 2^k -explicit ABPs. This result holds over any field that has at least n distinct elements.*
3. *The commutative rectangular determinant family $\{\text{rDet}(X)\}_{k>0}$, where X is a $k \times n$ matrix of variables has 2^k -explicit ABPs.*

We stress here that the constructive aspect of the above upper bounds is new. The *existence* of the ABPs claimed in the first two parts of Theorem 2 and the first part of Theorem 4 follows from Nisan's work [9] which shows a tight connection between optimal ABP-size for some $f \in \mathbb{F}\langle X \rangle$ and ranks of the matrices M_r whose rows are labeled by degree r monomials, columns by degree $k - r$ monomials and the $(m_1, m_2)^{\text{th}}$ entry is the coefficient of $m_1 m_2$ in f .

Next we describe the parameterized hardness result for rectangular determinant polynomial when we evaluate over matrix algebras.

► **Theorem 5.** *For any fixed $\epsilon > 0$, evaluating the $k \times n$ rectangular determinant polynomial over $n^\epsilon \times n^\epsilon$ rational matrices is #W[1]-hard, treating k as fixed parameter.*

However, we can easily design an algorithm of run time $O^*(2^k r^{2k})$ for computing the rectangular permanent and determinant polynomials with $r \times r$ matrix entries over any field.

Organization

The paper is organized as follows. In Section 2, we provide the necessary background. The proofs of Theorem 2 and Theorem 4 are given in Section 3 and Section 4 respectively. We prove Theorem 5 in Section 5.

2 Preliminaries

We provide some background results from noncommutative computation. Given a commutative circuit C , we can naturally associate a noncommutative circuit C^{nc} by prescribing an input order at each multiplication gate. This is captured in the following definition.

► **Definition 6.** *Given a commutative circuit C computing a polynomial in $\mathbb{F}[x_1, x_2, \dots, x_n]$, the noncommutative version of C , C^{nc} is the noncommutative circuit obtained from C by fixing an ordering of the inputs to each product gate in C and replacing x_i by the noncommuting variable $y_i : 1 \leq i \leq n$.*

Let $f \in \mathbb{F}[X]$ be a homogenous degree- k polynomial computed by a circuit C , and let $\hat{f}(Y) \in \mathbb{F}\langle Y \rangle$ be the polynomial computed by C^{nc} . Let X_k denote the set of all degree- k monomials over X . As usual, Y^k denotes all degree- k noncommutative monomials (i.e., words) over Y . Each monomial $m \in X_k$ can appear as different noncommutative monomials \hat{m} in \hat{f} . We use the notation $\hat{m} \rightarrow m$ to denote that $\hat{m} \in Y^k$ will be transformed to $m \in X_k$ by substituting x_i for $y_i, 1 \leq i \leq n$. Then, we observe the following, $[m]f = \sum_{\hat{m} \rightarrow m} [\hat{m}] \hat{f}$.

For each monomial $\hat{m} = y_{i_1} y_{i_2} \cdots y_{i_k}$, the permutation $\sigma \in S_k$ maps \hat{m} to the monomial \hat{m}^σ defined as $\hat{m}^\sigma = y_{i_{\sigma(1)}} y_{i_{\sigma(2)}} \cdots y_{i_{\sigma(k)}}$. By linearity, $\hat{f} = \sum_{\hat{m} \in Y^k} [\hat{m}] \hat{f} \cdot \hat{m}$ is mapped by σ to the polynomial, $\hat{f}^\sigma = \sum_{\hat{m} \in Y^k} [\hat{m}] \hat{f} \cdot \hat{m}^\sigma$. This gives the following definition.

► **Definition 7.** *The symmetrized polynomial of f , f^* is degree- k homogeneous polynomial $f^* = \sum_{\sigma \in S_k} \hat{f}^\sigma$.*

Next, we recall the definition of Hadamard product of two polynomials.

► **Definition 8.** *Given polynomials f, g , their Hadamard product is defined as*

$$f \circ g = \sum_m ([m]f \cdot [m]g) \cdot m,$$

where $[m]f$ denotes the coefficient of monomial m in f .

In the commutative setting, computing the Hadamard product is intractable in general. This is readily seen as the Hadamard product of the determinant polynomial with itself yields the permanent polynomial. However, in the noncommutative setting the Hadamard product of two ABPs can be computed efficiently [2].

► **Theorem 9 ([2]).** *Given a noncommutative ABP of size S' for degree k polynomial $f \in \mathbb{F}\langle y_1, y_2, \dots, y_n \rangle$ and a noncommutative ABP of size S for another degree k polynomial $g \in \mathbb{F}\langle y_1, y_2, \dots, y_n \rangle$, we can compute a noncommutative ABP of size SS' for $f \circ g$ in deterministic $SS' \cdot \text{poly}(n, k)$ time.*

Let C be a circuit and B an ABP computing homogeneous degree- k polynomials $f, g \in \mathbb{F}\langle Y \rangle$ respectively. Then their Hadamard product $f \circ g$ has a noncommutative circuit of polynomially bounded size which can be computed efficiently [2].

Furthermore, if C is given by black-box access then $f \circ g(a_1, a_2, \dots, a_n)$ for $a_i \in \mathbb{F}, 1 \leq i \leq n$ can be evaluated by evaluating C on matrices defined by the ABP B [3] as follows: For each $i \in [n]$, the transition matrix $M_i \in M_s(\mathbb{F})$ are computed from the noncommutative

38:6 On Explicit Branching Programs

ABP B (which is of size s) that encode layers. We define $M_i[k, \ell] = [x_i]L_{k, \ell}$, where $L_{k, \ell}$ is the linear form on the edge (k, ℓ) . Now to compute $(f \circ g)(a_1, a_2, \dots, a_n)$ where $a_i \in \mathbb{F}$ for each $1 \leq i \leq n$, we compute $C(a_1M_1, a_2M_2, \dots, a_nM_n)$. The value $(f \circ g)(a_1, a_2, \dots, a_n)$ is the $(1, s)^{th}$ entry of the matrix $f(a_1M_1, a_2M_2, \dots, a_nM_n)$.

► **Lemma 10** ([3]). *Given a circuit C and a ABP B computing homogeneous noncommutative polynomials f and g in $\mathbb{F}\langle Y \rangle$, the Hadamard product $f \circ g$ can be evaluated at any point $(a_1, \dots, a_n) \in \mathbb{F}^n$ by evaluating $C(a_1M_1, \dots, a_nM_n)$ where M_1, \dots, M_n are the transition matrices of B , and the dimension of each M_i is the size of B .*

3 The Proof of Theorem 2

In this section, we present the construction of explicit ABPs for $S_{n,k}^*(Y)$ and noncommutative $\text{rPer}(Y)$.

3.1 The construction of ABP for $S_{n,k}^*(Y)$

The construction of the ABP for $S_{n,k}^*(Y)$ is inspired by a inclusion-exclusion based dynamic programming algorithm for the disjoint sum problem [4].

Proof of Theorem 2.1. Let us denote by F the family of subsets of $[n]$ of size exactly $k/2$. Let $\downarrow F$ denote the family of subsets of $[n]$ of size at most $k/2$. For a subset $S \subset [n]$, we define $m_S = \prod_{j \in S} y_j$. Let us define

$$f_S = \sum_{\sigma \in S_{k/2}} \prod_{j=1}^{k/2} y_{i_{\sigma(j)}}$$

where $S \in F$ and $S = \{i_1, i_2, \dots, i_{k/2}\}$, otherwise for subsets $S \notin F$, we define $f_S = 0$. Note that, for each $S \in F$, f_S is the symmetrization of the monomial m_S which we denote by m_S^* (notice Definition 7).

For each $S \in \downarrow F$, let us define $\hat{f}_S = \sum_{S \subseteq A} f_A$ where $A \in F$. We now show, using the inclusion-exclusion principle, that we can express $S_{n,k}^*$ using an appropriate combination of these symmetrized polynomials for different subsets.

► **Lemma 11.**

$$S_{n,k}^* = \sum_{S \in \downarrow F} (-1)^{|S|} \hat{f}_S^2.$$

Proof. Let us first note that, $S_{n,k}^* = \sum_{A \in F} \sum_{B \in F} [A \cap B = \emptyset] f_A f_B$, where we use $[P]$ to denote that the proposition P is true. By the inclusion-exclusion principle:

$$\begin{aligned} S_{n,k}^* &= \sum_{A \in F} \sum_{B \in F} [A \cap B = \emptyset] f_A f_B \\ &= \sum_{A \in F} \sum_{B \in F} \sum_{S \in \downarrow F} (-1)^{|S|} [S \subseteq A \cap B] f_A f_B \\ &= \sum_{S \in \downarrow F} (-1)^{|S|} \sum_{A \in F} \sum_{B \in F} [S \subseteq A] [S \subseteq B] f_A f_B \\ &= \sum_{S \in \downarrow F} (-1)^{|S|} \left(\sum_{A \in F} [S \subseteq A] f_A \right)^2 = \sum_{S \in \downarrow F} (-1)^{|S|} \hat{f}_S^2. \quad \blacktriangleleft \end{aligned}$$

Now we describe two ABPs where the first ABP simultaneously computes f_A for each $A \in F$ and the second one simultaneously computes \hat{f}_S for each $S \in \downarrow F$.

► **Lemma 12.** *There is an $\binom{n}{\downarrow k/2}$ -explicit multi-output ABP B_1 that outputs the collection $\{f_A\}$ for each $A \in F$.*

Proof. First note that, $m_S^* = \sum_{j \in S} m_{S \setminus \{j\}}^* \cdot y_j$. Now, the construction of the ABP is obvious. It consists of $(k/2 + 1)$ layers where layer $\ell \in \{0, 1, \dots, k/2\}$ has $\binom{n}{\ell}$ many nodes indexed by ℓ size subsets of $[n]$. In $(\ell + 1)^{th}$ layer, the node indexed by S is connected to the nodes $S \setminus \{j\}$ in the previous layer with an edge label y_j for each $j \in S$. Clearly, in the last layer, the S^{th} sink node computes f_S . ◀

► **Lemma 13.** *There is an $\binom{n}{\downarrow k/2}$ -explicit multi-output ABP B_2 that outputs the collection $\{\hat{f}_S\}$ for each $S \in \downarrow F$.*

Proof. To construct such an ABP, we use ideas from [4]. We define $\hat{f}_{i,S} = \sum_{S \subseteq A} f_A$ where $S \subseteq A$ and $A \cap [i] = S \cap [i]$. Note that, $\hat{f}_{n,S} = f_S$ and $\hat{f}_{0,S} = \hat{f}_S$. From the definition, it is clear that $\hat{f}_{i-1,S} = \hat{f}_{i,S} + \hat{f}_{i,S \cup \{i\}}$ if $i \notin S$ and $\hat{f}_{i-1,S} = \hat{f}_{i,S}$ if $i \in S$. Hence, we can take a copy of ABP B_1 from Lemma 12, and then simultaneously compute $\hat{f}_{i,S}$ for each $S \in \downarrow F$ and i ranging from n to 0. Clearly, the new ABP B_2 consists of $(n + k/2 + 1)$ many layers and at most $\binom{n}{\downarrow k/2}$ nodes at each layer. The number of edges in the ABP is also linear in the number of nodes. ◀

Let $f = \sum_{m \in Y^k} [m]f \cdot m$ be a noncommutative polynomial of degree k in $\mathbb{F}\langle Y \rangle$. The reverse of f is defined as the polynomial

$$f^R = \sum_{m \in Y^k} [m]f \cdot m^R,$$

where m^R is the reverse of the word m .

► **Lemma 14** (Reversing an ABP). *Suppose B is a multi-output ABP with r sink nodes where the i^{th} sink node computes $f_i \in \mathbb{F}\langle Y \rangle$ for each $i \in [r]$. We can construct an ABP of twice the size of B that computes the polynomial $\sum_{i=1}^r f_i \cdot L_i \cdot f_i^R$ where L_i are affine linear forms.*

Proof. Suppose B has ℓ layers, then we construct an ABP of $2\ell + 1$ layers where the first ℓ layers are the copy of ABP B and the last ℓ layers are the “mirror image” of the ABP B , call it B^R . In the $(\ell + 1)^{th}$ layer we connect the i th sink node of ABP B to the i th source node of B^R by an edge with edge label L_i . Note that, B^R has r source nodes and one sink node and the polynomial computed between i th source node and sink is f_i^R . ◀

Now, applying the construction of Lemma 14 to the multi-output ABP B_2 of Lemma 13 with $L_S = (-1)^{|S|}$ we obtain an ABP that computes the polynomial $\sum_S (-1)^{|S|} \hat{f}_S \cdot \hat{f}_S^R$. Since \hat{f}_S is a symmetrized polynomial, we note that $\hat{f}_S^R = \hat{f}_S$ and using Lemma 11 we conclude that this ABP computes $S_{n,k}^*$. The ABP size is $O(k \binom{n}{\downarrow k/2})$. ◀

3.2 The construction of ABP for rPer(Y)

Proof of Theorem 2.2. A $\binom{n}{\downarrow k/2}$ -explicit ABP for the rectangular permanent polynomial can be obtained easily from the $\binom{n}{\downarrow k/2}$ -explicit ABP for $S_{n,k}^*(Y)$ by careful set-multilinearization. This can be done by simply renaming the variables $y_i : 1 \leq i \leq n$ at the position $1 \leq j \leq k$ by $y_{j,i}$. ◀

4 The Proof of Theorem 4

We divide the proof in three subsections.

4.1 A 2^k -explicit ABP for $k \times k$ noncommutative determinant

In this section, we present an optimal explicit ABP construction for the noncommutative determinant polynomial for the square symbolic matrix. .

Proof of Theorem 4.1. The ABP B has $k + 1$ layers with $\binom{k}{\ell}$ nodes at the layer ℓ for each $0 \leq \ell \leq k$. The source of the ABP is labeled \emptyset and the nodes in layer ℓ are labeled by the distinct size ℓ subsets $S \subseteq [k]$, $1 \leq \ell \leq k$, hence the sink is labeled $[k]$. From the node labeled S in layer ℓ , there are $k - \ell$ outgoing edges $(S, S \cup \{j\})$, $j \in [k] \setminus S$.

Define the sign $\text{sgn}(S, j)$ as $\text{sgn}(S, j) = (-1)^{t_j}$, where t_j is the number of elements in S larger than j . Equivalently, t_j is the number of swaps required to insert j in the correct position, treating S as a sorted list.

For noncommutative determinant polynomial, we connect the set S in the i^{th} layer to a set $S \cup \{j\}$ in the $(i + 1)^{\text{th}}$ layer with the edge label $\text{sgn}(S, j) \cdot y_{i+1, j}$. The source to sink paths in this ABP are in 1-1 correspondence to the node labels on the paths which give subset chains $\emptyset \subset T_1 \subset T_2 \subset \dots \subset T_k = [k]$ such that $|T_i \setminus T_{i-1}| = 1$ for all $i \leq k$. Such subset chains are clearly in 1-1 correspondence with permutations $\sigma \in S_k$ listed as a sequence: $\sigma(1), \sigma(2), \dots, \sigma(k)$, where $T_i = \{\sigma(1), \sigma(2), \dots, \sigma(i)\}$. The following claim spells out the connection between the sign $\text{sgn}(\sigma)$ of σ and the $\text{sgn}(S, j)$ function defined above.

▷ **Claim 15.** For each $\sigma \in S_k$ and $T_i = \{\sigma(1), \sigma(2), \dots, \sigma(i)\}$, we have

$$\text{sgn}(\sigma) = \prod_{i=1}^k \text{sgn}(T_{i-1}, \sigma(i)).$$

Proof. We first note that $\text{sgn}(\sigma) = (-1)^t$, if there are t transpositions $(r_i \ s_i)$, $1 \leq i \leq t$ such that $\sigma \cdot (r_1 \ s_1) \cdot (r_2 \ s_2) \cdot \dots \cdot (r_t \ s_t) = 1$. Equivalently, interpreting this as sorting the list $\sigma(1), \sigma(2), \dots, \sigma(k)$ by swaps $(r_i \ s_i)$, applying these t swaps will sort the list into $1, 2, \dots, k$. As already noted, $\text{sgn}(T_{i-1}, \sigma(i)) = (-1)^{t_i}$, where t_i is the number of swaps required to insert $\sigma(i)$ in the correct position into the sorted order of T_{i-1} (where $\sigma(i)$ is initially placed to the right of T_{i-1}). Hence, $\sum_{i=1}^k t_i$ is the total number of swaps required for this insertion sort procedure to sort $\sigma(1), \sigma(2), \dots, \sigma(k)$. It follows that $\prod_{i=1}^k \text{sgn}(T_{i-1}, \sigma(i)) = (-1)^{\sum_{i=1}^k t_i} = \text{sgn}(\sigma)$, which proves the claim. ◁

The fact that the ABP computes the noncommutative determinant polynomial follows directly from Claim 15 and the edge labels. ◀

4.2 A 2^k -explicit ABP weakly equivalent to $S_{n,k}^*$

A polynomial $f \in \mathbb{F}[X]$ (resp. $\mathbb{F}\langle Y \rangle$) is said to be *weakly equivalent* to a polynomial $g \in \mathbb{F}[X]$ (resp. $\mathbb{F}\langle Y \rangle$), if for each monomial m over X , $[m]f = 0$ if and only if $[m]g = 0$. For the construction of an ABP computing a polynomial weakly equivalent to $S_{n,k}^*$, we will suitably modify the ABP construction described above.

Proof of Theorem 4.2. Let α_i , $1 \leq i \leq n$ be distinct elements from \mathbb{F} . For each $j \in [k] \setminus S$, the edge $(S, S \cup \{j\})$ is labeled by the linear form $\text{sgn}(S, j) \cdot \sum_{i=1}^n \alpha_i^j y_i$, where y_i , $1 \leq i \leq n$ are noncommuting variables. This gives an ABP B of size $O^*(2^k)$.

We show that the polynomial computed by ABP B is weakly equivalent to $S_{n,k}^*$. Clearly, B computes a homogeneous degree k polynomial in the variables $y_i, 1 \leq i \leq n$. We determine the coefficient of a monomial $y_{i_1}y_{i_2} \cdots y_{i_k}$. As noted, each source to sink path in B corresponds to a permutation $\sigma \in S_k$. Along that path the ABP compute the product of linear forms

$$\text{sgn}(\sigma)L_{\sigma(1)}L_{\sigma(2)} \cdots L_{\sigma(k)}, \text{ where } L_{\sigma(q)} = \sum_{i=1}^n \alpha_i^{\sigma(q)} y_i,$$

where the sign is given by the previous claim. The coefficient of monomial $y_{i_1}y_{i_2} \cdots y_{i_k}$ in the above product is given by $\text{sgn}(\sigma) \prod_{q=1}^k \alpha_{i_q}^{\sigma(q)}$. Thus, the coefficient of $y_{i_1}y_{i_2} \cdots y_{i_k}$ in the ABP is given by $\sum_{\sigma \in S_k} \text{sgn}(\sigma) \prod_{q=1}^k \alpha_{i_q}^{\sigma(q)}$, which is the determinant of the $k \times k$ Vandermonde matrix whose q^{th} column is $(\alpha_{i_q}, \alpha_{i_q}^2, \dots, \alpha_{i_q}^k)^T$. Clearly, that determinant is non-zero if and only if the monomial $y_{i_1}y_{i_2} \cdots y_{i_k}$ is multilinear. Clearly the proof works for any field that contains at least n distinct elements. ◀

▶ **Remark 16.** A polynomial $f \in \mathbb{F}(Y)$ is *positively weakly equivalent* to $S_{n,k}^*$, if for each multilinear monomial $m \in Y^k$, $[m]f > 0$. In the above proof, let g be the polynomial computed by ABP B that is weakly equivalent to $S_{n,k}^*$. Clearly, $f = g \circ g$ is positively weakly equivalent to $S_{n,k}^*$, and f has a 4^k -explicit ABP, since B is 2^k -explicit. This follows from Theorem 9. We leave open the problem of finding a 2^k -explicit ABP for some polynomial that is positively weakly equivalent to $S_{n,k}^*$. Such an explicit construction would imply a deterministic $O^*(2^k)$ time algorithm for k -path which is a long-standing open problem [8].

4.3 A 2^k -explicit ABP for $k \times n$ commutative rectangular determinant

In this section, we present the ABP construction for commutative determinant polynomial for $k \times n$ symbolic matrix.

Proof of Theorem 4.3. We adapt the ABP presented in Subsection 4.1. The main difference is that, for the edge $(S, S \cup \{j\})$, the linear form is $\text{sgn}(S, j) \cdot (\sum_{i=1}^n x_{j,i} z_i)$, where $z_i : 1 \leq i \leq n$ are fresh noncommuting variables, and the $x_{j,i} : 1 \leq j \leq k, 1 \leq i \leq n$ are commuting variables.

Then with a similar argument as before, the coefficient of the monomial $z_{i_1}z_{i_2} \cdots z_{i_k}$ where $i_1 < i_2 < \dots < i_k$ is given by $\sum_{\sigma \in S_k} \text{sgn}(\sigma) x_{\sigma(1), i_1} \cdots x_{\sigma(k), i_k}$. Now for a fixed $\sigma \in S_k$, let τ^σ be the injection $[k] \rightarrow [n]$ such that $\tau^\sigma(j) = i_{\sigma^{-1}(j)} : 1 \leq j \leq k$.

Let (j_1, j_2) be an index pair that is an inversion in σ , i.e. $j_1 < j_2$ and $\sigma(j_1) > \sigma(j_2)$. Let $\ell_1 = \sigma(j_1)$ and $\ell_2 = \sigma(j_2)$. So $i_{\tau^\sigma(\ell_1)} = i_{\sigma^{-1}(\ell_1)}$ and $i_{\tau^\sigma(\ell_2)} = i_{\sigma^{-1}(\ell_2)}$. Clearly, $i_{\tau^\sigma(\ell_1)} < i_{\tau^\sigma(\ell_2)}$. Hence:

$$\sum_{\sigma \in S_k} \text{sgn}(\sigma) x_{\sigma(1), i_1} \cdots x_{\sigma(k), i_k} = \sum_{\tau^\sigma \in I_{k,n}} \text{sgn}(\tau^\sigma) x_{1, \tau^\sigma(1)} \cdots x_{k, \tau^\sigma(k)}.$$

Now the idea is to filter out only the good monomials $z_{i_1}z_{i_2} \cdots z_{i_k}$ where $i_1 < i_2 < \dots < i_k$ from among all the monomials. This can be done by taking Hadamard product (using Theorem 9) with the following polynomial,

$$S_{n,k}^{nc}(Z) = \sum_{S=\{i_1 < i_2 < \dots < i_k\}} z_{i_1} z_{i_2} \cdots z_{i_k}.$$

Clearly, $S_{n,k}^{nc}$ has a $\text{poly}(n, k)$ -sized ABP which is just the noncommutative version (see Definition 6) of the well-known ABP for commutative $S_{n,k}$. Finally, we substitute each $z_i = 1$ to get the desired ABP for $\text{rDet}(X)$. ◀

5 Hardness of Evaluating Rectangular Determinant Over Matrix Algebras

In this section we prove a hardness result for evaluating the rectangular determinant over matrix algebras. More precisely, if A is a $k \times n$ matrix whose entries A_{ij} are $n^\epsilon \times n^\epsilon$ rational matrices for a fixed $\epsilon > 0$, then it is $\#W[1]$ -hard to compute $\text{rDet}(A)$. We show this by a reduction from the $\#W[1]$ -complete problem of counting the number of simple k -paths in directed graphs.

However, there is a simple algorithm of run time $O^*(2^k r^{2k})$ to evaluate rectangular permanent or rectangular determinant of size $k \times n$ over matrix algebras of dimension r . The proof is given in the appendix.

For the proof of Theorem 5, we also use the notion of *Graph Polynomial*. Let $G(V, E)$ be a directed graph with n vertices where $V(G) = \{v_1, v_2, \dots, v_n\}$. A k -walk is a sequence of k vertices $v_{i_1}, v_{i_2}, \dots, v_{i_k}$ where $(v_{i_j}, v_{i_{j+1}}) \in E$ for each $1 \leq j \leq k-1$. A k -path is a k -walk where no vertex is repeated. Let A be the adjacency matrix of G , and let z_1, z_2, \dots, z_n be noncommuting variables. Define an $n \times n$ matrix B

$$B[i, j] = A[i, j] \cdot z_i, \quad 1 \leq i, j \leq n.$$

Let $\vec{1}$ denote the all 1's vector of length n . Let \vec{z} be the length n vector defined by $\vec{z}[i] = z_i$. The *graph polynomial* $C_G \in \mathbb{F}\langle Z \rangle$ is defined as

$$C_G(z_1, z_2, \dots, z_n) = \vec{1}^T \cdot B^{k-1} \cdot \vec{z}.$$

Let W be the set of all k -walks in G . The following observation is folklore.

► **Observation 1.**

$$C_G(z_1, z_2, \dots, z_n) = \sum_{v_{i_1} v_{i_2} \dots v_{i_k} \in W} z_{i_1} z_{i_2} \dots z_{i_k}.$$

Hence, G contains a k -path if and only if the graph polynomial C_G contains a multilinear term.

5.1 The Proof of Theorem 5

Let $I_{k,n}$ be the set of injections from $[k] \rightarrow [n]$. Define

$$S := \{f \in I_{2k, 2n} \mid \exists g \in I_{k,n} \text{ such that } \forall i \in [k], f(2i-1) = g(i); f(2i) = n + g(i)\}.$$

Clearly, there is a bijection between S and $I_{k,n}$. We denote each $f \in S$ as f_g where $g \in I_{k,n}$ is the corresponding injection. By a simple counting argument, we observe the following.

► **Observation 2.** For each $f \in S$, $\text{sgn}(f) = (-1)^{\frac{k(k-1)}{2}}$.

Consider a set of noncommuting variables $Y = \{y_{1,1}, y_{1,2}, \dots, y_{2k, 2n}\}$ corresponding to the entries of a $2k \times 2n$ symbolic matrix Y . Given $f \in I_{2k, 2n}$, define $m_f = \prod_{i=1}^{2k} y_{i, f(i)}$.

► **Lemma 17.** There is an ABP B of $\text{poly}(n, k)$ size that computes a polynomial $F \in \mathbb{F}\langle Y \rangle$ such that for each $f \in I_{2k, 2n}$, $[m_f]F = 1$ if $f \in S$ and otherwise $[m_f]F = 0$.

Proof. The ABP B consists of $2k + 1$ layers, labelled $\{0, 1, \dots, 2k\}$. For each even $i \in [0, 2k]$, there is exactly one node q_i at level i . For each odd $i \in [0, 2k]$, there are n nodes $p_{i,1}, p_{i,2}, \dots, p_{i,n}$ at level i . We now describe the edges of B . For each even $i \in [0, 2k - 2]$ and $j \in [n]$, there is an edge from q_i to $p_{i+1,j}$ labelled $y_{i+1,j}$. For each odd $i \in [0, 2k - 1]$ and $j \in [n]$, there is an edge from $p_{i,j}$ to q_{i+1} labelled $y_{i+1,n+j}$. For an injection $f \in I_{2k,2n}$, B contributes a monomial m_f if and only if $f \in S$ and B can be computed in $\text{poly}(n, k)$ time. \blacktriangleleft

Suppose, Y is a $2k \times 2n$ matrix where the $(i, j)^{\text{th}}$ entry is $y_{i,j}$. By Observation 2 and Lemma 17,

$$\text{rDet}(Y) \circ F(Y) = \sum_{f_g \in S} \text{sgn}(f_g) m_{f_g} = (-1)^{\frac{k(k-1)}{2}} \sum_{g \in I_{k,n}} m_{f_g}.$$

Let $Z = \{z_1, \dots, z_n\}$ be a set of noncommuting variables. Define for each $g \in I_{k,n}$, $m'_g = \prod_{i=1}^k z_{g(i)}$. Define a map τ such that $\tau : y_{i,j} \mapsto z_j$ if i is odd, and $\tau : y_{i,j} \mapsto 1$ for even i . In other words, $\tau(m_{f_g}) = m'_g$. Notice that,

$$\text{rDet}(Y) \circ F(Y)|_{\tau} = (-1)^{\frac{k(k-1)}{2}} \sum_{g \in I_{k,n}} m_{f_g}|_{\tau} = (-1)^{\frac{k(k-1)}{2}} \sum_{g \in I_{k,n}} m'_g = (-1)^{\frac{k(k-1)}{2}} S_{n,k}^*(Z).$$

Given a directed graph G on n vertices, we first construct an ABP for the noncommutative graph polynomial C_G over rationals. From the definition, it follows that C_G has a polynomial size ABP. Notice that, $(\text{rDet}(Y) \circ F(Y)|_{\tau}) \circ C_G(Z)(\vec{1}) = S_{n,k}^*(Z) \circ C_G(Z)(\vec{1})$ counts the number of directed k -paths in the graph G , and hence evaluating this term is $\#\text{W}[1]$ -hard. Let us modify the ABP for graph polynomial $C_G(Z)$ by replacing each edge labeled by z_j at i^{th} layer by two edges where the first edge is labeled by $y_{2i-1,j}$ and second one is labeled by $y_{2i,n+j}$. Let $C'_G(Y)$ is the new polynomial computed by the ABP. Notice that, each monomial of the modified graph polynomial looks like $\prod_{i=1}^{2k} y_{i,f(i)}$ for some $f : [2k] \mapsto [2n]$. More importantly, for each k -path $v_{i_1} v_{i_2} \dots v_{i_k}$, if $g \in I_{k,n}$ is the corresponding injection, then $\prod_{i=1}^k z_{g(i)}$ is converted to $\prod_{i=1}^{2k} y_{i,f_g(i)}$ for $f_g \in S$. Notice that, $(\text{rDet}(Y) \circ F(Y)|_{\tau}) \circ C_G(Z) = (\text{rDet}(Y) \circ F(Y) \circ C'_G(Y))|_{\tau}$ and hence, evaluating $(\text{rDet}(Y) \circ F(Y) \circ C'_G(Y))(\vec{1})$ is $\#\text{W}[1]$ -hard.

Now, assume to the contrary, we have an FPT algorithm \mathcal{A} to evaluate $\text{rDet}(Y)$ over matrix inputs. As, $C'_G(Y)$ and $F(Y)$ are computed by ABPs, we obtain an ABP B' computing $C'_G \circ F(Y)$. From ABP B' , we construct the $t \times t$ transition matrices $M_{1,1}, \dots, M_{2k,2n}$ where t is the size of the ABP B' . From Lemma 10 we know that, we are interested to compute $\text{rDet}(Y)$ over the matrix tuple $(M_{1,1}, \dots, M_{2k,2n})$ which is same as invoking the algorithm \mathcal{A} on the following $2k \times 2n$ matrix A : $a_{i,j} = M_{i,j}$. By a simple reduction we get a similar hardness over $n^\epsilon \times n^\epsilon$ dimensional matrix algebras for any fixed $\epsilon > 0$. \blacktriangleleft

6 Conclusion

In this paper, we have presented the construction of explicit algebraic branching programs for noncommutative symmetrized elementary symmetric polynomial, noncommutative rectangular permanent polynomial and commutative rectangular determinant polynomial. Additionally, we present an explicit algebraic branching program for noncommutative square determinant polynomial. In most of the cases the constructions are optimal in the sense of lower bound result of Nisan [9]. It is also shown that evaluating rectangular determinant polynomial over matrix algebras is $\#\text{W}[1]$ -hard. The paper brings out further avenues of research. A very interesting problem is to tightly classify the complexity of computing the commutative rectangular $k \times n$ determinant polynomial. Is it computable in $\text{poly}(n, k)$

time? If not, can one show a complexity theoretic hardness of evaluating the commutative determinant polynomial? We feel that the main obstacle is to interpret rectangular determinant computation combinatorially. Another open end is to construct an explicit algebraic branching program for noncommutative rectangular determinant polynomial of size $O^*(n^{ck})$ for some $c < 1$ similar to the one we have constructed for noncommutative rectangular permanent polynomial.

References

- 1 Vikraman Arvind, Abhranil Chatterjee, Rajit Datta, and Partha Mukhopadhyay. Fast Exact Algorithms Using Hadamard Product of Polynomials. *CoRR*, abs/1807.04496, 2018. [arXiv:1807.04496](#).
- 2 Vikraman Arvind, Pushkar S. Joglekar, and Srikanth Srinivasan. Arithmetic Circuits and the Hadamard Product of Polynomials. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2009, December 15-17, 2009, IIT Kanpur, India*, pages 25–36, 2009. doi:10.4230/LIPIcs.FSTTCS.2009.2304.
- 3 Vikraman Arvind and Srikanth Srinivasan. On the hardness of the noncommutative determinant. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 677–686, 2010. doi:10.1145/1806689.1806782.
- 4 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Counting Paths and Packings in Halves. In Amos Fiat and Peter Sanders, editors, *Algorithms - ESA 2009*, pages 578–586, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
- 5 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Evaluation of permanents in rings and semirings. *Inf. Process. Lett.*, 110(20):867–870, 2010. doi:10.1016/j.ipl.2010.07.005.
- 6 Markus Bläser. Noncommutativity Makes Determinants Hard. In *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, volume 243, pages 172–183, 2013. doi:10.1007/978-3-642-39206-1_15.
- 7 Steve Chien, Prahladh Harsha, Alistair Sinclair, and Srikanth Srinivasan. Almost settling the hardness of noncommutative determinant. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 499–508, 2011. doi:10.1145/1993636.1993703.
- 8 Ioannis Koutis and Ryan Williams. LIMITS and applications of group algebras for parameterized problems. *ACM Trans. Algorithms*, 12(3):31:1–31:18, 2016. doi:10.1145/2885499.
- 9 Noam Nisan. Lower Bounds for Non-Commutative Computation (Extended Abstract). In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*, pages 410–418, 1991. doi:10.1145/103418.103462.
- 10 Virginia Vassilevska Williams and Ryan Williams. Finding, Minimizing, and Counting Weighted Subgraphs. *SIAM J. Comput.*, 42(3):831–854, 2013. doi:10.1137/09076619X.

A Computing Rectangular Permanent and Determinant over Small Dimensional Algebras

The main result of the section is as follows.

► **Theorem 18.** *Let \mathbb{F} be any field and \mathcal{A} be an r dimensional algebra over \mathbb{F} with basis e_1, e_2, \dots, e_r . Let $\{A_{ij}\}_{\substack{1 \leq i \leq k \\ 1 \leq j \leq n}}$ be a $k \times n$ matrix with $A_{ij} \in \mathcal{A}$. Then $\text{rPer}(A)$ and $\text{rDet}(A)$ can be computed in deterministic $O^*(2^{kr^k})$ time.*

Proof. We present the proof for rectangular permanent. The proof for rectangular determinant is identical. The proof follows easily from expressing each entry $A_{i,j}$ in the standard basis and then rearranging terms. Let e_1, e_2, \dots, e_r be the standard basis for \mathcal{A} over \mathbb{F} . First we note that

$$\begin{aligned}
 \text{rPer}(A) &= \sum_{f \in I_{k,n}} \prod_{i=1}^k A_{if(i)} \\
 &= \sum_{f \in I_{k,n}} \prod_{i=1}^k \sum_{\ell=1}^r A_{if(i)}^{(\ell)} e_{\ell} \\
 &= \sum_{f \in I_{k,n}} \sum_{(t_1, t_2, \dots, t_k) \in [r]^k} \prod_{i=1}^k A_{if(i)}^{(t_i)} \prod_{i=1}^k e_{t_i} \\
 &= \sum_{(t_1, t_2, \dots, t_k) \in [r]^k} \left(\sum_{f \in I_{k,n}} \prod_{i=1}^k A_{if(i)}^{(t_i)} \right) \prod_{i=1}^k e_{t_i}. \tag{1}
 \end{aligned}$$

Now we observe that

$$\sum_{f \in I_{k,n}} \prod_{i=1}^k A_{if(i)}^{(t_i)} = \text{rPer}(A^{(t_1, t_2, \dots, t_k)}),$$

where $A^{(t_1, t_2, \dots, t_k)}$ is the $k \times n$ matrix defined as $A_{ij}^{(t_1, t_2, \dots, t_k)} = A_{ij}^{(t_i)}$. Thus we have

$$\text{rPer}(A) = \sum_{(t_1, t_2, \dots, t_k) \in [r]^k} \text{rPer}(A^{(t_1, t_2, \dots, t_k)}) \prod_{i=1}^k e_{t_i}. \tag{2}$$

For a fixed $(t_1, t_2, \dots, t_k) \in [r]^k$ the value $\text{rPer}(A^{(t_1, t_2, \dots, t_k)})$ can be computed in $O^*(2^k)$ time using the rectangular permanent algorithm [10]. Now we can compute $\text{rPer}(A)$ by computing r^k many such rectangular permanents and putting them together according to equation 2. This gives a deterministic $O^*(2^k r^{2k})$ time algorithm for computing $\text{rPer}(A)$. ◀

As a direct corollary we get the following.

► **Corollary 19.** *Let \mathbb{F} be any field and let A be a $k \times n$ matrix with $A_{ij} \in \mathbb{M}_{r \times r}(\mathbb{F})$. Then $\text{rPer}(A)$ and $\text{rDet}(A)$ can be computed in $O^*(2^k r^k)$ time.*

A Competitive Algorithm for Random-Order Stochastic Virtual Circuit Routing

Thắng Nguyễn Kim 

IBISC, Univ Evry, University Paris Saclay, Evry, France
kimthang.nguyen@univ-evry.fr

Abstract

We consider the virtual circuit routing problem in the stochastic model with uniformly random arrival requests. In the problem, a graph is given and requests arrive in a uniform random order. Each request is specified by its connectivity demand and the load of a request on an edge is a random variable with known distribution. The objective is to satisfy the connectivity request demands while maintaining the expected congestion (the maximum edge load) of the underlying network as small as possible.

Despite a large literature on congestion minimization in the deterministic model, not much is known in the stochastic model even in the offline setting. In this paper, we present an $O(\log n / \log \log n)$ -competitive algorithm when optimal routing is sufficiently congested. This ratio matches to the lower bound $\Omega(\log n / \log \log n)$ (assuming some reasonable complexity assumption) in the offline setting. Additionally, we show that, restricting on the offline setting with deterministic loads, our algorithm yields the tight approximation ratio of $\Theta(\log n / \log \log n)$. The algorithm is essentially greedy (without solving LP/rounding) and the simplicity makes it practically appealing.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Approximation Algorithms, Congestion Minimization

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.39

Funding Research supported by the ANR project OATA n° ANR-15-CE40-0015-01.

1 Introduction

Congestion minimization is a fundamental problem for network operations/communication. In the former, there are connectivity requests and serving requests induces loads on network links. The load vector of each request is *deterministically* given. The objective is to satisfy the connectivity demands while maintaining the congestion of the underlying network as small as possible. The problem has been widely studied and several algorithms with performance guarantee have been designed.

In real-world scenarios, given the presence of uncertainty, request loads are rarely deterministic but vary as random variables. Uncertainty may come from different sources due to unexpected events, noise, etc. The uncertainty in the loads represents the main difficulty in designing performant algorithms in such scenarios. In this paper, we take one step closer to real-world situations by considering the congestion minimization in the stochastic model.

Stochastic Virtual Circuit Routing Problem (SVCR). Given a directed graph $G(V, E)$ where $|V| = n$, $|E| = m$ and a set of k requests. A request i (for $1 \leq i \leq k$) is specified by a origin/destination pair (o_i, d_i) and a random variable $X_{i,e}$ whose distribution is known that represents the load of request i on an edge e . Assume that $X_{i,e}$'s are bounded and without loss of generality, $X_{i,e}$'s take values in $[0, 1]$. For each request i , one needs to choose a routing path connecting o_i to d_i . The *expected congestion* of a routing (connecting all requests' pairs) is $\mathbb{E}[\max_e \sum_{i \in T_e} X_{i,e}]$ where T_e is the set of requests whose routing path passes through e . The objective is to minimize the expected congestion.



© Thắng Nguyễn Kim;

licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 39; pp. 39:1–39:12

Leibniz International Proceedings in Informatics



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

In this paper, we consider the SVCR problem in the random-order setting. In the latter, requests are released over time in an *uniformly random* order and at the arrival of a request, one needs to make an irrevocable decision to satisfy the request. The random-order setting is similar to the online one; however, in the former the adversary can choose request parameters but has no influence on the request arrival order (which is uniformly random).

The congestion objective belongs to the class of ℓ_p -norm functions on load vectors. Specifically, the former corresponds to the ℓ_∞ -norm and it is well-known that the ℓ_∞ -norm of a m -vector can be approximated up to a constant factor by the ℓ_p -norm where $p = \log m$. In the SVCR problem, we also consider ℓ_p -norm objective functions on load vectors. Note that when we mention the SVCR problem without stating explicitly the objective, it means that the congestion objective is considered.

Stochastic algorithmic problems are common in real-world situations and have been extensively studied in different domains, including approximation algorithms. There are two classes of algorithms for stochastic problems: *non-adaptive* and *adaptive*. In the former, the decisions have been made up-front and then the realization of the randomness will be revealed. In the latter, the randomness is revealed instantaneously after each decision (so an algorithm can adapt its strategy due to the outcome of random variables observed so far). In virtual circuit routing, non-adaptive solutions are preferable and more suitable than the adaptive ones since the former is usually simpler and easier to implement. In this paper, we are interested in designing non-adaptive solutions for the SVCR problem.

The virtual circuit routing problem has been well understood in the deterministic model. Specifically, in offline setting Raghavan and Thompson [23] gave an $O(\log n / \log \log n)$ -approximation algorithm and in online setting Aspnes et al. [3] provided an $O(\log n)$ -competitive algorithm. The bounds are optimal up to a constant factor. However, not much in term of approximation is known in the stochastic model. A closely related problem to SVCR, the stochastic load balancing problem, has been studied in the offline setting. In the problem, given a set of jobs and machines, one needs to assign jobs to machines such that the (expected) maximum load of the assignment is minimized. Kleinberg et al. [17] first considered this problem and gave a constant approximation for identical machines, i.e., for each job j , the random loads of a job on all machines are identical. Goel and Indyk [11] provided better approximations when the job loads follow some specific distributions, for example Poisson distributions, Exponential distributions. Very recently, Gupta et al. [12] gave a constant approximation for unrelated machines. They also considered the objective of minimizing the ℓ_p -norm of machine loads and showed an $O(p / \log p)$ -approximation algorithm. Their technique is based on a linear program which guarantees a strong lower bound for the stochastic load balancing problem. In their paper, Gupta et al. [12] raised an open question of designing algorithms for the SVCR problem. The main difficulty, which resists to current approaches, is to deal with the correlation of edges loads where different paths may share common edges.

1.1 Our Contribution and Approach

We give a competitive algorithm for the SVCR problem in the random-order setting. Specifically, our algorithm is $O(\log n / \log \log n)$ -competitive if the congestion of the optimal solution is at least 1, i.e., informally, optimal routing is sufficiently congested. Note that even in the offline setting with deterministic loads, the problem is known to be hard to approximate within factor $\Omega(\log n / \log \log n)$ unless all problems in NP have randomized algorithms with running time $n^{\text{poly} \log n}$ [2, 8]. The result shows that in terms of approximation, one can guarantee the quality of the algorithmic solutions for the virtual circuit routing problem even with uncertainty in the request loads. Moreover, our algorithm is essentially greedy which makes it practically appealing and is easy to implement.

In order to design algorithms for the SVCR problem, we study the more general objective of minimizing the ℓ_p -norm of edge loads. We consider the primal-dual technique with configuration LPs [25]. This approach provides a clean way to deal with non-linear objective functions and intuitive constructions of dual variables. Our algorithm is a generalized version of Greedy Restart algorithms introduced by Molinaro [22] in the context of machine load balancing (which can be seen as a special case of the SVCR problem where the network consists of two nodes and parallel edges connecting these two nodes). Informally, for every request the algorithm selects a routing path greedily with respect to some function $\psi_{\kappa,p}$ (defined later) which depends on the current load vector. However, when half of the requests have been considered, the algorithm *restarts* the procedure: it still chooses a routing path greedily with respect to the function $\psi_{\kappa,p}$ but now the function $\psi_{\kappa,p}$ depends on the load vector induced *only* by the second half of the requests. Building on the primal-dual technique with configuration LPs [25] and useful probability inequalities together with insightful observations by Molinaro [22], we prove the competitiveness of the algorithm in the online random-order setting.

Besides, we revisit the classic virtual circuit routing problem in offline setting with deterministic loads (where $X_{i,e}$'s are deterministic values w_i for every e). We show that our algorithm achieves the tight approximation ratio of $\Theta(\log n / \log \log n)$. Remark that our greedy algorithm is simpler than the algorithms by Raghavan and Thompson [23], Srinivasan [24] which are based on LP-rounding techniques or the recent algorithm by Chekuri and Idleman [6] which relies on the notion of multiroute flows [16].

1.2 Further Related Works

In the offline setting, the virtual circuit routing problem is also known under the name of the congestion minimization problem. The latter is a relaxation of the classic edge-disjoint paths problem: given a graph and a collection of source-sink pairs, can the pairs be connected via edge-disjoint paths. For the variant of the congestion minimization problem where $d_i = 1$ and $w_i \equiv 1$ for every $1 \leq i \leq k$, Raghavan and Thompson gave an $O(\log n / \log \log n)$ -approximation algorithm via their influential randomized rounding technique [23]. This ratio is subsequently proved by Chuzhoy et al. [8] to be tight assuming some complexity hypothesis. Srinivasan [24] considered the multipath congestion minimization problem corresponding to the setting where $d_i \geq 1$ and $w_i \equiv 1$ for every $1 \leq i \leq k$. Srinivasan presented an $O(\log n / \log \log n)$ -approximation algorithm by developing a dependent rounding technique for cardinality constraints [24]. The technique is extended in subsequent works for handling more general constraints [10, 9, 7]. Recently, Chekuri and Idleman [6] gave a simple algorithm for the multipath congestion minimization problem. They showed the $O(\log n / \log \log n)$ approximation ratio via the notion of multiroute flows which were originally introduced by Kishimoto and Takeuchi [16]. That enables a simple solution without using dependent rounding and also allows them to improve the results in some particular cases.

The congestion minimization problem has been also studied in online setting where requests arrive online. Aspnes et al. [3] gave an $O(\log n)$ -competitive algorithm and proved that this bound is optimal up to a constant factor. For the more general objective of ℓ_p -norm, Awerbuch et al. [4] considered the load balancing problem and proved that greedy algorithm achieved the bound of $O(p)$, also optimal up to a constant factor. Caragiannis [5] strengthened and significantly simplified the analysis of the greedy algorithm and showed the optimal bound of $\frac{1}{2^{1/p}-1}$.

Stochastic combinatorial optimization problems such as shortest paths, minimum spanning trees, knapsack, bin-packing etc have been considered by Li and Deshpande [18] and Li and Yuan [19] and Kleinberg et al. [17]. In these problems, parameters (length, weights, etc)

are given as random variables with known distributions and the objective is to optimize the expected value of some cost/utility functions. In this paper, we are interested in the class of non-adaptive algorithms. Several works [21, 20, 15, 14] have considered adaptive algorithms where the decisions of algorithms depend on the current state of the solutions.

2 Preliminaries

In this section, we give some definitions and technical lemmas which are useful in our analysis. This part is drawn significantly from Molinaro [22]. Recall that in the random-order model, the cost of a routing is the expected ℓ_p -norm of the load vector where the expectation is taken over the random order and the random vectors $X_{i,e}$'s.

Given $p > 1$, its Hölder conjugate q is the number that satisfies $\frac{1}{p} + \frac{1}{q} = 1$. The dual of the ℓ_p -norm is the ℓ_q -norm. Let ℓ_q^+ be the set of non-negative vectors in \mathbb{R}^m with ℓ_q -norm at most 1. Given a constant κ and p , define function $\psi_{\kappa,p} : \mathbb{R}^m \rightarrow \mathbb{R}$ as $\psi_{\kappa,p}(\mathbf{u}) = \frac{p}{\kappa} \left(\left\| \mathbf{1} + \frac{\kappa}{p} \mathbf{u} \right\|_p - 1 \right)$. The function $\psi_{\kappa,p}$ can be equivalently written as

$$\psi_{\kappa,p}(\mathbf{u}) = f_{\kappa,p}^{-1} \left(\sum_{h=1}^m f_{\kappa,p}(u_h) \right) \quad \text{where } f_{\kappa,p}(u_h) = \left(1 + \frac{\kappa}{p} u_h \right)^p$$

Recall that $\|\mathbf{u}\|_p = g^{-1} \left(\sum_{h=1}^m g(u_h) \right)$ where $g(u_h) = (u_h)^p$. Informally, $\psi_{\kappa,p}(\cdot)$ is a smooth approximation of $\|\cdot\|_p$ as shown later in Lemma 1. In the paper, we are interested in the congestion which is the ℓ_∞ -norm of the load vectors. It is well-known that the ℓ_∞ -norm of any vector can be approximated by ℓ_p -norm of that vector where m is the number of coordinates and $p = \log m$. Molinaro [22] introduced the function $\psi_{\kappa,p}$ as a smoother version of ℓ_p -norm and showed that using function $\psi_{\kappa,p}$, one can obtain tighter bound than using directly the ℓ_p -norm function for the scheduling problem of minimizing the ℓ_p -norm of the load vectors in the random-order model.

First, observe that

$$\nabla \psi_{\kappa,p}(\mathbf{u}) = \frac{p}{\kappa} \cdot \nabla \left\| \mathbf{1} + \frac{\kappa}{p} \mathbf{u} \right\|_p \in \ell_q^+ \quad (1)$$

where $q = p/(p-1)$ since

$$\begin{aligned} \frac{p}{\kappa} \cdot \frac{\partial}{\partial u_h} \left\| \mathbf{1} + \frac{\kappa}{p} \mathbf{u} \right\|_p &= \frac{\left(1 + \frac{\kappa}{p} u_h \right)^{p-1}}{\left(\sum_{h=1}^m \left(1 + \frac{\kappa}{p} u_h \right)^p \right)^{1-1/p}} \quad \forall 1 \leq h \leq m \\ \Rightarrow \|\nabla \psi_{\kappa,p}(\mathbf{u})\|_q &= 1. \end{aligned}$$

The following lemma shows useful properties of functions $\psi_{\kappa,p}$'s and relates them to the ℓ_p -norm function.

► **Lemma 1** ([22]). *For arbitrary $\kappa > 0$, it holds that*

■ *For all $\mathbf{u} \in \mathbb{R}_+^m$,*

$$\|\mathbf{u}\|_p \leq \psi_{\kappa,p}(\mathbf{u}) \leq \|\mathbf{u}\|_p + \frac{p(m^{1/p} - 1)}{\kappa} \quad (2)$$

■ *For all $\mathbf{u} \in \mathbb{R}_+^m$ and $\mathbf{v} \in [0, 1]^m$, for every coordinate $1 \leq h \leq m$,*

$$e^{-\kappa} (\nabla \psi_{\kappa,p}(\mathbf{u}))_h \leq (\nabla \psi_{\kappa,p}(\mathbf{u} + \mathbf{v}))_h \leq e^\kappa (\nabla \psi_{\kappa,p}(\mathbf{u}))_h \quad (3)$$

The following key inequality is proved in [22, Lemma 3.1].

► **Lemma 2** ([22]). *Consider a set of vector $\{\mathbf{v}_1, \dots, \mathbf{v}_k\} \in [0, 1]^m$ and let V_1, \dots, V_t be sample without replacement from this set for $1 \leq t \leq k$. Let U be a random vector in ℓ_q^+ that depends only on V_1, \dots, V_{t-1} . Then, for all $\kappa > 0$,*

$$\mathbb{E} [\langle V^t, U \rangle] \leq e^\kappa \|\mathbb{E} V_t\|_p + \frac{1}{k - (t - 1)} \cdot \frac{p(m^{1/p} - 1)}{\kappa}$$

The following corollary is a direct consequence by replacing κ by $\kappa \cdot \frac{1}{4} \log \log n$.

► **Corollary 3.** *Consider a set of vector $\{\mathbf{v}_1, \dots, \mathbf{v}_k\} \in [0, 1]^m$ and let V_1, \dots, V_t be sample without replacement from this set for $1 \leq t \leq k$. Let U be a random vector in ℓ_q^+ that depends only on V_1, \dots, V_{t-1} . Then, for all $\kappa > 0$,*

$$\mathbb{E} [\langle V^t, U \rangle] \leq e^\kappa (\log^{1/4} n) \|\mathbb{E} V_t\|_p + \frac{1}{k - (t - 1)} \cdot \frac{p(m^{1/p} - 1)}{\kappa} \frac{1}{\frac{1}{4} \log \log n}$$

► **Remark.** We emphasize that Lemma 2 and Corollary 3 hold with arbitrary $\kappa > 0$ (not necessarily $0 < \kappa < 1$). Molinaro [22] proved Lemma 2 using the regret-minimization technique from online learning. It has been observed that there is an interesting connection between regret minimization and the random-order model: regret minimization techniques can be used to prove probability inequalities. This direction has been recently explored in [1, 13, 22]. In particular, employing Lemma 2 and other powerful inequalities, Molinaro [22] proved competitive algorithms for the load balancing problem in the random-order model.

3 An $O(\log n / \log \log n)$ -Competitive Algorithm in Random-Order Setting

We consider the SVCR problem in the random-order setting with the objective of minimizing the ℓ_p -norm of edge loads. The algorithm for the congestion objective will be deduced by choosing appropriate parameters.

Formulation. We say that C is a *configuration* if C is a partial feasible solution of the problem. In other words, a configuration C is a set $\{(i, P_{ij}) : 1 \leq i \leq k, P_{ij} \in \mathcal{P}_i\}$ where the couple (i, P_{ij}) represents request i and the selected $o_i - d_i$ path P_{ij} in configuration C to satisfy request i . Given an arrival order (a permutation) π , denote $\pi(t)$ the request which is released at step t in the order π . For any permutation π , let $x_{\pi(t),j}^\pi$ be a variable indicating whether the selected path for request $\pi(t)$ is $P_{\pi(t),j}$. For a configuration C and a permutation π , let z_C^π be a variable such that $z_C^\pi = 1$ if and only if for every $(\pi(t), P_{\pi(t),j}) \in C$, $x_{\pi(t),j}^\pi = 1$. In other words, $z_C^\pi = 1$ iff the selected solution is C when the request arrival order is π .

Let $\ell(i, P_{ij}) \in \mathbb{R}^m$ be the load random vector of path P_{ij} , i.e., $\ell(i, P_{ij})_e = X_{i,e}$ for every $e \in P_{ij}$ and equals 0 otherwise ($e \notin P_{ij}$). Moreover, let $\ell(C)$ be the load random vector of configuration C , i.e., $\ell(C) = \sum_{(i, P_{ij}) \in C} \ell(i, P_{ij})$. The expected cost (ℓ_p -norm objective) of configuration C is $\mathbb{E}_X [\|\ell(C)\|_p]$ where the expectation is taken over the random vectors $X_{i,e}$'s. We consider the following formulation (left-hand side) and the dual of its relaxation.

$$\begin{aligned}
 \min \mathbb{E}_\pi \left[\sum_C \mathbb{E}_X [\|\ell(C)\|_p] z_C^\pi \right] & \quad \max \sum_\pi \left(\sum_t \alpha_t^\pi + \gamma^\pi \right) \\
 \sum_{j: P_{\pi(t),j} \in \mathcal{P}_{\pi(t)}} x_{\pi(t),j}^\pi = 1 \quad \forall \pi, t & \quad \alpha_t^\pi \leq \beta_{t,j}^\pi \quad \forall \pi, t, j \quad (4) \\
 \sum_{C: (\pi(t), P_{\pi(t),j}) \in C} z_C^\pi = x_{\pi(t),j}^\pi \quad \forall \pi, t, j & \quad \gamma^\pi + \sum_{(\pi(t), P_{\pi(t),j}) \in C} \beta_{t,j}^\pi \leq \\
 \sum_C z_C^\pi = 1 \quad \forall \pi & \quad \leq \mathbb{P}[\pi] \cdot \mathbb{E}_X [\|\ell(C)\|_p] \quad \forall \pi, C \quad (5) \\
 x_{\pi(t),j}^\pi, z_C^\pi \in \{0, 1\} \quad \forall \pi, t, j, C &
 \end{aligned}$$

In the primal, the first constraint guarantees that for any arrival order π , request $\pi(t)$ has to be satisfied by some path $P_{\pi(t),j} \in \mathcal{P}_{\pi(t)}$. The second constraint ensures that if request $\pi(t)$ selects path $P_{\pi(t),j}$ then the couple $(\pi(t), P_{\pi(t),j})$ must be in the solution. The third constraint says that one always has to output a solution for the problem.

Algorithm. The algorithm is primarily a form of Greedy Restart introduced by Molinaro [22] in the context of machine load balancing. We consider a generalized version for the SVCR problem in the angle of a primal-dual method with configuration LPs. Informally, for every request the algorithm selects a routing path *greedily* with respect to the function $\psi_{\kappa,p}$ which depends on the current load vector. However, when half of the requests have been considered, the algorithm *restarts* the procedure: it still chooses a routing path greedily with respect to a function $\psi_{\kappa,p}$ but now the function $\psi_{\kappa,p}$ depends on the load vector induced *only* by the second half of the requests. The intuition is the following. In the worst-case lower bound construction [3, 4, 5], at every time given the current routing the adversary traps every algorithm to accumulate the loads on links which become congested later. The restart step in the algorithm avoids accumulating the loads on potentially-congested links. The formal description of the algorithm is the following.

Let $\kappa > 0$ be a fixed parameter to be determined later. Let A_t be the configuration (partial solution) of the algorithm before the arrival of the t^{th} request. Initially, $A_0 = B_0 = \emptyset$. At the arrival of the t^{th} request, denoted as i , select a path P_{i,j^*} that is an optimal solution of

$$\min_{P_{i,j} \in \mathcal{P}_i} \left\{ \psi_{\kappa',p}(\ell(B_t) + \ell(i, P_{i,j})) - \psi_{\kappa',p}(\ell(B_t)) \right\}$$

where ℓ is the load function (defined in the formulation) and $\kappa' = \kappa \cdot \frac{1}{4} \log \log n$. Update $A_{t+1} = A_t \cup (i, P_{i,j^*})$ and $B_{t+1} = B_t \cup (i, P_{i,j^*})$. If $t = k/2 + 1$, reset $B_t = \emptyset$.

In the above description of the algorithm, we need the knowledge of k – the number of requests – in order to reset B_t at $t = k/2 + 1$. In fact, one can implement the algorithm without the knowledge of k as the following. Initially, $A_0 = B_{\text{odd}} = B_{\text{even}} = \emptyset$. At the arrival of the t^{th} request, denoted as i , select a path P_{i,j^*} that is an optimal solution of

$$\begin{cases} \min_{P_{i,j} \in \mathcal{P}_i} \left\{ \psi_{\kappa',p}(\ell(B_{\text{odd}}) + \ell(i, P_{i,j})) - \psi_{\kappa',p}(\ell(B_{\text{odd}})) \right\} & \text{if } t \text{ is odd} \\ \min_{P_{i,j} \in \mathcal{P}_i} \left\{ \psi_{\kappa',p}(\ell(B_{\text{even}}) + \ell(i, P_{i,j})) - \psi_{\kappa',p}(\ell(B_{\text{even}})) \right\} & \text{if } t \text{ is even} \end{cases}$$

where ℓ is the load function (defined in the formulation) and $\kappa' = \kappa \cdot \frac{1}{4} \log \log n$. Update $A_{t+1} = A_t \cup (i, P_{i,j^*})$ and update B_{odd} or B_{even} depending on whether t is odd or even.

Analysis

For the sake of simplicity, we will analyze the algorithm using its first description. In the sequel, we will define the dual variables, prove the feasibility and show the competitive ratio. As κ (so κ') and p are fixed, for simplicity, we drop the indices κ' and p in $\psi_{\kappa',p}$.

Dual variables. For any permutation σ , denote A_t^σ and B_t^σ as the configurations A_t and B_t (respectively) in the execution of algorithm (before the arrival of the t^{th} request assuming that the request arrival order is σ). Define the dual variables as follows.

$$\begin{aligned}\beta_{t,j}^\pi &:= \frac{\mathbb{P}[\pi]}{e^{2\kappa}(\log^{1/4} n)} \mathbb{E}_X \mathbb{E}_\sigma \left[\psi(\ell(B_t^\sigma) + \ell(\sigma(t), P_{\sigma(t),j})) - \psi(\ell(B_t^\sigma)) \right], \\ \alpha_t^\pi &:= \frac{\mathbb{P}[\pi]}{e^{2\kappa}(\log^{1/4} n)} \mathbb{E}_X \mathbb{E}_\sigma \left[\min_j \left\{ \psi(\ell(B_t^\sigma) + \ell(\sigma(t), P_{\sigma(t),j})) - \psi(\ell(B_t^\sigma)) \right\} \right] \\ &= \frac{\mathbb{P}[\pi]}{e^{2\kappa}(\log^{1/4} n)} \mathbb{E}_X \mathbb{E}_\sigma \left[\psi(\ell(B_t^\sigma) + \ell(\sigma(t), P_{\sigma(t),j^*})) - \psi(\ell(B_t^\sigma)) \right], \\ \gamma^\pi &:= -\frac{\mathbb{P}[\pi]}{2e^{2\kappa}(\log^{1/4} n)} \mathbb{E}_X \mathbb{E}_\sigma [\|\ell(A^\sigma)\|_p].\end{aligned}$$

Informally, $\beta_{t,j}^\pi$ is proportional (up to a factor $\mathbb{P}[\pi] = 1/n!$) to the expected marginal increase (over random order σ) of the objective at the arrival of request $\sigma(t)$ assuming that the selected strategy to serve $\sigma(t)$ is $P_{\sigma(t),j}$. Variable α_t^π is also proportional (up to a factor $\mathbb{P}[\pi] = 1/n!$) to the expected marginal increase of the objective at the arrival of request $\sigma(t)$ due to the algorithm.

► **Lemma 4.** *For any permutation σ , denote A^σ as the final configuration of the algorithm in case that the request arrival order is σ . Suppose that the cost of the algorithm $\mathbb{E}_X \mathbb{E}_\sigma [\|\ell(A^\sigma)\|_p] \geq \frac{4e^\kappa p(m^{1/p}-1)}{\kappa \cdot \frac{1}{4} \log \log n}$. Then the variables defined above constitute a dual feasible solution.*

Proof. The first dual constraint (4) follows immediately the definitions of α_t^π and $\beta_{t,j}^\pi$. In the remaining of the proof, we prove the second dual constraint (5). Fix a configuration C and a permutation π . Let $P_{i,c(i)}$ be the path of request i in configuration C . In other words, configuration C consists of couples $(i, P_{i,c(i)})$ for all requests i .

By the definition of dual variables, the second constraint reads: for any given permutation π and any given configuration C ,

$$\begin{aligned}-\frac{1}{2} \mathbb{P}[\pi] \cdot \mathbb{E}_X \mathbb{E}_\sigma [\|\ell(A^\sigma)\|_p] + \sum_{t=1}^k \mathbb{P}[\pi] \cdot \mathbb{E}_X \mathbb{E}_\sigma [\psi(\ell(B_t^\sigma) + \ell(\sigma(t), P_{\sigma(t),j})) - \psi(\ell(B_t^\sigma))] \\ \leq e^{2\kappa}(\log^{1/4} n) \cdot \mathbb{P}[\pi] \cdot \mathbb{E}_X [\|\ell(C)\|_p]\end{aligned}$$

where for any permutation σ , the path $P_{\sigma(t),c(\sigma(t))}$ of request $\sigma(t)$ is completely determined in configuration C , i.e., $(\sigma(t), P_{\sigma(t),c(\sigma(t))}) \in C$. This is equivalent to

$$\begin{aligned}\sum_{t=1}^k \mathbb{E}_X \mathbb{E}_\sigma [\psi(\ell(B_t^\sigma) + \ell(\sigma(t), P_{\sigma(t),j})) - \psi(\ell(B_t^\sigma))] \\ \leq e^{2\kappa}(\log^{1/4} n) \cdot \mathbb{E}_X [\|\ell(C)\|_p] + \frac{1}{2} \cdot \mathbb{E}_X \mathbb{E}_\sigma [\|\ell(A^\sigma)\|_p].\end{aligned}\tag{6}$$

We prove Inequality (6). First we bound the sum in the left-hand side for all $1 \leq t \leq k/2$.

$$\begin{aligned}
 & \mathbb{E}_X \sum_{t=1}^{k/2} \mathbb{E}_\sigma \left[\psi(\ell(B_t^\sigma) + \ell(\sigma(t), P_{\sigma(t), c(\sigma(t))})) - \psi(\ell(B_t^\sigma)) \right] \\
 & \leq \mathbb{E}_X \sum_{t=1}^{k/2} \mathbb{E}_\sigma \left[\left\langle \nabla \psi(\ell(B_t^\sigma) + \ell(\sigma(t), P_{\sigma(t), c(\sigma(t))})), \ell(\sigma(t), P_{\sigma(t), c(\sigma(t))}) \right\rangle \right] \\
 & \leq e^\kappa \sum_{t=1}^{k/2} \mathbb{E}_X \mathbb{E}_\sigma \left[\left\langle \nabla \psi(\ell(B_t^\sigma)), \ell(\sigma(t), P_{\sigma(t), c(\sigma(t))}) \right\rangle \right] \\
 & \leq e^\kappa \cdot \sum_{t=1}^{k/2} \left(e^\kappa (\log^{1/4} n) \cdot \mathbb{E}_X \left\| \mathbb{E}_\sigma \left[\ell(\sigma(t), P_{\sigma(t), c(\sigma(t))}) \right] \right\|_p + \frac{1}{k-t+1} \cdot \frac{p(m^{1/p} - 1)}{\kappa \cdot \frac{1}{4} \log \log n} \right) \\
 & = e^{2\kappa} (\log^{1/4} n) \cdot \frac{k}{2} \cdot \mathbb{E}_X \left\| \frac{\ell(C)}{k} \right\|_p + e^\kappa \sum_{t=1}^{k/2} \frac{1}{k-t+1} \cdot \frac{p(m^{1/p} - 1)}{\kappa \cdot \frac{1}{4} \log \log n} \\
 & \leq \frac{e^{2\kappa} (\log^{1/4} n)}{2} \mathbb{E}_X [\|\ell(C)\|_p] + e^\kappa \cdot \frac{p(m^{1/p} - 1)}{\kappa \cdot \frac{1}{4} \log \log n} \\
 & < \frac{e^{2\kappa} (\log^{1/4} n)}{2} \mathbb{E}_X [\|\ell(C)\|_p] + \frac{1}{4} \cdot \mathbb{E}_X \mathbb{E}_\sigma [\|\ell(A^\sigma)\|_p]. \tag{7}
 \end{aligned}$$

Recall that $\ell(\sigma(t), P_{\sigma(t), c(\sigma(t))}) \in [0, 1]^m$. The first and second inequalities follow the convexity of ψ and Lemma 1 (Inequality (3)), respectively. The third inequality holds by Corollary 3 and note that $\nabla \psi(\ell(B_t^\sigma)) \in \ell_t^+$ by observation (1). The next equality is due to the fact that σ is an uniform random order. The last inequality follows the assumption of the algorithm cost.

Now we bound the sum of the left-hand side of Inequality (6) for $k/2 < t \leq k$. That can be done similarly with a subtle observation. For completeness, we show all steps.

$$\begin{aligned}
 & \mathbb{E}_X \sum_{t=k/2+1}^k \mathbb{E}_\sigma \left[\psi(\ell(B_t^\sigma) + \ell(\sigma(t), P_{\sigma(t), c(\sigma(t))})) - \psi(\ell(B_t^\sigma)) \right] \\
 & \leq \mathbb{E}_X \sum_{t=k/2+1}^k \mathbb{E}_\sigma \left[\left\langle \nabla \psi(\ell(B_t^\sigma) + \ell(\sigma(t), P_{\sigma(t), c(\sigma(t))})), \ell(\sigma(t), P_{\sigma(t), c(\sigma(t))}) \right\rangle \right] \\
 & \leq e^\kappa \sum_{t=k/2+1}^k \mathbb{E}_X \mathbb{E}_\sigma \left[\left\langle \nabla \psi(\ell(B_t^\sigma)), \ell(\sigma(t), P_{\sigma(t), c(\sigma(t))}) \right\rangle \right] \\
 & \leq e^\kappa \cdot \sum_{t=k/2+1}^k \left(e^\kappa (\log^{1/4} n) \cdot \mathbb{E}_X \left\| \mathbb{E}_\sigma \left[\ell(\sigma(t), P_{\sigma(t), c(\sigma(t))}) \right] \right\|_p \right. \\
 & \quad \left. + \frac{1}{k-(t-k/2-1)} \cdot \frac{p(m^{1/p} - 1)}{\kappa \cdot \frac{1}{4} \log \log n} \right) \\
 & = e^{2\kappa} (\log^{1/4} n) \cdot \frac{k}{2} \cdot \mathbb{E}_X \left\| \frac{\ell(C)}{k} \right\|_p + e^\kappa \sum_{t=k/2+1}^k \frac{1}{k-(t-k/2-1)} \cdot \frac{p(m^{1/p} - 1)}{\kappa \cdot \frac{1}{4} \log \log n} \\
 & \leq \frac{e^{2\kappa} (\log^{1/4} n)}{2} \mathbb{E}_X [\|\ell(C)\|_p] + e^\kappa \cdot \frac{p(m^{1/p} - 1)}{\kappa \cdot \frac{1}{4} \log \log n} \\
 & < \frac{e^{2\kappa} (\log^{1/4} n)}{2} \mathbb{E}_X [\|\ell(C)\|_p] + \frac{1}{4} \cdot \mathbb{E}_X \mathbb{E}_\sigma [\|\ell(A^\sigma)\|_p]. \tag{8}
 \end{aligned}$$

All the above equalities and inequalities follow by the same arguments as before except the third inequality. In the latter, we apply Corollary 3 with the observation that $\nabla\psi(\ell(B_t^\sigma))$ depends only on $(t - k/2 - 1)$ random load variables due to the fact that the algorithm restarts at $t = k/2$. This interesting idea has been observed by Molinaro [22]. Note that this is the only place we use the restart property of the algorithm.

Hence, summing Inequalities (7) and (8), Inequality (6) follows. \blacktriangleleft

► **Theorem 5.** *For any arbitrary $\kappa > 0$, the algorithm has expected cost at most $2e^{2\kappa}(\log^{1/4} n)$ times the optimal value plus an additive constant $\frac{4e^\kappa p(m^{1/p} - 1)}{\kappa \cdot \frac{1}{4} \log \log n}$ for the SVCR problem with ℓ_p -norm objective in the random-order setting.*

Proof. Consider first the case where the (expected) cost of the algorithm $\mathbb{E}_X \mathbb{E}_\sigma [\|\ell(A^\sigma)\|_p] \geq \frac{4e^\kappa p(m^{1/p} - 1)}{\kappa \cdot \frac{1}{4} \log \log n}$. Then, by the algorithm and the definition of dual variables, the dual objective equals

$$\begin{aligned} & \sum_{\pi} \left(\sum_t \alpha_t^\pi + \gamma^\pi \right) \\ &= \frac{\mathbb{P}[\pi]}{e^{2\kappa}(\log^{1/4} n)} \sum_{\pi, t} \mathbb{E}_X \mathbb{E}_\sigma [\psi(\ell(B_t^\sigma) + \ell(\sigma(t), P_{\sigma(t), j^*})) - \psi(\ell(B_t^\sigma))] \\ & \quad - \frac{\mathbb{P}[\pi]}{2e^{2\kappa}(\log^{1/4} n)} \sum_{\pi} \mathbb{E}_X \mathbb{E}_\sigma [\|\ell(A^\sigma)\|_p] \\ &= \frac{1}{e^{2\kappa}(\log^{1/4} n)} \mathbb{E}_X \mathbb{E}_\sigma [\psi(\ell(B_{n/2}^\sigma) + \psi(\ell(B_n^\sigma))] - \frac{1}{2e^{2\kappa}(\log^{1/4} n)} \cdot \mathbb{E}_X \mathbb{E}_\sigma [\|\ell(A^\sigma)\|_p] \\ &\geq \frac{1}{e^{2\kappa}(\log^{1/4} n)} \mathbb{E}_X \mathbb{E}_\sigma \left[\|\ell(B_{n/2}^\sigma)\|_p + \|\ell(B_n^\sigma)\|_p \right] - \frac{1}{2e^{2\kappa}(\log^{1/4} n)} \cdot \mathbb{E}_X \mathbb{E}_\sigma [\|\ell(A^\sigma)\|_p] \\ &\geq \frac{1}{e^{2\kappa}(\log^{1/4} n)} \mathbb{E}_X \mathbb{E}_\sigma \left[\|\ell(B_{n/2}^\sigma) + \ell(B_n^\sigma)\|_p \right] - \frac{1}{2e^{2\kappa}(\log^{1/4} n)} \cdot \mathbb{E}_X \mathbb{E}_\sigma [\|\ell(A^\sigma)\|_p] \\ &= \frac{1}{e^{2\kappa}(\log^{1/4} n)} \mathbb{E}_X \mathbb{E}_\sigma [\|\ell(A^\sigma)\|_p] - \frac{1}{2e^{2\kappa}(\log^{1/4} n)} \cdot \mathbb{E}_X \mathbb{E}_\sigma [\|\ell(A^\sigma)\|_p] \\ &= \frac{1}{2e^{2\kappa}(\log^{1/4} n)} \cdot \mathbb{E}_X \mathbb{E}_\sigma [\|\ell(A^\sigma)\|_p]. \end{aligned}$$

The first inequality follows the properties of ψ (Lemma 1, Inequality (2)). The second inequality is due to the norm inequality $\|\mathbf{a}\|_p + \|\mathbf{b}\|_p \geq \|\mathbf{a} + \mathbf{b}\|_p$. The subsequent equality holds since $B_{n/2}^\sigma \uplus B_n^\sigma = A^\sigma$ (note that $B_{n/2+1}^\sigma$ was re-initialized as an empty set).

Besides, the primal is $\mathbb{E}_X \mathbb{E}_\sigma [\|\ell(A^\sigma)\|_p]$. Therefore, by weak duality, $\mathbb{E}_X \mathbb{E}_\sigma [\|\ell(A^\sigma)\|_p] \leq 2e^{2\kappa}(\log^{1/4} n)OPT$ where OPT is the value of an optimal solution.

Now consider the case that the expected cost of the algorithm $\mathbb{E}_X \mathbb{E}_\sigma [\|\ell(A^\sigma)\|_p]$ is at most $\frac{4e^\kappa p(m^{1/p} - 1)}{\kappa \cdot \frac{1}{4} \log \log n}$. Obviously, $\mathbb{E}_X \mathbb{E}_\sigma [\|\ell(A^\sigma)\|_p] < OPT + \frac{4e^\kappa p(m^{1/p} - 1)}{\kappa \cdot \frac{1}{4} \log \log n}$. Therefore, combining the cases we deduce that

$$\mathbb{E}_X \mathbb{E}_\sigma [\|\ell(A^\sigma)\|_p] \leq 2e^{2\kappa}(\log^{1/4} n)OPT + \frac{4e^\kappa p(m^{1/p} - 1)}{\kappa \cdot \frac{1}{4} \log \log n}. \quad \blacktriangleleft$$

► **Corollary 6.** *Assume that the optimum solution is at least 1 (i.e., the optimal routing is sufficiently congested). Then the algorithm with parameters $p = O(\log n)$ and $\kappa = 1$ is $O(\log n / \log \log n)$ -approximation for the SVCR problem.*

39:10 A Competitive Algorithm for Random-Order Stochastic Virtual Circuit Routing

Proof. Recall that the congestion (ℓ_∞ -norms over edge loads) can be approximated up to a constant factor by the ℓ_p -norm function for $p = \log m = O(\log n)$. Applying Theorem 5 for $p = O(\log n)$ and $\kappa = 1$, we have the following upper-bound on the congestion of the algorithm:

$$\begin{aligned} O(e^{2\kappa}(\log^{1/4} n))OPT + \frac{4e^\kappa p(m^{1/p} - 1)}{\kappa \cdot \frac{1}{4} \log \log n} &\leq O\left(e^{2\kappa}(\log^{1/4} n) + \frac{e^\kappa \log n}{\kappa \cdot \frac{1}{4} \log \log n}\right)OPT \\ &= O\left(\log^{1/4} n + \frac{\log n}{\log \log n}\right)OPT = O\left(\frac{\log n}{\log \log n}\right)OPT \end{aligned} \quad (9)$$

where OPT is the value of an optimal solution. As the optimum solution is at least 1, the corollary follows. ◀

4 A Simple $\Theta(\log n / \log \log n)$ -Approximation Algorithm for Virtual Circuit Routing

In this section, we revisit the classic virtual circuit routing problem and provide a simple algorithm with tight approximation guarantee (assuming some complexity hypothesis).

Virtual Circuit Routing. In the problem, there is a directed graph $G(V, E)$ where $|V| = n$ and a collection of k requests. A request i for $1 \leq i \leq k$ is specified by a origin-destination pairs $o_i, d_i \in V$, and a positive weight w_i representing the (deterministic) load of request i on an edge e if it is used by request i . The goal is to choose for each request i a routing path connecting o_i and d_i so that the *congestion* induced by the collection of all paths is minimized. The load of an edge e is equal to the total weight of requests routing through e , i.e., $\sum_i w_i$ where the sum is taken over all requests i whose some path contains e . The congestion of a collection of paths is the maximum load over all edges.

Approximation algorithm.

1. Normalize all request weights by dividing every weight by $\max_{i'} w_{i'}$. The new *normalized* weights $\tilde{w}_i = \frac{w_i}{\max_{i'} w_{i'}}$ satisfy $\tilde{w}_i \in [0, 1]$.
2. Define the parameters $p = O(\log n)$, $\kappa = 1$ and $\kappa' = \frac{1}{4} \log \log n$.
3. Sample an uniform random order of the requests and consider requests in this order.
4. Let A_t be the configuration (partial solution) of the algorithm before the arrival of the t^{th} request. Initially, $A_0 = B_0 = \emptyset$. At the arrival of the t^{th} request, denoted as i , select a path P_{i,j^*} that is an optimal solution of

$$\min_{P_{i,j} \in \mathcal{P}_i} \psi_{\kappa', p}(\tilde{\ell}(B_t) + \tilde{\ell}(i, P_{i,j})) - \psi_{\kappa', p}(\tilde{\ell}(B_t))$$

where $\tilde{\ell}$ is the load function with respect to the normalized weights. Update $A_{t+1} = A_t \cup (i, P_{i,j^*})$ and $B_{t+1} = B_t \cup (i, P_{i,j^*})$. If $t = k/2 + 1$, reset $B_t = \emptyset$.

► **Theorem 7** ([23, 24, 6]). *The algorithm has approximation ratio $O(\log n / \log \log n)$.*

Proof. By Corollary 6, specifically Inequality (9), we have the bound on the congestion of the algorithm (after normalizing the weights):

$$\mathbb{E}[\widetilde{ALG}] \leq O\left(\frac{\log n}{\log \log n}\right)\widetilde{OPT}$$

where \widetilde{ALG} and \widetilde{OPT} are the congestions of the algorithm and the optimal solution with normalized weights, respectively. Multiplying both sides by the normalizing factor, the theorem follows. ◀

5 Conclusion

In the paper, we have provided a competitive algorithm for the SCVR problem and prove that the quality of approximation solutions to the problem can be preserved even with the presence of uncertainty. Through the paper, we also show that primal-dual approaches are robust in the stochastic model and the random-order model can be used to design/simplify randomized approximation algorithms. A direction is to design randomized algorithms for other (stochastic) problems using primal-dual techniques and random-order request sequences.

References

- 1 Shipra Agrawal and Nikhil R Devanur. Fast algorithms for online stochastic convex programming. In *Proc. 26th ACM-SIAM symposium on Discrete algorithms*, pages 1405–1424, 2014.
- 2 Matthew Andrews and Lisa Zhang. Logarithmic hardness of the directed congestion minimization problem. In *Proc. 38th Symposium on Theory of Computing*, pages 517–526, 2006.
- 3 James Aspnes, Yossi Azar, Amos Fiat, Serge Plotkin, and Orli Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *Journal of the ACM (JACM)*, 44(3):486–504, 1997.
- 4 Baruch Awerbuch, Yossi Azar, Edward F Grove, Ming-Yang Kao, P Krishnan, and Jeffrey Scott Vitter. Load balancing in the ℓ_p -norm. In *Proc. 36th Foundations of Computer Science*, pages 383–391, 1995.
- 5 Ioannis Caragiannis. Better bounds for online load balancing on unrelated machines. In *Proc. 19th Symposium on Discrete Algorithms*, pages 972–981, 2008.
- 6 Chandra Chekuri and Mark Idleman. Congestion minimization for multipath routing via multiroute flows. In *Proc. 1st Symposium on Simplicity in Algorithms*, 2018.
- 7 Chandra Chekuri, Jan Vondrak, and Rico Zenklusen. Dependent randomized rounding via exchange properties of combinatorial structures. In *Proc. 51st Annual IEEE Symposium on Foundations of Computer Science*, pages 575–584, 2010.
- 8 Julia Chuzhoy, Venkatesan Guruswami, Sanjeev Khanna, and Kunal Talwar. Hardness of routing with congestion in directed graphs. In *Proc. 39th ACM Symposium on Theory of Computing*, pages 165–178, 2007.
- 9 Benjamin Doerr. Randomly rounding rationals with cardinality constraints and derandomizations. In *Symposium on Theoretical Aspects of Computer Science*, pages 441–452, 2007.
- 10 Rajiv Gandhi, Samir Khuller, Srinivasan Parthasarathy, and Aravind Srinivasan. Dependent rounding and its applications to approximation algorithms. *Journal of the ACM*, 53(3):324–360, 2006.
- 11 Ashish Goel and Piotr Indyk. Stochastic load balancing and related problems. In *Proc. 40th Symposium on Foundations of Computer Science*, pages 579–586, 1999.
- 12 Anupam Gupta, Amit Kumar, Viswanath Nagarajan, and Xiangkun Shen. Stochastic load balancing on unrelated machines. In *Proc. 29th Symposium on Discrete Algorithms*, pages 1274–1285, 2018.
- 13 Anupam Gupta and Marco Molinaro. How the experts algorithm can help solve LPs online. *Mathematics of Operations Research*, 41(4):1404–1431, 2016.
- 14 Varun Gupta, Benjamin Moseley, Marc Uetz, and Qiaomin Xie. Stochastic online scheduling on unrelated machines. In *Conference on Integer Programming and Combinatorial Optimization*, pages 228–240, 2017.
- 15 Sungjin Im, Benjamin Moseley, and Kirk Pruhs. Stochastic scheduling of heavy-tailed jobs. In *Proc. 32nd Symposium on Theoretical Aspects of Computer Science*, pages 474–486, 2015.
- 16 Wataru Kishimoto and Masashi Takeuchi. m-route flows in a network. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, 77(5):1–18, 1994.

39:12 A Competitive Algorithm for Random-Order Stochastic Virtual Circuit Routing

- 17 Jon Kleinberg, Yuval Rabani, and Éva Tardos. Allocating bandwidth for bursty connections. *SIAM Journal on Computing*, 30(1):191–217, 2000.
- 18 Jian Li and Amol Deshpande. Maximizing expected utility for stochastic combinatorial optimization problems. *Mathematics of Operations Research*, 2018.
- 19 Jian Li and Wen Yuan. Stochastic combinatorial optimization via poisson approximation. In *Proc. 45th Symposium on Theory of Computing*, pages 971–980, 2013.
- 20 Nicole Megow, Marc Uetz, and Tjark Vredeveld. Models and algorithms for stochastic online scheduling. *Mathematics of Operations Research*, 31(3):513–525, 2006.
- 21 Rolf H Möhring, Andreas S Schulz, and Marc Uetz. Approximation in stochastic scheduling: the power of LP-based priority policies. *Journal of the ACM*, 46(6):924–942, 1999.
- 22 Marco Molinaro. Online and random-order load balancing simultaneously. In *Proc. 28th ACM-SIAM Symposium on Discrete Algorithms*, pages 1638–1650, 2017.
- 23 Prabhakar Raghavan and Clark D Thompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987.
- 24 Aravind Srinivasan. Distributions on level-sets with applications to approximation algorithms. In *Proc. 42nd IEEE Symposium on Foundations of Computer Science*, pages 588–597, 2001.
- 25 Nguyen Kim Thang. Online primal-dual algorithms with configuration linear programs. *arXiv*, 2017. [arXiv:1708.04903](https://arxiv.org/abs/1708.04903).

An Improved Data Structure for Left-Right Maximal Generic Words Problem

Yuta Fujishige

Department of Informatics, Kyushu University, Japan
yuta.fujishige@inf.kyushu-u.ac.jp

Yuto Nakashima

Department of Informatics, Kyushu University, Japan
yuto.nakashima@inf.kyushu-u.ac.jp

Shunsuke Inenaga

Department of Informatics, Kyushu University, Japan
inenaga@inf.kyushu-u.ac.jp

Hideo Bannai 

Department of Informatics, Kyushu University, Japan
bannai@inf.kyushu-u.ac.jp

Masayuki Takeda

Department of Informatics, Kyushu University, Japan
takeda@inf.kyushu-u.ac.jp

Abstract

For a set D of documents and a positive integer d , a string w is said to be *d-left-right maximal*, if (1) w occurs in at least d documents in D , and (2) any proper superstring of w occurs in less than d documents. The *left-right-maximal generic words problem* is, given a set D of documents, to preprocess D so that for any string p and for any positive integer d , all the superstrings of p that are *d-left-right maximal* can be answered quickly. In this paper, we present an $O(n \log m)$ space data structure (in words) which answers queries in $O(|p| + o \log \log m)$ time, where n is the total length of documents in D , m is the number of documents in D and o is the number of outputs. Our solution improves the previous one by Nishimoto et al. (PSC 2015), which uses an $O(n \log n)$ space data structure answering queries in $O(|p| + r \cdot \log n + o \cdot \log^2 n)$ time, where r is the number of right-extensions q of p occurring in at least d documents such that any proper right extension of q occurs in less than d documents.

2012 ACM Subject Classification Mathematics of computing → Combinatorial algorithms; Mathematics of computing → Combinatorics on words

Keywords and phrases generic words, suffix trees, string processing algorithms

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.40

Funding *Yuto Nakashima*: Supported by JSPS KAKENHI Grant Number JP18K18002.

Shunsuke Inenaga: Supported by JSPS KAKENHI Grant Number JP17H01697.

Hideo Bannai: Supported by JSPS KAKENHI Grant Number JP16H02783.

Masayuki Takeda: Supported by JSPS KAKENHI Grant Number JP18H04098.

1 Introduction

String Data Mining is an important research area which has received special attention. One of the fundamental tasks in this area is the frequent pattern mining, the aim of which is to find patterns occurring in at least d documents in D for a given collection D of documents and a given threshold d , where the patterns are drawn from a fixed hypothesis space. The task is useful not only in extracting patterns which characterize the documents in D , but also in enumerating candidates for the most classificatory pattern that separates two given sets of



© Yuta Fujishige, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda; licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 40; pp. 40:1–40:12

Leibniz International Proceedings in Informatics



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

strings. The hypothesis space varies depending upon users' particular interest or purpose, and is ranging from the substring patterns to the VLDC patterns. Frequent substring patterns are often referred to as *generic words*. The generic words mining problem (or the frequent substring pattern mining problem) has a wide variety of applications, e.g., Computational Biology, Text mining, and Text Classification [5, 2, 3].

One interesting variant of the generic words mining problem is the *right maximal generic words problem*, formulated by Kucherov et al. [5]. In this variant, a pattern p is given as additional input, which limits the outputs to the right extensions of p . Moreover, the outputs are limited to the *maximal* ones. Formally, the problem is to preprocess D so that, for any pattern p and for any threshold d , all right extensions of p that are d -right maximal can be computed efficiently, where a string w is said to be d -right maximal if x occurs in at least d documents but xa occurs in less than d documents for any character a . They presented in [5] an $O(n)$ -size data structure which answers queries in $O(|p| + r)$ time, where n is the total length of strings in D and r is the number of outputs. Later, Biswas et al. [2] developed a succinct data structure of size $n \log |\Sigma| + o(n \log |\Sigma|) + O(n)$ bits of space, which answers queries in $O(|p| + \log \log n + r)$ time.

As a generalization, Nishimoto et al. [7] defined the *left-right-maximal generic word problem*. In this problem, all superstrings of p that are d -left-right maximal should be answered, where a string w is said to be d -left-right maximal if x has a document frequency $\geq d$ but xa and ax respectively have a document frequency $< d$ for any character a .

One naive solution to this problem is to compute the sets M_d of d -left-right maximal strings for $d = 1, \dots, m$, where m is the number of documents in D and then apply the optimal algorithm of Muthukrishnan [6] for the document listing problem, regarding M_d as input document collection. The query time is $O(|p| + o)$ time, where o is the number of outputs. The space requirement is $O(n^2 \log m)$ since the Muthukrishnan algorithm uses the (generalized) suffix tree of input document collection and the size of suffix tree for M_d can be shown to be $O(n^2/d)$ for every $d = 1, \dots, m$. The $O(n^2 \log m)$ space requirement is, however, impractical when dealing with a large-scale document collection.

In [7] Nishimoto et al. presented an $O(n \log n)$ -space data structure which answers queries in $O(|p| + r \log n + o \log^2 n)$ time, where r is the number of d -right-maximal strings that subsume p as a prefix. The factor $O(r \log n)$ is for computing the d -right-maximal right extensions of p , which are required for computing d -left-right-maximal extensions of p in their method.

In this paper, we address the left-right-maximal generic word problem and propose an $O(n \log m)$ -space data structure with query time $O(|p| + o \log \log m)$. The data structure outperforms the previous work by Nishimoto et al. [7] both in the query time and in the space requirement.

Our method uses the suffix trees of M_d for $d = 1, \dots, m$. For a string set $S = \{w_1, \dots, w_\ell\}$, Usually, "the suffix tree of S " means the suffix tree of $\{w_1\$_1, \dots, w_\ell\$_\ell\}$ with ℓ distinct endmarkers $\$_1, \dots, \$_\ell$, or the suffix tree of $S\$ = \{w_1\$, \dots, w_\ell\}$ with a single endmarker $\$$. In both cases, the size of suffix tree is proportional to the total length of the strings in S . The total size of suffix trees of $M_d\$$ for $d = 1, \dots, m$ is $O(nm)$, where n is the total length of D . Our idea in reducing the space requirement is to replace the suffix tree of $M_d\$$ with the suffix tree of M_d . Removing the endmarker successfully reduces the $O(nm)$ total size of the suffix trees to $O(n \log m)$, with a small sacrifice of query time.

2 Preliminaries

2.1 Strings

Let Σ be an alphabet, that is, a nonempty, finite set of characters. Throughout this paper, we assume that Σ is an ordered alphabet of constant size. A *string* over Σ is a finite sequence of characters from Σ . Let Σ^* denote the set of strings over Σ . The *length* of a string w is the number of characters in w and denoted by $|w|$. The string of length 0 is called the *empty string* and denoted by ε . Let $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. The i -th character of a string w is denoted by $w[i]$ for $1 \leq i \leq |w|$. Strings x , y , and, z are, respectively, said to be a *prefix*, *substring*, and *suffix* of string $w = xyz$. The substring of a string w that begins at position i and ends at position j is denoted by $w[i..j]$ for $1 \leq i \leq j \leq |w|$. That is, $w[i..j] = w[i] \cdots w[j]$. For convenience, let $w[i..j] = \varepsilon$ for $i > j$. We use $w[..j]$ and $w[i..]$ as abbreviations of $w[1..j]$ and $w[i..|w|]$. Let $Pre(w)$, $Sub(w)$ and $Suf(w)$ denote the sets of prefixes, substrings, and suffixes of a string w , respectively. Let $Pre(S) = \bigcup_{w \in S} Pre(w)$, $Sub(S) = \bigcup_{w \in S} Sub(w)$ and $Suf(S) = \bigcup_{w \in S} Suf(w)$ for any set S of strings. The *reversal* of a string w , denoted by w^R , is defined to be $w[|w|] \dots w[1]$. Let $S^R = \{w^R \mid w \in S\}$ for any set S of strings.

The *longest repeating suffix* of a string x is the longest suffix of x that occurs elsewhere in x . Let $LRS(x)$ denote the length of the longest repeating suffix of x . We note that any suffix of x longer than $LRS(x)$ occurs only once in x .

2.2 d -left-right maximality of strings

Let D be a set of documents (strings). The *document frequency* of a string x in D , denoted by $df_D(x)$, is defined to be the number of documents in D that contain x as a substring. We write $df(x)$ instead of $df_D(x)$ when D is clear from the context.

A string x is said to be *d -left maximal* w.r.t. D if $df(x) \geq d$ and $df(ax) < d$ for all $a \in \Sigma$, and said to be *d -right maximal* w.r.t. D if $df(x) \geq d$ and $df(xa) < d$ for all $a \in \Sigma$. A string x is said to be *d -left-right maximal* w.r.t. D if it is d -left maximal and d -right maximal w.r.t. D . Let M_d denote the sets of d -left-right maximal strings w.r.t. D .

► **Example 1.** For $D = \{\text{aaabaabaaa}, \text{aaabaabbba}, \text{aabababbaa}, \text{abaababbba}\}$, the sets of d -left-right maximal strings for $d = 1, 2, 3, 4$ are as follows: $M_1 = D$, $M_2 = \{\text{aaabaab}, \text{aabab}, \text{abaaba}, \text{ababb}, \text{abbba}\}$, $M_3 = \{\text{aaba}, \text{abaab}, \text{abb}, \text{bba}\}$ and $M_4 = \{\text{aaba}, \text{baa}\}$.

► **Lemma 2** ([5]). *For any set D of strings with total length n , the number of d -right maximal strings w.r.t. D is $O(n/d)$.*

► **Lemma 3.** *For any string y the following statements hold.*

1. *Let z be the shortest string such that $yz \in Suf(M_d)$. If $xyz \in M_d$ for some string x , then xy is d -left maximal.*
2. *Let x be the shortest string such that $xy \in Pre(M_d)$. If $xyz \in M_d$ for some string z , then yz is d -right maximal.*

Proof. It suffices to give proof only for the first statement. Suppose to the contrary that xy is not d -left maximal. Then, $df(xy) \geq df(xyz) \geq d$, and there exists some $\alpha \in \Sigma^+$ such that αxy is d -left maximal. Since xyz is d -maximal, $df(\alpha xyz) < d$. Furthermore, since $df(\alpha xy) \geq d$, there exists a prefix z' of z such that $\alpha xyz'$ is d -maximal and $|z'| < |z|$. This implies $yz' \in Suf(M_d)$ and contradicts that z is the shortest such string. Therefore, xy must be d -left maximal. ◀

2.3 Suffix trees

Let $S = \{w_1, \dots, w_\ell\}$ be a set of nonempty strings with total length n . The suffix tree [8] of S , denoted by $ST(S)$, is a path-compressed trie which represents all suffixes of S . More formally, $ST(S)$ is an edge-labeled rooted tree such that (1) Every internal node is branching; (2) The out-going edges of every internal node begin with mutually distinct characters; (3) Each edge is labeled by a non-empty substring of S ; (4) For each suffix s of S , there is a unique path from the root which spells out s and the path possibly ends on an edge; (5) Each path from the root to a leaf spells out a suffix of S . It follows from the definition of $ST(S)$ that the numbers of nodes and edges in $ST(S)$ are $O(n)$, respectively. By representing every edge label x by a triple (i, j, k) of integers such that $x = w_k[i..j]$, $ST(S)$ can be represented in $O(n)$ space. The *size* of suffix tree $ST(S)$ is defined to be the number of nodes and is denoted by $|ST(S)|$.

A node v of $ST(S)$ is said to *represent* a string x if the path from the root to v spells out x . For a substring x of S , the *locus* of x in $ST(S)$ is defined to be the highest node v that represents a right extension of x . A string x is said to be *explicit* in $ST(S)$ if there exists a node v of $ST(S)$ that represents x and *implicit* otherwise.

In this paper, we properly use the suffix trees of the following three types to suit its use.

1. $ST(\bar{S})$ where $\bar{S} = \{w_1\$1, \dots, w_\ell\$ \ell\}$ and $\$1, \dots, \ℓ are mutually distinct endmarkers not in Σ .
2. $ST(S\$)$ where $\$$ is an endmarker not in Σ .
3. $ST(S)$ without endmarker.

The above suffix trees are all capable of determining whether $x \in Sub(S)$ for any $x \in \Sigma^+$. $ST(S)$ cannot distinguish the elements of $Suf(S)$ from those of $Sub(S)$ whereas $ST(S\$)$ and $ST(\bar{S})$ can determine whether $x \in Suf(S)$ for any $x \in Sub(S)$. In addition, $ST(\bar{S})$ can determine the set of indices k such that $x \in Suf(w_k)$. It is easy to see that:

► **Lemma 4.** $|ST(\bar{S})| \geq |ST(S\$)| \geq |ST(S)|$ for any set S of strings.

2.4 Tools

Let x be a fixed string over $A = \{1, \dots, \sigma\}$. The Rank query $rank_x(a, i)$ returns the number of occurrences of $a \in A$ in the prefix $x[..i]$ of x , and the Select query $select_x(a, j)$ returns the position of j -th occurrence of $a \in A$ in x .

► **Lemma 5** ([4]). *There is an $O(|x|)$ space data structure that answers Rank/Select queries in $O(\log \log \sigma)$ time.*

Let T be an ordered tree with n nodes and with function val that maps the nodes to the integers. The find-less-than (FLT) query on tree T is, given a threshold τ and a node v of T , to enumerate the descendants u of v with $val(u) < \tau$.

► **Lemma 6.** *We can build from T an $O(n)$ space data structure in $O(n)$ time that answers FLT queries in $O(out)$ time, where out is the number of outputs.*

Proof. Let v_1, \dots, v_n be the nodes T in the preorder. Let B be an array such that $B[i] = val(v_i)$ for all $i \in [1..n]$. Then, the problem of FLT queries on tree can be reduced to the problem of FLT queries on array B defined as follows:

Given a threshold τ and a subinterval $[i..j]$ of $[1..n]$, enumerate the indices k in $[i..j]$ such that $val(B[k]) < \tau$.

FLT queries on array B of size n can be answered in linear time proportional to the number of outputs, by repeated use of the Range Minimum Query (see [6]). ◀

2.5 Computation model

Our model of computation is the word RAM: We shall assume that the computer word size is at least $\lceil \log_2 n \rceil$, and hence, standard operations on values representing lengths and positions of strings can be manipulated in constant time. Space complexities will be determined by the number of computer words (not bits).

3 Main Result and Algorithm Outline

3.1 Main result

Our problem is formulated as follows.

► **Problem 7.**

To-preprocess: A subset $D = \{w_1, \dots, w_m\}$ of Σ^+ .

Query: A string $p \in \Sigma^+$ and an integer $d \in [1..m]$.

Answer: The strings in $\Sigma^* p \Sigma^* \cap M_d$.

One naive solution to the problem would be to apply the optimal algorithm of Muthukrishnan et al. [6] for the document listing problem regarding M_d as input document collection. This solution requires space proportional to the total size of suffix trees $ST(\overline{M_d})$ for $d = 1, \dots, m$.

► **Lemma 8.** *The suffix trees $ST(M_d\$)$ and $ST(M_d)$ are of size $O(n)$ for any $d = 1, \dots, m$, and the suffix tree $ST(\overline{M_d})$ is of size $O(n^2/d)$ for any $d = 1, \dots, m$.*

Proof. First, we show that $ST(M_d\$)$ has $O(n)$ leaves. Let v be any leaf of $ST(M_d\$)$, and let $x\$$ be the string represented by v . There is a string α such that $\alpha x \in M_d$. Assume, for a contradiction, that x is implicit in $ST(\overline{D})$. Then, there uniquely exists a character a such that every occurrence of x in the strings of \overline{D} is followed by a . This contradicts $\alpha x \in M_d$. Hence x is explicit in $ST(\overline{D})$. The number of leaves of $ST(M_d\$)$ is not greater than the number of nodes of $ST(\overline{D})$, which is $O(n)$. By Lemma 4, $ST(M_d\$)$ and $ST(M_d)$ are of size $O(n)$. Next, we prove that $ST(\overline{M_d})$ has $O(n^2/d)$ leaves. Let v be any leaf of $ST(\overline{M_d})$, and let $x\$_i$ be the string represented by v . As the previous discussion, x is explicit in $ST(\overline{D})$. There are $|M_d|$ endmarkers $\$_j$ in $\overline{M_d}$, and by Lemma 2 we have $|M_d| = O(n/d)$. Hence the number of leaves of $ST(\overline{M_d})$ is not greater than $O(n/d)$ times the number of nodes of $ST(\overline{D})$, which is $O(n^2/d)$. ◀

The naive solution answers queries in $O(|p| + o)$ time using $O(n^2 \log m)$ space, where o is the number of outputs. The $O(n^2 \log m)$ space requirement is, however, impractical for dealing with a large-scale document set.

Our solution reduces the $O(n^2 \log m)$ space requirement to $O(n \log m)$ with a little sacrifice in query response time.

► **Theorem 9.** *There exists an $O(n \log m)$ space data structure for Problem 7 which answers queries in $O(|p| + o \log \log m)$ time, where o is the number of outputs.*

3.2 Algorithm outline

Our task is, given a string $p \in \Sigma^+$, to enumerate the strings $\alpha p \beta$ in M_d with $\alpha, \beta \in \Sigma^*$. One solution would be to enumerate the strings αp in $Pre(M_d)$ with $\alpha \in \Sigma^*$, and then, for each αp enumerate the strings $\alpha p \beta$ in M_d with $\beta \in \Sigma^*$. The resulting enumeration, however,

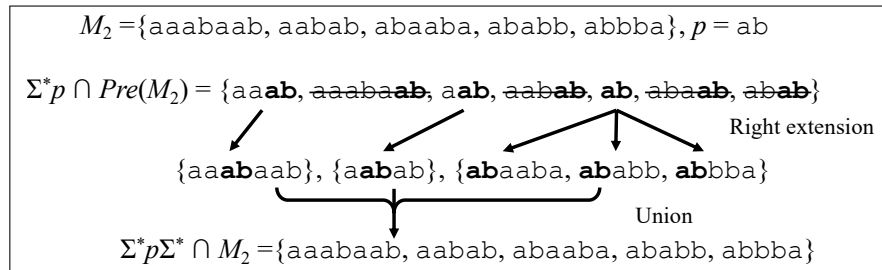
contains duplicates if there is some string in M_d containing p more than once. Consider the string $abaaba$ which contains $p = ab$ twice in Example 10. The strings ab ($\alpha = \varepsilon$) and $abaab$ ($\alpha = aba$) appear in the enumeration of αp , and therefore the string $abaaba$ appears twice in the enumeration of $\alpha p \beta$.

► **Example 10.** Let $D = \{aaabaabaaa, aaabaabbba, aabababbba, abaababbba\}$, $d = 2$ and $p = ab$. Then M_2 is $\{aaabaab, aabab, abaaba, ababb, abbba\}$ and the answer is $\{aaabaab, aabab, abaaba, ababb, abbba\}$. (1) $\Sigma^*p \cap Pre(M_d) = \{aaab, aaabaab, aab, aabab, ab, abaab, abab\}$. (2) Their d -left-right-maximal extensions are $\{aaabaab\}$, $\{aaabaab\}$, $\{aabab\}$, $\{aabab\}$, $\{abaaba, ababb, abbba\}$, $\{abaaba\}$, $\{ababb\}$, respectively. (3) The union of these string sets is $\{aaabaab, aabab, abaaba, ababb, abbba\}$, which coincides with the answer.

In order to avoid such duplicates in enumeration, we put a restriction on the enumeration of the strings $\alpha p \in Pre(M_d)$. That is, we enumerate the strings $\alpha p \in Pre(M_d)$ satisfying the condition that αp contains p just once, which can be replaced with $LRS(\alpha p) < |p|$. The outline of our algorithm is as follows:

- Step 1.** Enumerate the strings αp such that $\alpha \in \Sigma^*$, $\alpha p \in Pre(M_d)$ and $LRS(\alpha p) < |p|$.
- Step 2.** For each string αp obtained in Step 1, enumerate the strings $\alpha p \beta$ such that $\beta \in \Sigma^*$ and $\alpha p \beta \in M_d$.

► **Example 11.** Let $D = \{aaabaabaaa, aaabaabbba, aabababbba, abaababbba\}$, $d = 2$ and $p = ab$. Then M_2 is $\{aaabaab, aabab, abaaba, ababb, abbba\}$ and the answer is $\{aaabaab, aabab, abaaba, ababb, abbba\}$. (1) $\Sigma^*p \cap Pre(M_d) = \{aaab, aaabaab, aab, aabab, ab, abaab, abab\}$. (2) Of the seven strings, the three strings $aaab$, aab , ab satisfy the condition $LRS(x) < |p|$. Their d -right extensions are $\{aaabaab\}$, $\{aabab\}$, $\{abaaba, ababb, abbba\}$, respectively. These sets are mutually disjoint. (3) The union of the disjoint sets is $\{aaabaab, aabab, abaaba, ababb, abbba\}$, which coincides with the answer (see Figure 1).



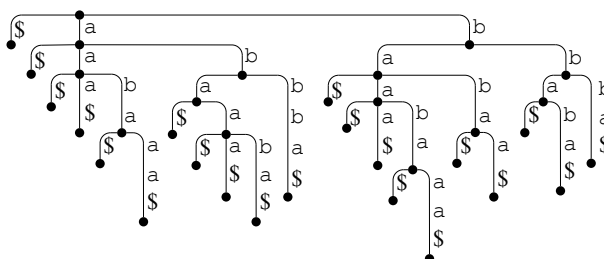
■ **Figure 1** Illustration of Example 11.

4 Simplified Solution

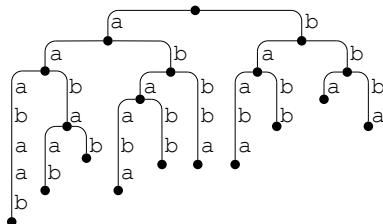
For the sake of simplicity in presentation, we here present a simplified version of our algorithm using an $O(nm)$ space data structure which answers queries in $O(|p| + o)$ time, where o is the number of outputs. How to improve the data structure will be described in the next section.

Basically, we represent substrings of M_d as their loci in $ST(M_d)$. We note that although the strings αp in Step 1 may be represented as implicit nodes of $ST(M_d)$, using their loci does not affect the result of Step 2. The algorithm outline can then be rewritten as follows.

- Step 1.** Enumerate the loci v of αp in $ST(M_d)$ such that $\alpha \in \Sigma^*$, $\alpha p \in Pre(M_d)$ and $LRS(\alpha p) < |p|$.
- Step 2.** For each locus v obtained in Step 1, enumerate the loci of $x\beta$ in $ST(M_d)$ such that $\beta \in \Sigma^*$ and $x\beta \in M_d$, where x is the string represented by v in $ST(M_d)$.



■ **Figure 2** Illustration of Suffix tree $ST(M_2^R\$)$.



■ **Figure 3** Illustration of Suffix tree $ST(M_2)$.

4.1 Implementation of Step 1

We use the suffix trees $ST(M_d^R\$)$ for $d = 1, \dots, m$. We note that there is a natural one-to-one correspondence between the strings x in $Pre(M_d)$ and the leaves of $ST(M_d^R\$)$ representing $x^R\$$. We also note that for any $p \in \Sigma^+$, the strings in $Pre(M_d) \cap \Sigma^*p$ correspond to the leaves of the subtree rooted at the locus v of p^R in $ST(M_d^R\$)$. Of the leaves representing $x^R\$$, we have to select those satisfying $LRS(x) < |p|$.

In the running example, the leaves of the subtree rooted at the locus of $p^R = ba$ in $ST(M_2^R\$)$ represent the strings $ba\$$, $baa\$$, $baaa\$$, $baaba\$$, $baabaaa\$$, $babaa\$$, $babaaa\$$ (see Figure 2). Of the seven strings of the form $x^R\$$, the three strings $ba\$$, $baa\$$, $baaa\$$ satisfy the condition $LRS(x) < |p|$.

Define the function val from the nodes of $ST(M_d^R\$)$ to the integers by: For any node u of $ST(M_d^R\$)$, let $val(u) = LRS(x)$ if u is a leaf of $ST(M_d^R\$)$, and ∞ otherwise, where x is the string such that u represents x^R . By applying the FLT query technique, mentioned in Section 2.4, to the tree $ST(M_d^R\$)$ with val , we can compute the leaves of $ST(M_d^R\$)$ representing $(\alpha p)^R\$$ such that $\alpha \in \Sigma^*$, $\alpha p \in Pre(M_d)$ and $LRS(\alpha p) < |p|$. From such a leaf, we can obtain the locus of αp in $ST(M_d)$ in constant time by keeping pointers from the nodes u of $ST(M_d^R\$)$ to the loci of x in $ST(M_d)$, where x is the string such that x^R is represented by u in $ST(M_d^R\$)$.

4.2 Implementation of Step 2

We note that the locus of any string in M_d is a leaf of $ST(M_d)$ (see Figure 3). The outputs of Step 2 are thus the leaves u of the subtree rooted at v representing strings in M_d . Define the function val from the nodes of $ST(M_d)$ to the integers by: For any node u of $ST(M_d)$, let $val(u) = 0$ if u is a leaf and represents some string in M_d , and 1 otherwise. We again apply the FLT query technique to the tree $ST(M_d)$ with val , to enumerate the loci of $x\beta$ appropriately.

4.3 Query time and space requirement

In Step 1, computing the locus of p^R in $ST(M_d^R\$)$ takes $O(|p|)$ time. Each execution of the FLT query takes constant time in Steps 1 and 2. Thus the query time is $O(|p| + o)$, where o is the number of outputs. For $d = 1, \dots, m$, the suffix trees $ST(M_d)$ and $ST(M_d^R\$)$, and the relevant data structures for the FLT queries require $O(n)$ space. The total space of our data structure is $O(nm)$.

5 Space Efficient Implementation of Step 1

As seen in Section 4.3, the use of the suffix trees $ST(M_d^R\$)$ for $d = 1, \dots, m$ in Step 1 causes the $O(nm)$ space requirement. Our idea to reduce the space requirement is to substitute $ST(M_d^R)$ for $ST(M_d^R\$)$. The following lemma gives an upper bound on the total size of suffix trees $ST(M_d^R)$.

► **Lemma 12.** *The suffix trees $ST(M_d)$ for $d = 1, \dots, m$ are, respectively, of size $O(n/d)$, and their total size is $O(n \log m)$.*

Proof. It suffices to show that $ST(M_d)$ has $O(n/d)$ leaves. Let v be any leaf of $ST(M_d)$, and let x be the string represented by v . Assume, for a contradiction, that x is not d -right maximal. Then, there exists a string $\beta \in \Sigma^+$ such that $\alpha x \beta \in M_d$ for some $\alpha \in \Sigma^*$. Thus $x\beta$ is a suffix of M_d , which contradicts that v is a leaf of $ST(M_d)$. Therefore x is d -right maximal. The number of leaves of $ST(M_d)$ is not greater than the number of d -right maximal strings, which is $O(n/d)$ by Lemma 2. ◀

The difficulty in using not $ST(M_d^R\$)$ but $ST(M_d^R)$ is that the string $(\alpha p)^R$ is possibly implicit in $ST(M_d^R)$ whereas the string $(\alpha p)^R\$$ is necessarily explicit and represented by a leaf in $ST(M_d^R\$)$. We partition Step 1 into two parts:

Step 1A. Enumerate the loci of αp in $ST(M_d)$ such that $\alpha \in \Sigma^*$, $\alpha p \in \text{Pre}(M_d)$, $\text{LRS}(\alpha p) < |p|$ and $(\alpha p)^R$ is explicit in $ST(M_d^R)$.

Step 1B. Enumerate the loci of αp in $ST(M_d)$ such that $\alpha \in \Sigma^*$, $\alpha p \in \text{Pre}(M_d)$, $\text{LRS}(\alpha p) < |p|$ and $(\alpha p)^R$ is implicit in $ST(M_d^R)$.

Step 1A can be done in $O(|p| + o)$ time with $O(n \log m)$ space in almost the same way as Section 4.1. Below we describe how to implement Step 1B.

5.1 Implementation of Step 1B

► **Lemma 13.** *For any string x in $\text{Pre}(M_d)$, x^R is explicit in $ST(D^R\$)$.*

Proof. Let $\beta \in \Sigma^*$ be a string such that $x\beta \in M_d$. Since the string $(x\beta)^R$ is d -left-right maximal, it is explicit in $ST(D^R\$)$ and therefore its suffix x^R is also explicit in $ST(D^R\$)$. ◀

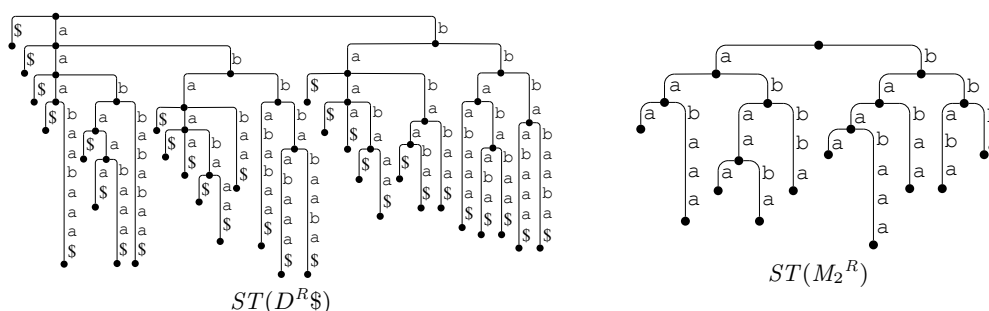
We thus use $ST(D^R\$)$ to represent strings in $\text{Pre}(M_d)$.

Let q_1 and q_2 be the strings represented by the loci of p^R in $ST(D^R\$)$ and in $ST(M_d^R)$, respectively. The p -critical path of $ST(D^R\$)$ is the path from u_1 to u_2 such that u_1 and u_2 are the nodes of $ST(D^R\$)$ representing q_1 and q_2 , respectively. A string x and the node representing x^R in $ST(D^R\$)$ are said to be p -satisfying if x is a left extension of p such that $x \in \text{Pre}(M_d)$, $\text{LRS}(x) < |p|$ and x^R is implicit in $ST(M_d^R)$. An edge e of $ST(M_d^R)$ and the path corresponding to e in $ST(D^R\$)$ are said to be p -admissible if e is in the subtree rooted at the node representing q_2 and at least one implicit node is present on e which represents the reversal x^R of a p -satisfying string x .

Every p -satisfying node of $ST(D^R\$)$ is present on: (i) the p -critical path of $ST(D^R\$)$ or (ii) a p -admissible path of $ST(M_d^R)$. Thus, the enumeration of the loci of αp in $ST(M_d)$ can be performed as follows.

- (1) Enumerate the p -admissible paths of $ST(D^R\$)$.
- (2) For each p -admissible path of $ST(D^R\$)$ and for the p -critical path of $ST(D^R\$)$, enumerate the p -satisfying nodes on it.
- (3) For each p -satisfying node of $ST(D^R\$)$ representing x^R , compute the locus of x in $ST(M_d)$.

► **Example 14.** Suppose that $D = \{aaabaabaaa, aaabaabbba, aabababbaa, abaababbba\}$, $d = 2$ and $p = abab$. Then, $q_1 = baba$ and $q_2 = babaaa$ (see Figure 4). We want to compute $baba$ in Step 1B.



■ **Figure 4** $ST(D^R\$)$ and $ST(M_2^R)$ for $D = \{aaabaabaaa, aaabaabbba, aabababbaa, abaababbba\}$.

In (1), we shall enumerate all p -admissible edges of $ST(M_d^R)$. With each edge (s, t) of $ST(M_d^R)$, we associate the value $LRS(z^R)$ such that x and y are the strings represented by s and t , respectively, and $z = y[.i]$ where i is the smallest integer in $[|x| + 1, |y| - 1]$ with $z \in Suf(M_d^R)$ (i.e. $z^R \in Pre(M_d)$). We associate ∞ with it, if no such i exists. Then, we can enumerate all p -admissible edges of $ST(M_d^R)$, by applying the FLT query technique to $ST(M_d^R)$, with regarding the value associated with the incoming edge (s, t) of a node t as the value of t . Computing the loci of p^R in $ST(M_d^R\$)$ and $ST(D^R\$)$ takes $O(|p|)$ time. Execution of the FLT query takes constant time.

In (2), we proceed to examine nodes representing x^R on the path until we encounter a node representing x^R with $LRS(x) \geq |p|$, by repeatedly querying with the data structure stated in the following lemma.

► **Lemma 15.** *There exists an $O(n \log m)$ size data structure which, given a node of $ST(D^R\$)$ representing string y^R , returns in $O(\log \log m)$ time the locus of $(xy)^R$ in $ST(D^R\$)$ such that x is the shortest string with $xy \in Pre(M_d)$, and nil if no such x exists.*

In (3), for each p -satisfying node of $ST(D^R\$)$ representing x^R , compute the locus of x in $ST(M_d)$ by using the data structure stated in the following lemma.

► **Lemma 16.** *The locus of a string x in $ST(M_d)$ can be computed in $O(\log \log m)$ time from the locus of x^R in $ST(D^R\$)$ using an $O(n \log m)$ space data structure.*

The suffix tree $ST(D^R\$)$ takes $O(n)$ space. For $d = 1, \dots, m$, the suffix trees $ST(M_d)$ and $ST(M_d^R\$)$, and the relevant data structures for the FLT queries require $O(n)$ space. The total computation time of Step 1B is $O(|p| + o \log \log m)$ and the total space of our data structure is $O(n \log m)$.

5.2 Proofs of Lemmas 15 and 16

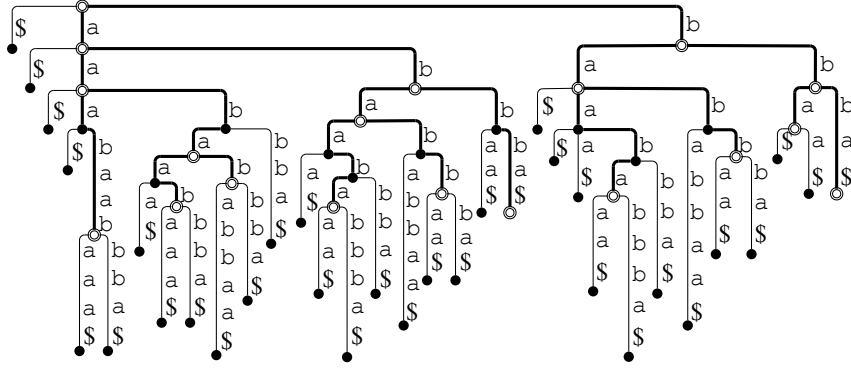
To complete the proof of Theorem 9, we give proofs of Lemmas 15 and 16. For the sake of convenience, we first prove Lemma 16.

5.2.1 Proof of Lemma 16

From the locus of x^R in $ST(D^R\$)$ we can obtain the locus of x in $ST(D\$)$ in constant time by using direct links from the nodes of $ST(D^R\$)$ to the corresponding nodes of $ST(D\$)$. Thus we describe how to compute from the locus of x in $ST(D\$)$ the locus of x in $ST(M_d)$ in $O(\log \log m)$ time using $O(n \log m)$ space.

A node v of $ST(D\$)$ representing string z is called a d -node if z is explicit in $ST(M_d)$. The locus of x in $ST(M_d)$ then corresponds to the earliest d -node preceded by the locus of x in the pre-order traversal of $ST(D\$)$.

► **Example 17.** Suppose that $D = \{\text{aaabaabaaa}, \text{aaabaabbba}, \text{aabababbaa}, \text{abaababbba}\}$, $d = 2$ and $x = \text{aab}$. Then the earliest 2-node preceded by the locus of x is the node representing **aaba** (see Figure 5). The locus of x in $ST(M_2)$ represents the same string **aaba**.



■ **Figure 5** Illustration of $ST(D\$)$, where the double lined circles represent the 2-nodes.

For any node s of $ST(D\$)$, let $L(s)$ be the sequence of non-negative integers d arranged in the increasing order such that $d = 0$ or s is a d -node. Let A be the sequence obtained by concatenating $L(s)$ according to the pre-order of nodes s of $ST(D\$)$. Let u and v be the loci of x in $ST(D\$)$ and $ST(M_d)$, respectively. Then v corresponds to the leftmost occurrence of d in $A[i+1..]$ such that i is the position of j -th occurrence of 0 where j is the rank of u in the pre-order traversal of $ST(D\$)$. Thus v can be computed from u as follows. For the rightmost leaf l_u of the subtree rooted at u , $v = nil$ if $\text{rank}_A(d, \text{select}_A(0, \text{PreOrd}(u))) > \text{select}_A(0, \text{PreOrd}(l_u))$, and otherwise, v corresponds to $A[\text{rank}_A(d, \text{select}_A(0, \text{PreOrd}(u)))]$, where $\text{PreOrd}(s)$ denotes the rank of a node s in the pre-order traversal of $ST(D\$)$.

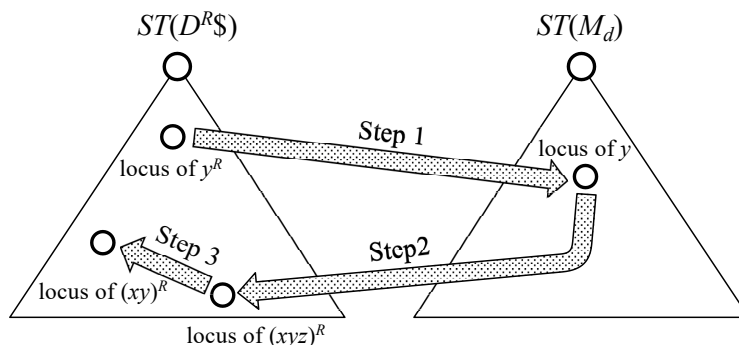
The numbers of 0's and d 's in the array A are $O(n)$ and $O(n/d)$, respectively, and hence we have $|A| = O(n \log m)$. By Lemma 5, we can compute the locus of x in $ST(M_d)$ from the locus of x in $ST(D^R\$)$ in $O(\log \log m)$ time using $O(n \log m)$ space.

5.2.2 Proof of Lemma 15

By Lemma 3, $xyz \in M_d$ implies that yz is d -right maximal. For each d -right maximal string β , let $\text{len}(\beta)$ denote the length of the shortest string α with $\alpha\beta \in M_d$. Then the desired string xy can be obtained from the d -right maximal extension yz of y that minimizes $\text{len}(yz)$.

From the locus of y^R in $ST(D^R\$)$, the locus of $(xy)^R$ in $ST(D^R\$)$ can be computed in three steps (see Figure 6).

- Step 1.** From the locus of y^R in $ST(D^R\$)$, find the locus of y in $ST(M_d)$.
- Step 2.** From the locus of y in $ST(M_d)$, find the locus of $(xyz)^R$ in $ST(D^R\$)$ such that x is one of the shortest strings x satisfying $xyz \in M_d$ for some string z .
- Step 3.** From the locus of $(xyz)^R$, find the locus of $(xy)^R$ in $ST(D^R\$)$.



■ **Figure 6** Computing the locus of $(xy)^R$ from the locus of y^R in $ST(D^R\$)$.

Step 1 requires $O(\log \log m)$ time by using the $O(n \log m)$ -size data structure stated in Lemma 16.

For Step 2, we define two functions len and $link$ on the set of nodes of $ST(M_d)$ as follows: For any node u of $ST(M_d)$, let β be the string represented by u . If there is some string α such that $\alpha\beta \in Pre(M_d)$, choose α as short as possible, and let $len(u) = |\alpha|$ and let $link(u)$ be the locus of α in $ST(D^R\$)$. If there is no such α , let $len(u) = \infty$ and $link(u) = nil$.

Suppose that v is the descendant of the locus of y in $ST(M_d)$ that minimizes $len(v)$. Then $len(v) = |x|$ and $link(v)$ is the locus of $(xyz)^R$ in $ST(D^R\$)$ since yz is d -right maximal. The locus of $(xyz)^R$ in $ST(D^R)$ can then be computed in constant time by storing the values $len(u)$ and $link(u)$ into the nodes u of $ST(M_d)$ and applying the Range Minimum Query technique.

In Step 3, the locus of $(xy)^R$ is obtained from the locus of $(xyz)^R$ in $ST(D^R\$)$ by traversing suffix links $|x|$ times. The task can be done in constant time by using the $O(n)$ space data structure for the level ancestor query [1] on suffix link tree of $ST(D^R\$)$.

Step 1 through Step 3 can be done in $O(\log \log m)$ time using $O(n \log m)$ space.

6 Conclusion

In this paper, we addressed the left-right maximal generic words problem and developed an $O(n \log m)$ size data structure, which answers queries in $O(|p| + o \log \log m)$ time, where o is the size of outputs. Our method is better than the previous work by Nishimoto et al. [7] both in the space requirement and in the query time. We achieved the $O(n \log m)$ space requirement by substituting $ST(M_d)$ for $ST(M_d\$)$, with the conjecture that the total size of $ST(M_d\$)$'s for $d = 1, \dots, m$ are $\Theta(nm)$. To prove that the total size is $\Omega(nm)$ is left as future work.

References

- 1 Omer Berkman and Uzi Vishkin. Finding Level-Ancestors in Trees. *J. Comput. Syst. Sci.*, 48(2):214–230, 1994.
- 2 Sudip Biswas, Manish Patil, Rahul Shah, and Sharma V. Thankachan. Succinct Indexes for Reporting Discriminating and Generic Words. In Edleno Silva de Moura and Maxime Crochemore, editors, *String Processing and Information Retrieval - 21st International Symposium, SPIRE 2014, Ouro Preto, Brazil, October 20-22, 2014. Proceedings*, volume 8799 of *Lecture Notes in Computer Science*, pages 89–100. Springer, 2014. doi:10.1007/978-3-319-11918-2.
- 3 Pawel Gawrychowski, Gregory Kucherov, Yakov Nekrich, and Tatiana A. Starikovskaya. Minimal Discriminating Words Problem Revisited. 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 129–140. Springer, 2013. doi:10.1007/978-3-319-02432-5.
- 4 Alexander Golynski, J. Ian Munro, and S. Srinivasa Rao. Rank/select operations on large alphabets: a tool for text indexing. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 368–373. ACM Press, 2006. URL: <http://dl.acm.org/citation.cfm?id=1109557.1109599>.
- 5 Gregory Kucherov, Yakov Nekrich, and Tatiana A. Starikovskaya. Computing Discriminating and Generic Words. In Liliana Calderón-Benavides, Cristina N. González-Caro, Edgar Chávez, and Nivio Ziviani, editors, *String Processing and Information Retrieval - 19th International Symposium, SPIRE 2012, Cartagena de Indias, Colombia, October 21-25, 2012. Proceedings*, volume 7608 of *Lecture Notes in Computer Science*, pages 307–317. Springer, 2012. doi:10.1007/978-3-642-34109-0.
- 6 S. Muthukrishnan. Efficient algorithms for document retrieval problems. In David Eppstein, editor, *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA.*, pages 657–666. ACM/SIAM, 2002. URL: <http://dl.acm.org/citation.cfm?id=545381.545469>.
- 7 Takaaki Nishimoto, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Computing Left-Right Maximal Generic Words. In Jan Holub and Jan Zdárek, editors, *Proceedings of the Prague Stringology Conference 2015, Prague, Czech Republic, August 24-26, 2015*, pages 5–16. Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, 2015. URL: <http://www.stringology.org/event/2015/p02.html>.
- 8 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, 1973.

Parameterized Complexity Classification of Deletion to List Matrix-Partition for Low-Order Matrices

Akanksha Agrawal

Ben-Gurion University of the Negev, Beer-Sheva, Israel
agrawal@post.bgu.ac.il

Sudeshna Kolay

Ben-Gurion University of the Negev, Beer-Sheva, Israel
sudeshna@post.bgu.ac.il

Jayakrishnan Madathil

The Institute of Mathematical Sciences, HBNI, Chennai, India
jayakrishnanm@imsc.res.in

Saket Saurabh

University of Bergen, Bergen, Norway
The Institute of Mathematical Sciences, HBNI, Chennai, India
saket@imsc.res.in

Abstract

Given a symmetric $\ell \times \ell$ matrix $M = (m_{i,j})$ with entries in $\{0, 1, *\}$, a graph G and a function $L : V(G) \rightarrow 2^{[\ell]}$ (where $[\ell] = \{1, 2, \dots, \ell\}$), a list M -partition of G with respect to L is a partition of $V(G)$ into ℓ parts, say, V_1, V_2, \dots, V_ℓ such that for each $i, j \in \{1, 2, \dots, \ell\}$, (i) if $m_{i,j} = 0$ then for any $u \in V_i$ and $v \in V_j$, $uv \notin E(G)$, (ii) if $m_{i,j} = 1$ then for any (distinct) $u \in V_i$ and $v \in V_j$, $uv \in E(G)$, (iii) for each $v \in V(G)$, if $v \in V_i$ then $i \in L(v)$. We consider the DELETION TO LIST M -PARTITION problem that takes as input a graph G , a list function $L : V(G) \rightarrow 2^{[\ell]}$ and a positive integer k . The aim is to determine whether there is a k -sized set $S \subseteq V(G)$ such that $G - S$ has a list M -partition. Many important problems like VERTEX COVER, ODD CYCLE TRANSVERSAL, SPLIT VERTEX DELETION, MULTIWAY CUT and DELETION TO LIST HOMOMORPHISM are special cases of the DELETION TO LIST M -PARTITION problem. In this paper, we provide a classification of the parameterized complexity of DELETION TO LIST M -PARTITION, parameterized by k , (a) when M is of order at most 3, and (b) when M is of order 4 with all diagonal entries belonging to $\{0, 1\}$.

2012 ACM Subject Classification Theory of computation \rightarrow Design and analysis of algorithms; Theory of computation \rightarrow Fixed parameter tractability

Keywords and phrases list matrix partitions, parameterized classification, Almost 2-SAT, important separators, iterative compression

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.41

Funding *Akanksha Agrawal*: Supported by the PBC Program of Fellowships for Outstanding Post-doctoral Researchers from China and India (No. 5101479000).

Saket Saurabh: Supported by the Horizon 2020 Framework program, ERC Consolidator Grant LOPPRE (No. 819416).

1 Introduction

A large number of problems in algorithmic graph theory are of the following two types. (1) Given a graph G , can the vertices of G be partitioned subject to a set of constraints? And (2) given a graph G and a non-negative integer k , is it possible to delete at most k vertices from G so that the vertices of the resulting graph can be partitioned subject to a set of



© Akanksha Agrawal, Sudeshna Kolay, Jayakrishnan Madathil, and Saket Saurabh; licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 41; pp. 41:1–41:14

Leibniz International Proceedings in Informatics



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

constraints? In this paper, we study the parameterized complexity of a family of problems of the second type, (with k , the size of the deletion set, as the parameter). We consider those partitions that can be characterised by a matrix of order at most 4. In this regard, we follow Feder et al. [17], who undertook a similar study of partition problems of the first type.

Let M be a symmetric $\ell \times \ell$ matrix with entries from $\{0, 1, *\}$. For a graph G , an M -partition \mathcal{V} of G is a partition of $V(G)$ into ℓ parts $\{V_1, V_2, \dots, V_\ell\}$ (where some part could be empty) such that for every $i \in [\ell]$, (i) V_i is an independent set if $m_{i,i} = 0$, (ii) $G[V_i]$ is a clique if $m_{i,i} = 1$ (and no restriction on V_i if $m_{i,i} = *$); and for distinct indices $i, j \in [\ell]$, (iii) V_i and V_j are completely adjacent if $m_{i,j} = 1$, (iv) V_i and V_j are completely non-adjacent if $m_{i,j} = 0$ (and no restriction on the edges between V_i and V_j if $m_{i,j} = *$). The M -PARTITION problem takes as input a graph G , and the objective is to determine if G admits an M -partition. This problem encompasses recognition of many graph classes that can be characterised by a partition of the vertex set satisfying certain constraints. For instance, consider the following matrices:

$$M_1 = \begin{pmatrix} 0 & * \\ * & 0 \end{pmatrix} \quad M_2 = \begin{pmatrix} 0 & * \\ * & 1 \end{pmatrix} \quad M_\ell = \begin{pmatrix} 0 & * & \cdots & * \\ * & 0 & \cdots & * \\ \vdots & \vdots & \ddots & \vdots \\ * & * & \cdots & 0 \end{pmatrix}_{\ell \times \ell}$$

The set of graphs that admit an M_1 -partition and M_2 -partition are exactly the family of bipartite graphs and split graphs, respectively. Note that both bipartite graphs and split graphs have polynomial time recognition algorithms [3, 22, 25]. The graphs that admit an M_ℓ -partition are exactly the graphs that admit a proper colouring using at most ℓ colours. It is well-known that while 2-colouring is polynomial time solvable [3], ℓ -colouring is NP-hard for every $\ell \geq 3$ [19, 20].

For an $\ell \times \ell$ matrix M , LIST M -PARTITION is a generalization of M -PARTITION. Let M be a symmetric $\ell \times \ell$ matrix over $\{0, 1, *\}$. Given a graph G and a function $L : V(G) \rightarrow 2^{[\ell]}$ (L is called a *list function*, and for each $v \in V(G)$, $L(v)$ is called the *list* of v), a *list M -partition of G with respect to L* (or a list M -partition of G that respects L) is an M -partition $\mathcal{V} = \{V_1, V_2, \dots, V_\ell\}$ of G such that for each $v \in V(G)$, if $v \in V_i$ for some $i \in [\ell]$ then $i \in L(v)$. LIST M -PARTITION takes as input a graph G , a list function $L : V(G) \rightarrow 2^{[\ell]}$, and the objective is to determine if G admits a list M -partition of G that respects L . Note that when an instance of LIST M -PARTITION has the input list function mapping every vertex to the set $[\ell]$, then it is also an instance of M -PARTITION (where we forget the list function).

Both M -PARTITION and LIST M -PARTITION problems have been extensively studied in the literature, both for restricted matrices and for restricted graph classes (see, for example, [1, 5, 8, 10, 11, 12, 16, 17, 24, 29] and references therein). The most widely known special case of LIST M -PARTITION is perhaps the LIST COLOURING problem. The notion of studying the restriction of colouring problems with a list of allowed colours on vertices was introduced and studied (independently) by Vizing [28] and Erdős et al. [13]. Since its advent, the problem has been extensively studied in both graph theory and algorithms [13, 24, 29]. Another important special case of LIST M -PARTITION is the LIST HOMOMORPHISM problem, which to the best of our knowledge, was introduced by Feder and Hell [14], and has been extensively studied in the literature [1, 8, 10, 11, 12, 16].

Feder et al. [17] studied LIST M -PARTITION, and established a complete classification of LIST M -PARTITION for small matrices M (into P/NP-hard/quasi polynomial time). Their results form a special case of later results due to Feder and Hell [15, Corollary 3.4] on

constraint satisfaction problems. In this paper, we look at the deletion version of LIST M -PARTITION, which we call DELETION TO LIST M -PARTITION. The problem is formally defined as follows.

DELETION TO LIST M -PARTITION **Parameter:** k
Input: A graph G , a list function $L : V(G) \rightarrow 2^{[\ell]}$ where ℓ is the order of M , and a non-negative integer k .
Question: Does there exist $X \subseteq V(G)$ such that $|X| \leq k$ and $G - X$ admits a list M -partition that respects L ?

The DELETION TO LIST M -PARTITION problem generalises many well-studied classical problems, such as VERTEX COVER (VC), ODD CYCLE TRANSVERSAL (OCT), SPLIT VERTEX DELETION (SVD) and MULTIWAY CUT, to name a few. These problems (VC, OCT, SVD etc.) have been studied in both classical and parameterized complexity settings and are all NP-hard [30, 9]. Chitnis et al. [7] initiated the study of the deletion version of LIST HOMOMORPHISM to a graph H , called DL-HOM(H), which is a special case of DELETION TO LIST M -PARTITION. (In [7], H is considered to be a loopless, simple graph.) They showed that DL-HOM(H) is FPT (parameterized by k and $|H|$) for any (P_6, C_6) -free bipartite graph H , and conjectured that the problem is FPT for those graphs H for which LIST HOMOMORPHISM (i.e., without deletions) is polynomial-time solvable. Notice that for some of the matrices that do not contain both 1 and 0 and do not have a $*$ as a diagonal entry, the corresponding DELETION TO LIST M -PARTITION problem is covered by the results in [7].¹ While we study the deletion version of LIST M -PARTITION, the ‘‘counting version,’’ denoted by #LIST M -PARTITION, where given G and L as input, the goal is to determine the number of M -partitions of G that respect L , has been studied by Göbel et al. [21]. They established a complete dichotomy by showing that for any symmetric matrix M over $\{0, 1, *\}$, #LIST M -PARTITION is either in FP or #P-complete.

Our Results and Methods

We study the parameterized complexity of DELETION TO LIST M -PARTITION for different matrices M , and obtain a classification of these problems (into polynomial time solvable, NP-hard and FPT or para-NP-hard) when M is a matrix of order at most 4.

M is of order at most 3. First, we resolve the classical complexity of DELETION TO LIST M -PARTITION problems when M is of order at most 3, except for one matrix. We extend the study to explore the parameterized complexity of these deletion problems, parameterized by the size k of the deletion set. Specifically, we prove the following theorem.

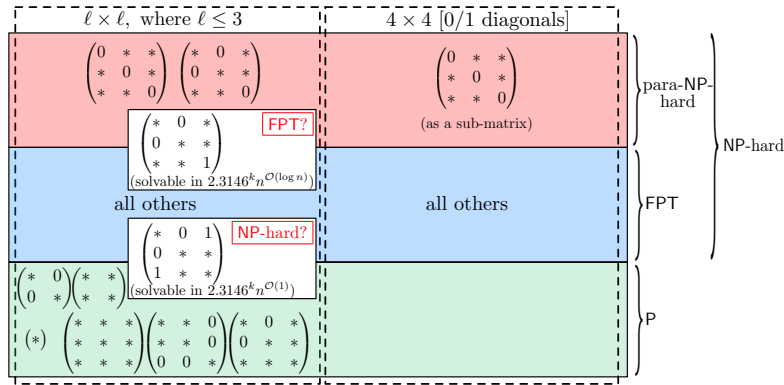
► **Theorem 1** (\star^2). *For a 3×3 symmetric matrix M over $\{0, 1, *\}$, the DELETION TO LIST M -PARTITION problem is*

1. *polynomial time solvable if either M or \overline{M} is equivalent to one of the three matrices*

$$\begin{pmatrix} * & * & * \\ * & * & * \\ * & * & * \end{pmatrix}, \begin{pmatrix} * & 0 & * \\ 0 & * & * \\ * & * & * \end{pmatrix} \text{ or } \begin{pmatrix} * & * & 0 \\ * & * & 0 \\ 0 & 0 & * \end{pmatrix};$$

¹ The results in [7] only cover a few matrices of the described form because there is an additional constraint of H being a (P_6, C_6) -free bipartite graph.

² Due to paucity of space, (full) proofs of statements marked with a \star have been omitted.



■ **Figure 1** An overview of the results on matrices, up to complementation and equivalence.

2. *para-NP-hard if either M or \overline{M} is equivalent to one of the two matrices*

$$\begin{pmatrix} 0 & * & * \\ * & 0 & * \\ * & * & 0 \end{pmatrix} \text{ or } \begin{pmatrix} * & 0 & * \\ 0 & * & * \\ * & * & 0 \end{pmatrix};$$

3. *solvable by an algorithm running in time $2.3146^k n^{O(\log n)}$, when M or \overline{M} is equivalent to*

$$\begin{pmatrix} * & 0 & * \\ 0 & * & * \\ * & * & 1 \end{pmatrix};$$

4. *FPT if M is not covered by any of the previous cases.*

Theorem 1 shows that the DELETION TO LIST M -PARTITION problem has polynomial time algorithms, FPT algorithms or algorithms of the form $c^k n^{O(\log n)}$, depending on the matrix M (see also Figure 1). We also have two cases for M where the DELETION TO LIST M -PARTITION problem is para-NP-hard. With such a varied range of complexities, we use several techniques to design the algorithms. The polynomial time algorithms are based on ideas that help to reduce the problem to an equivalent minimum separator problem [18]. For our FPT algorithms, we utilize the notion of important separators, which was introduced by Marx [27]. Another technique we use to design our FPT algorithms for DELETION TO LIST M -PARTITION is based on reducing the given instance to (polynomially many) instances of VARIABLE DELETION ALMOST 2-SAT, and then employing the known FPT algorithm for VARIABLE DELETION ALMOST 2-SAT to resolve the instance. We also use the technique of “iterative compression” for designing FPT algorithm for one of our cases. For the matrix M defined in item 3 of Theorem 1, although the FPT solvability versus W-hardness of DELETION TO LIST M -PARTITION remains open, we design an algorithm for this problem that runs in time $2.3146^k n^{O(\log n)}$, where n is the number of vertices in the input graph. We use the technique of separating families introduced by Feder et al. [17] to design this algorithm. These results can be found in Section 3.

M is of order 4. We restrict the matrices M to have only 0s and 1s as their diagonal entries. First, we observe that all these problems are NP-hard. Second, we design FPT algorithms for these problems, unless the matrix M “encompasses” the 3-COLOURING problem. Let M_{col} denote the 3×3 matrix with only 0s on the diagonal, and *s elsewhere. We prove the following theorem.

► **Theorem 2** (*). *Consider a DELETION TO LIST M -PARTITION problem, where M is a 4×4 matrix over $\{0, 1, *\}$ that has only 0s and 1s as diagonal entries. If M does not contain an equivalent matrix of M_{col} (or its complement) as a sub-matrix, then the problem is FPT. Otherwise, the problem is para-NP-hard.*

Our FPT algorithms use the technique of iterative compression along with the known FPT algorithm for VARIABLE DELETION ALMOST 2-SAT. Our use of iterative compression exploits structural properties provided by M in order to design the FPT algorithms. Our results, in particular show that, for a 3×3 matrix (except the matrix defined in item 3 of Theorem 1) or a 4×4 matrix with no *s on the diagonal, whenever the LIST M -PARTITION is polynomial time solvable, the corresponding DELETION TO LIST M -PARTITION problem is fixed-parameter tractable, and whenever LIST M -PARTITION is NP-hard, DELETION TO LIST M -PARTITION is para-NP-hard, and thus provide a (partial) parameterized analogue of the results established in [17].

2 Preliminaries and Basic Tools

For a graph G , $V(G)$ and $E(G)$ denote respectively the vertex set and edge set of G . Given a partition \mathcal{V} of $V' \subseteq V(G)$, $V(\mathcal{V}) = V'$, and for $X \subseteq V'$, $\mathcal{V} - X$ denotes the restriction of the partition to $V' \setminus X$. Let G be a graph and $X, Y \subseteq V(G)$. A set of vertices $S \subseteq V(G)$ is said to be an (X, Y) -separator if $G - S$ contains no path from X to Y .

Definitions and results for some useful problems. Consider a 2-CNF formula ψ . The variable set of ψ is denoted by $\text{Var}(\psi)$. For a set $Y \subseteq \text{Var}(\psi)$, $\psi - Y$ denotes the 2-CNF formula obtained from ψ by deleting all the clauses that contain a variable from Y . The 2-SAT problem takes as input a 2-CNF formula ψ , and the question is to test if there is a satisfying assignment for ψ . The 2-SAT problem admits a polynomial time algorithm [23]. The VARIABLE DELETION ALMOST 2-SAT problem takes as input a 2-CNF formula ψ and a non-negative integer k , and the objective is to test if there is a set $X \subseteq \text{Var}(\psi)$ of size at most k such that $\psi - X$ is satisfiable. It is known that VARIABLE DELETION ALMOST 2-SAT admits an algorithm that runs in time $2.3146^k n^{\mathcal{O}(1)}$, where n is the number of variables [26], and hence is in FPT, when parameterized by k .

Matrices and list partitioning. For an $\ell \times \ell$ matrix M , consider an M -partition $\mathcal{V} = \{V_1, V_2, \dots, V_\ell\}$ of a graph G . Then the part V_i will be said to have index i . For an instance $(G, L : V(G) \rightarrow 2^{[\ell]})$ of LIST M -PARTITION, throughout the paper we assume that $L(v) \neq \emptyset$, as otherwise, we can immediately report that (G, L) does not admit an M -partition that respects L . Similarly, for an instance $(G, L : V(G) \rightarrow 2^{[\ell]}, k)$ of DELETION TO LIST M -PARTITION, we assume that $L(v) \neq \emptyset$, as otherwise, we can (safely) delete such a vertex from G and reduce k by 1 (or return that it is a no-instance when $k \leq 0$).

Given a matrix M , the complement \overline{M} is defined as follows: $m_{i,j} = 0 \iff \overline{m}_{i,j} = 1$, $m_{i,j} = 1 \iff \overline{m}_{i,j} = 0$, $m_{i,j} = * \iff \overline{m}_{i,j} = *$. The lower triangular submatrix $M_L = (m_{i,j}^L)$ of a matrix $M = (m_{i,j})$ is defined as follows: $\forall i \geq j, m_{i,j}^L = m_{i,j}$ and $\forall i < j, m_{i,j}^L = 0$. When the context is clear we drop the superscript from the entry names of the lower triangular matrix M_L and simply use the entry names $m_{i,j}$ of M . Similarly, the upper triangular submatrix $M_U = (m_{i,j}^U)$ of a matrix $M = (m_{i,j})$ is defined as follows: $\forall i \leq j, m_{i,j}^U = m_{i,j}$ and $\forall i > j, m_{i,j}^U = 0$. Again, we drop superscripts when the context is clear. The following observation follows from the definition of the complement of a matrix.

► **Observation 3.** *A graph G admits a list M -partition with respect to a list function L if and only if \overline{G} admits a list \overline{M} -partition with respect to L .*

In this paper, for an $\ell \times \ell$ matrix M , a submatrix M' of order $p \leq \ell$ is defined as follows: there are p distinct indices $\{i_1, i_2, \dots, i_p\} \in [\ell]$ such that $m'_{a,b} = m_{i_a, i_b}$ for all $a, b \in [p]$. Consider two symmetric $\ell \times \ell$ matrices $M = (m_{i,j})$ and $M' = (m'_{i,j})$ over $\{0, 1, *\}$. We say that M is *equivalent* to M' , if there is a permutation $\pi : [\ell] \rightarrow [\ell]$ such that $m'_{i,j} = m_{\pi(i), \pi(j)}$. And such a permutation π is called a *witness-permutation* for (M, M') . Notice that if M is equivalent to M' , then M' is also equivalent to M with witness-permutation π^{-1} . That is, M and M' are equivalent if they define the same partition up to a re-indexing of the parts. We immediately obtain the following result.

► **Proposition 4.** *Let M be a symmetric matrix equivalent to M' , with a witness-permutation π . Then, a graph G admits a list M -partition with respect to a list function L if and only if G admits a list M' -partition with respect to the list function L' , where $L'(v) = \{\pi(i) \mid i \in L(v)\}$ for every $v \in V(G)$.*

Some Useful Results on List M -Partition

We present a summary of results from [17], which will be used throughout.

Reducing List M -partition to 2-SAT, for a 2×2 matrix M . It was shown in [17] that an instance of the LIST M -PARTITION problem can be reduced (in polynomial time) to an equivalent instance of 2-SAT, provided that M is a 2×2 matrix. And since 2-SAT is polynomial time solvable [2], so is LIST M -PARTITION, when M is a 2×2 matrix. What is interesting is that this reduction works even if M is not of order 2, but the size of $L(v)$ is at most 2 for every vertex v of the input graph G . The LIST M -PARTITION problem such that the list of every vertex in G has size at most two will also be referred to as the 2-LIST M -PARTITION problem. The reduction from 2-LIST M -PARTITION to 2-SAT will also be useful while designing algorithms for DELETION TO LIST M -PARTITION. Next, we state the properties of the reduction from 2-LIST M -PARTITION to 2-SAT.

► **Proposition 5** (\star). *Let $(G, L : V(G) \rightarrow 2^{[\ell]})$ be an instance of 2-LIST M -PARTITION. In polynomial time we can output an instance ψ of 2-SAT and a bijective function $f : V(G) \rightarrow \text{Var}(\psi)$, such that for every $X \subseteq V(G)$: i) $\psi - f(X)$ is a yes instance of 2-SAT if and only if $(G - X, L|_{V(G-X)} : V(G - X) \rightarrow 2^{[\ell]})$ is a yes instance of 2-LIST M -PARTITION, and ii) a set $A \subseteq \text{Var}(\psi - f(X))$ is a satisfying assignment for $\psi - f(X)$ if and only if for each $v \in \{f^{-1}(a) \mid a \in A\}$ with $L|_{V(G-X)}(v) = \{i, j\}$, where $i \leq j$, we have $v \in V_i$. Here, $\mathcal{V} = \{V_1, V_2, \dots, V_\ell\}$ is an M -partition of $G - X$ (if it exists).*

From the above proposition, we can conclude that 2-LIST M -PARTITION admits a polynomial time algorithm. It also suggests a strategy for solving LIST M -PARTITION: try to reduce the size of every list. It is indeed possible to do that, as shown in [17], if the matrix M has a row that contains both a 0 and a 1 (see Proposition 2.3 and Corollary 2.4 in [17]).

► **Proposition 6** ([17]). *Suppose the matrix M has $m_{i_1, i_2} = 0$ and $m_{i_1, i_3} = 1$ (one of i_3 or i_2 can be equal to i_1). Then an instance (G, L) of the LIST M -PARTITION problem can be reduced (in polynomial time) to (i) one instance with no list containing i_1 and (ii) at most $|V(G)|$ instances with no list containing both i_2 and i_3 , such that (G, L) is a yes instance if and only if at least one of the $|V(G)| + 1$ instances is a yes instance.*

The above proposition comes handy when M is a 3×3 matrix, as in this case the problem reduces to $|V(G)| + 1$ instances that have only lists of size at most two. Another notion that will be useful in our algorithm is “domination”, defined below.

► **Definition 7.** For a symmetric matrix M of order ℓ over $\{0, 1, *\}$, and rows $i_1, i_2 \in [\ell]$, i_1 dominates i_2 in M if for each column $i_3 \in [\ell]$, either $m_{i_1 i_3} = m_{i_2 i_3}$ or $m_{i_1 i_3} = *$.

► **Proposition 8** (Proposition 2.5 [17]). Consider a matrix M of order ℓ , where the row i_1 dominates the row i_2 , and an instance (G, L) of LIST M -PARTITION. Let L' be the list function such that for each $v \in V(G)$, (i) $L'(v) = L(v)$ if $|L(v) \cap \{i_1, i_2\}| \leq 1$, and (ii) $L'(v) = L(v) \setminus \{i_2\}$ otherwise. The graph G admits a list M -partition that respects L if and only if it admits a list M -partition that respects L' .

Basic tools for Deletion to List M -Partition. We show how the results presented earlier for LIST M -PARTITION can be used for solving DELETION TO LIST M -PARTITION. Consider an instance (G, L, k) of DELETION TO LIST M -PARTITION. Suppose $|L(v)| \leq 2$ for every $v \in V(G)$. Let ψ be the 2-CNF formula and $f : V(G) \rightarrow \text{Var}(\psi)$ be the bijective function returned by Proposition 5. The properties of ψ and f (as in Proposition 5), ensures that deleting a set $X \subseteq V(G)$ of vertices from G is equivalent to deleting the variables $f(X)$ from ψ . Let DELETION TO 2-LIST M -PARTITION be the special case of DELETION TO LIST M -PARTITION where the list of each vertex has size at most two. By Proposition 5, DELETION TO 2-LIST M -PARTITION is equivalent to testing whether k variables can be deleted from ψ to make it satisfiable. This is exactly the same as the VARIABLE DELETION ALMOST 2-SAT problem, which admits an FPT algorithm running in time $2.3146^{k'} n'^{\mathcal{O}(1)}$, where k' is the size of the deletion set and n' is the number of variables. This together with Proposition 5 gives us the following result.

► **Proposition 9.** DELETION TO 2-LIST M -PARTITION is fixed-parameter tractable with running time $2.3146^k n^{\mathcal{O}(1)}$.

Now using Propositions 6 and 9, we obtain the following result.

► **Proposition 10.** Let M be a 3×3 matrix such that M has a row that contains both a 0 and a 1. Then DELETION TO LIST M -PARTITION is fixed-parameter tractable.

► **Proposition 11** (\star). DELETION TO LIST M -PARTITION is NP-hard if at least one of the diagonal entries of M is 0 or 1.

The reduction rule stated in the following lemma will be useful in our algorithms.

► **Lemma 12** (\star). For a matrix M , let (G, L, k) be an instance of DELETION TO LIST M -PARTITION, with a vertex $v \in V(G)$, such that $L(v) = \emptyset$. Then, (G, L, k) and $(G - \{v\}, L|_{V(G) \setminus \{v\}}, k - 1)$ are equivalent instances of DELETION TO LIST M -PARTITION.

► **Remark 13.** We note that for any DELETION TO LIST M -PARTITION problem, we assume that we are looking for a list M -partition where each part is non-empty. First, if there is a part that is empty in the list M -partition, then our current instance can be resolved as an instance of DELETION TO LIST M' -PARTITION, where M' is a matrix of order strictly less than M . Otherwise, for no list M -partition is any part empty. In such a case, in a polynomial time preprocessing step, we can guess one vertex v_i per part i of the hypothetical list M -partition \mathcal{V} and appropriately reduce $L(v_i) = \{i\}$.

■ **Table 1** An overview of our NP-hardness vs. P results (not covered by Proposition 11) for matrices of order 3×3 , where complement matrices are not shown.

Matrices (and witness-permutation $\pi : [3] \rightarrow [3]$) $\pi(1) = 1, \pi(2) = 3, \pi(3) = 2$ $\pi(1) = 3, \pi(2) = 2, \pi(3) = 1$		Equivalent matrix	Class	Proof
		$\begin{pmatrix} * & * & * \\ * & * & * \\ * & * & * \end{pmatrix}$	P	Lemma 16(a)
$\begin{pmatrix} * & * & 0 \\ * & * & * \\ 0 & * & * \end{pmatrix}$	$\begin{pmatrix} * & * & * \\ * & * & 0 \\ * & 0 & * \end{pmatrix}$	$\begin{pmatrix} * & 0 & * \\ 0 & * & * \\ * & * & * \end{pmatrix}$	P	Lemma 16(b)
$\begin{pmatrix} * & 0 & * \\ 0 & * & 0 \\ * & 0 & * \end{pmatrix}$	$\begin{pmatrix} * & 0 & 0 \\ 0 & * & * \\ 0 & * & * \end{pmatrix}$	$\begin{pmatrix} * & * & 0 \\ * & * & 0 \\ 0 & 0 & * \end{pmatrix}$	P	Lemma 16(c)
		$\begin{pmatrix} * & 0 & 0 \\ 0 & * & 0 \\ 0 & 0 & * \end{pmatrix}$	NP-hard	Lemma 14(a)
$\begin{pmatrix} * & 0 & 1 \\ 0 & * & 0 \\ 1 & 0 & * \end{pmatrix}$	$\begin{pmatrix} * & 0 & 0 \\ 0 & * & 1 \\ 0 & 1 & * \end{pmatrix}$	$\begin{pmatrix} * & 1 & 0 \\ 1 & * & 0 \\ 0 & 0 & * \end{pmatrix}$	NP-hard	Lemma 14(b)

3 Classification of 3×3 matrices

In this section, our main objective is to classify DELETION TO LIST M -PARTITION for matrices M of order 3, both in classical complexity as well as parameterized complexity. Throughout the section, $M = (m_{i,j})$ denotes a symmetric 3×3 matrix over $\{0, 1, *\}$. We prove some of the main algorithmic results that we obtain for DELETION TO LIST M -PARTITION for matrices M of order 3, and thus present a partial proof of Theorem 1.

Before we go into the proof of Theorem 1, we first address the question of NP-hardness versus polynomial time solvability of these problems. Already we saw in Proposition 11 that DELETION TO LIST M -PARTITION is NP-hard if at least one of the diagonal entries of M is 0 or 1. So we now need to consider only those matrices M for which $m_{1,1} = m_{2,2} = m_{3,3} = *$. And up to complementation and equivalence of matrices, we are left with only six such matrices. We resolve five of these cases; see Table 1 for an overview of these results.

► **Lemma 14** (*). DELETION TO LIST M -PARTITION is NP-hard if M is one of the following matrices: (a) $m_{1,1} = m_{2,2} = m_{3,3} = *$, $m_{1,2} = m_{1,3} = m_{2,3} = 0$, and (b) $m_{1,1} = m_{2,2} = m_{3,3} = *$, $m_{1,3} = m_{2,3} = 0$, $m_{1,2} = 1$.

► **Remark 15**. We do not know whether DELETION TO LIST M -PARTITION is NP-hard or not when M is described as $m_{1,1} = m_{2,2} = m_{3,3} = *$, $m_{1,2} = 0$, $m_{1,3} = 1$, $m_{2,3} = *$. However, in the course of the proof of Theorem 1, we show that DELETION TO LIST M -PARTITION is FPT, when M or \overline{M} is equivalent to the above matrix.

We now briefly discuss the polynomial time solvability of the cases described in Theorem 1, item 1.

► **Lemma 16** (*). DELETION TO LIST M -PARTITION is polynomial time solvable if M is one of the following matrices:

$$(a) \begin{pmatrix} * & * & * \\ * & * & * \\ * & * & * \end{pmatrix}, (b) \begin{pmatrix} * & 0 & * \\ 0 & * & * \\ * & * & * \end{pmatrix}, (c) \begin{pmatrix} * & * & 0 \\ * & * & 0 \\ 0 & 0 & * \end{pmatrix}.$$

Proof Sketch. In each of the cases below, (G, L, k) denotes an input instance of DELETION TO LIST M -PARTITION. We assume that for each $v \in V(G)$, we have $L(v) \neq \emptyset$ (see Lemma 12). And recall that for any $X, Y \subseteq V(G)$, an (X, Y) -separator can be found in polynomial time [18].

- (a) In this case, all entries of the matrix M are *. Notice that a vertex $v \in V(G)$ can go to any one of the parts V_i in a list M -partition as long as $i \in L(v)$.
- (b) Let $X = \{v \in V(G) \mid L(v) = \{1\}\}$, $Y = \{v \in V(G) \mid L(v) = \{2\}\}$ and $Z = \{v \in V(G) \mid 3 \in L(v)\}$. We can show that $S \subseteq V(G)$ is a solution for the DELETION TO LIST M -PARTITION instance (G, L, k) if and only if S is an (X, Y) -separator in $G - Z$.
- (c) In this case, indices 1 and 2 dominate each other. Thus we can assume that the list of no vertex contains both 1 and 2. Let $X = \{v \in V(G) \mid L(v) = \{3\}\}$ and $Y = \{x \in V(G) \mid L(x) \subseteq \{1, 2\}\}$. Notice that a set $S \subseteq V(G)$ is a solution to the DELETION TO LIST M -PARTITION instance (G, L, k) if and only if S is an (X, Y) -separator. ◀

Now we turn our attention to the parameterized complexity of DELETION TO LIST M -PARTITION when M is of order 3. First, we consider the matrices defined in item 2 of Theorem 1.

► **Observation 17.** *The problem DELETION TO LIST M -PARTITION is para-NP-hard, when M is one of the two matrices defined in item 2 of Theorem 1. The first matrix in item 2 corresponds to a 3-colouring and the second matrix corresponds to a stable-cutset partition. Therefore, the corresponding LIST M -PARTITION problems correspond to the problems 3-COLOURING and STABLE CUTSET, respectively, both of which are NP-hard [19, 4].*

In the remaining part of this section, we consider some special cases from Theorem 1 and describe algorithms for these special cases. The rest of the case analysis leading to the proof of Theorem 1 has been omitted due to space constraints.

Algorithm for the Deletion to List Clique-Cutset Partition. The problem is DELETION TO LIST M -PARTITION, where M is the matrix such that an M -partition is a clique-cutset partition, i.e., M is such that $m_{1,1} = m_{2,2} = *$, $m_{3,3} = 1$ and $m_{1,3} = m_{2,3} = *$. $m_{1,2} = 0$. Consider a clique-cutset partition $\mathcal{V} = \{V_1, V_2, V_3\}$ of a graph G . The subgraph $G[V_3]$ is a clique, and V_3 is also a cutset between the parts V_1 and V_2 . Hence the name clique-cutset partition. We now prove the following lemma, the proof of which relies on a family of vertex subsets that “separate cliques and stable-pairs,” defined below.

► **Lemma 18.** *DELETION TO LIST CLIQUE-CUTSET PARTITION is solvable in $2.3146^k n^{\mathcal{O}(\log n)}$ time.*

► **Definition 19.** *For a graph G , a pair of disjoint sets $A, B \subseteq V(G)$ is a stable-pair if for every $a \in A$ and $b \in B$, $ab \notin E(G)$. A family $\mathcal{F} \subseteq \{F \mid F \subseteq V(G)\}$ separates cliques and stable-pairs if for every $A, B, C \subseteq V(G)$ such that A, B is a stable-pair, $G[C]$ is a clique and $C \cap (A \cup B) = \emptyset$, there is $F \in \mathcal{F}$ such that $C \subseteq F$ and either $F \cap A = \emptyset$ or $F \cap B = \emptyset$.*

► **Proposition 20** (Theorem 4.3 [17]). *Every graph on n vertices has a family of size $n^{\log n}$ that separate cliques and stable-pairs. Moreover, such a family can be found in time $n^{\mathcal{O}(\log n)}$.*

Let us see the implication of this proposition to our problem. Consider an instance (G, L, k) of DELETION TO LIST CLIQUE-CUTSET PARTITION, and its solution $S \subseteq V(G)$ (if it exists). Let \mathcal{F} be a family of sets that separates cliques and stable-pairs in G . And consider the list

■ **Algorithm 3.1** Algorithm for DELETION TO LIST CLIQUE-CUTSET PARTITION.

Input: (G, L, k) .
Output: yes or no.

- 1 Construct a family \mathcal{F} of size $n^{\log n}$ that separates cliques and stable-pairs in G .
- 2 **for** each $F \in \mathcal{F}$ **do**
- 3 Define $L_1 : V(G) \rightarrow 2^{[3]}$ as follows. For every $v \in F$, set $L_1(v) = L(v) \setminus \{1\}$. For every $v \notin F$, set $L_1(v) = L(v) \setminus \{3\}$.
- 4 **if** Proposition 9 returns yes for the instance (G, L_1, k) **then**
- 5 **return** yes
- 6 Define $L_2 : V(G) \rightarrow 2^{[3]}$ as follows. For every $v \in F$, set $L_2(v) = L(v) \setminus \{2\}$. For every $v \notin F$, set $L_2(v) = L(v) \setminus \{3\}$.
- 7 **if** Proposition 9 returns yes for the instance (G, L_2, k) **then**
- 8 **return** yes
- 9 **return** no

clique-cutset partition (V_1, V_2, V_3) of $G - S$. Note that $G[V_3]$ is a clique, V_1, V_2 is a stable-pair, and V_3 is disjoint from $V_1 \cup V_2$. Then, by Proposition 20, there exists $F \in \mathcal{F}$ such that F contains V_3 and F is disjoint from at least one of V_1 and V_2 . Consider the set of lists L_1 obtained from L by removing 1 from $L(v)$ for every $v \in F$ and 3 from $L(v)$ for every $v \notin F$. Observe that if $F \cap V_1 = \emptyset$, then S is a solution for the DELETION TO LIST M -PARTITION instance (G, L_1, k) as well. Similarly, let L_2 be the set of lists obtained from L by removing 2 from $L(v)$ for every $v \in F$ and 3 from $L(v)$ for every $v \notin F$. Then if $F \cap V_2 = \emptyset$, then S is a solution for the DELETION TO LIST M -PARTITION instance (G, L_2, k) as well. But we know that either $F \cap V_1 = \emptyset$ or $F \cap V_2 = \emptyset$. Therefore, S is a solution to either (G, L_1, k) or (G, L_2, k) . These observations lead us to our algorithm (see Algorithm 3.1).

The correctness of Algorithm 3.1 is apparent from our previous discussions. As for the running time, Step 1 takes $n^{\mathcal{O}(\log n)}$ time to construct \mathcal{F} , and we have $|\mathcal{F}| \leq n^{\log n}$. For each $F \in \mathcal{F}$, Steps 2–8 take $2.3146^k n^{\mathcal{O}(1)}$ time. Therefore, the algorithm runs in time $2.3146^k n^{\mathcal{O}(\log n)}$. We have thus proved Lemma 18.

Algorithm for Deletion to List M -Partition, where M is the bipartite-star matrix or the three-stars matrix. We are now going to design an FPT algorithm that works for two different partitions. The first of these is an M -partition when M is defined by $m_{1,1} = m_{2,2} = 0$, $m_{3,3} = *$, $m_{1,2} = *$, $m_{1,3} = m_{2,3} = 0$. We call M the bipartite-star matrix and an M -partition a bipartite-star partition. The second partition is an M -partition for M defined by $m_{1,1} = m_{2,2} = m_{3,3} = *$, $m_{1,2} = m_{1,3} = m_{2,3} = 0$. We call M the three-stars matrix and an M -partition a three-stars partition. We shall prove the following lemma.

► **Lemma 21.** DELETION TO LIST M -PARTITION, where M is either the bipartite-star matrix or the three-stars matrix is fixed-parameter tractable.

We prove Lemma 21 by showing that Algorithm 3.2 solves DELETION TO LIST M -PARTITION in $16^k n^{\mathcal{O}(1)}$ time, when M is either the bipartite-star matrix or the three-stars matrix. Let (G, L, k) be an instance of DELETION TO LIST M -PARTITION. The idea behind our algorithm is as follows. Assume that (G, L, k) is a yes-instance. Let $S \subseteq V(G)$ be an optimal solution for the problem, and $\mathcal{V} = \{V_1, V_2, V_3\}$ be a list M -partition of $G - S$. Then, S contains an (X, Y) -separator where $X = \{u \in V(G) \mid L(u) = \{3\}\}$ and $Y = \{u \in V(G) \mid L(u) \subseteq \{1, 2\}\}$. And for a vertex u with $3 \in L(u)$, note that u can be placed in V_3 (because $m_{3,3} = *$) if u

■ **Algorithm 3.2** Algorithm for the proof of Lemma 21.

Input: (G, L, k) .
Output: yes or no.

- 1 Define $X = \{u \in V(G) \mid L(u) = \{3\}\}$ and $Y = \{u \in V(G) \mid L(u) \subseteq \{1, 2\}\}$.
- 2 Let \mathcal{F} be the family of all important (X, Y) -separators of size at most k .
- 3 **for** each $\hat{S} \in \mathcal{F}$ **do**
- 4 Let \hat{Z} be the reachability set of $Y \setminus \hat{S}$ in $G - \hat{S}$. (Note that for
 $v \in V(G) \setminus (\hat{S} \cup \hat{Z})$, we have $3 \in L(v)$, and for $v \in \hat{Z}$, $L(v) \cap \{1, 2\} \neq \emptyset$.)
- 5 Define a new list function L' for the vertices in \hat{Z} . For every $v \in \hat{Z}$, let
 $L'(v) = L(v) \setminus \{3\}$. (Then $|L'(v)| \leq 2$ for every $v \in \hat{Z}$.)
- 6 Set $k' = k - |\hat{S}|$.
- 7 **if** Proposition 9 returns yes for the instance $(G[\hat{Z}], L', k')$ **then**
- 8 **return** yes
- 9 **return** no

is not reachable from Y in $G - S$. Hence we try to place as many vertices as possible in V_3 by choosing an (X, Y) -separator that is “farthest from X ,” (and by augmenting such a separator to ensure that the other constraints dictated by M are also satisfied). For this, Algorithm 3.2 uses the concept of an important separator, introduced by Marx in [27]. For a graph G and $A \subseteq V(G)$, $R_G(A) = \{v \in V(G) \mid \text{there is a path from } x \text{ to } v \text{ in } G \text{ for some } x \in A\}$. And the set $R_G(A)$ is called the *reachability set of A in G* . For $A, B \subseteq V(G)$, a minimal (A, B) -separator $S \subseteq V(G)$ is said to be an *important (A, B) -separator* if there exists no (A, B) -separator S' such that $|S'| \leq |S|$ and $R_{G-S}(A) \subset R_{G-S'}(A)$.

► **Proposition 22** ([27, 6]). *Given a graph G , sets $A, B \subseteq V(G)$ and an integer k , G contains at most 4^k important (A, B) -separators of size at most k . Moreover, all these important separators can be enumerated in time $\mathcal{O}(4^k(|V(G)| + |E(G)|))$.*

► **Lemma 23.** *Algorithm 3.2 is correct.*

Proof. In light of Remark 13, we assume that $X, Y \neq \emptyset$, where X and Y are as defined in Step 1 of Algorithm 3.2.

Notice first that if Algorithm 3.2 returns yes, then (G, L, k) is indeed a yes-instance. Assume that the algorithm returns yes. Then there exists $\hat{S} \in \mathcal{F}$ such that \hat{S} is an important (X, Y) -separator, and $(G[\hat{Z}], L', k')$ is a yes-instance, where $\hat{Z} = R_{(G-\hat{S})}(Y \setminus \hat{S})$, (and X, Y, \mathcal{F}, L' and k' are as defined in the algorithm). Let $U \subseteq \hat{Z}$ be an optimal solution for the instance $(G[\hat{Z}], L', k')$. Then, $|U| \leq k' = k - |\hat{S}|$. Let (V_1, V_2, V_3) be a partition of $Z \setminus U$ that respects L' . Since $3 \notin L'(v)$ for every $v \in Z$, we have $V_3 = \emptyset$. It is not difficult to see that $(V_1, V_2, V(G) \setminus (\hat{Z} \cup \hat{S} \cup U))$ is a list partition of $V(G) \setminus (\hat{S} \cup U)$ that respects L . That is, $\hat{S} \cup U$ is a solution for the instance (G, L, k) and $|\hat{S} \cup U| \leq k$.

Now, we prove that if (G, L, k) is a yes-instance, then Algorithm 3.2 returns yes. Assume that (G, L, k) is a yes-instance, and let S be an optimal solution to the list partition instance (G, L, k) , and $S' \subseteq S$ be a minimal (X, Y) -separator. Let \hat{S} be an important (X, Y) -separator that dominates S' , i.e., $|\hat{S}| \leq |S'|$ and $R_{G-S'}(X) \subset R_{G-\hat{S}}(X)$. We will show that $\tilde{S} = (S \setminus S') \cup \hat{S}$ is also an optimal solution.

We first show that \hat{S} is an $(S' \setminus \hat{S}, Y)$ -separator. Suppose not, then there is an s - y path P in $G - \hat{S}$ for some $s \in S' \setminus \hat{S}$ and $y \in Y \setminus \hat{S}$. As S' is a minimal (X, Y) -separator, there is an X - Y path, say P' , that intersects S' only in s . Consider the X - s subpath of

P' and call it P'' . None of the vertices of P'' can belong to \hat{S} , as for each $v \in V(P'')$, $v \in R_{G-S'}(X) \subseteq R_{G-\hat{S}}(X)$. Thus, P'' is a path in $G - \hat{S}$. Then P'' followed by the path P is an X - Y path in $G - \hat{S}$, which contradicts the fact that \hat{S} is an (X, Y) -separator. Thus, \hat{S} is an $(S' \setminus \hat{S}, Y)$ -separator.

Now, let Z' be the reachability set of $Y \setminus S'$ in $G - S'$, and \hat{Z} the reachability set of $Y \setminus \hat{S}$ in $G - \hat{S}$. (Notice that for every vertex $v \notin \hat{Z}$, we have $3 \in L(v)$, and therefore, v can be placed in V_3 without violating any of the constraints on $m_{3,3}$ or $m_{1,3}$ or $m_{2,3}$. And every vertex $v \in \hat{Z}$ has either 1 or 2 on its list.) We claim that $\hat{Z} \subseteq Z'$. Consider $z \in \hat{Z}$. Let Q be a w - z path in $G - \hat{S}$ for some $w \in Y$. Suppose $z \notin Z'$. Then Q must pass through S' , i.e., $V(Q) \cap S' \neq \emptyset$. Let $u \in V(Q) \cap S' \neq \emptyset$. But then the u - w subpath of Q is an $(S' \setminus \hat{S})$ - Y path in $G - \hat{S}$, which contradicts the fact that \hat{S} is an $(S' \setminus \hat{S}, Y)$ -separator. Thus, $\hat{Z} \subseteq Z'$, and $G[\hat{Z}]$ is an induced subgraph of $G[Z']$. Observe that $(S \setminus S')$ is a solution of size at most $k - |S'|$ for the list partition instance on $G[Z']$. Hence, $(S \setminus S')$ is a solution for the list partition instance on $G[\hat{Z}]$ as well. ◀

► **Lemma 24** (*). *Algorithm 3.2 runs in time $16^k n^{\mathcal{O}(1)}$.*

Lemmas 23 and 24 together prove Lemma 21. Next we give a proof sketch of Theorem 1.

Proof sketch of Theorem 1. The proofs of item 1, item 2 and item 3 of the theorem statement follow from Lemma 16, Observation 17 and Lemma 18, respectively. Therefore, we next focus on the proof of item 4 of the theorem statement. We only consider matrices that are not covered by any of the previous three cases. Let us first classify matrices M depending on the number of off-diagonal entries that are 0s. In fact, we will only make distinctions based on the off-diagonal entries in the upper triangular submatrix M_U of M , since M is a symmetric matrix. Below, we deal with one of the classes, which is illustrative of the techniques used to resolve these problems.

Exactly two off-diagonal entries of M_U are 0s. Assume that $m_{1,3} = m_{2,3} = 0$ and $m_{1,2} \in \{1, *\}$. All other cases of two off-diagonal entries of M_U being 0 can be symmetrically argued. If any one of $m_{1,1}, m_{2,2}$ or $m_{3,3}$ is 1, then M has a row that contains both a 0 and a 1, and then by Proposition 10, the problem is fixed-parameter tractable. So assume that $m_{1,1}, m_{2,2}, m_{3,3} \in \{0, *\}$. We consider each possibility of $m_{1,2}$ for our analysis. First, suppose $m_{1,2} = *$. If $m_{1,1} = *$, then index 1 dominates 2. If $m_{2,2} = *$, then index 2 dominates 1. In either case, the problem is fixed-parameter tractable, by Propositions 8 and 9. So assume that $m_{1,1} = m_{2,2} = 0$. If $m_{3,3} = 0$, then index 1 dominates 3, and by Propositions 8 and 9 the problem is again fixed-parameter tractable. If $m_{3,3} = *$, then an M -partition is a bipartite-star partition, and by Lemma 21, the problem is fixed-parameter tractable. The other possibility for $m_{1,2}$ is that $m_{1,2} = 1$. Then M has a row containing both a 0 and a 1, and by Proposition 10, the problem is fixed-parameter tractable. ◀

4 Conclusion

We almost complete the parameterized classification of DELETION TO LIST M -PARTITION when the matrix M is of order ≤ 3 , or when it is of order 4 and has its diagonal entries from $\{0, 1\}$. We do not know whether the DELETION TO CLIQUE CUTSET problem is FPT— we obtain an algorithm with running time $2.3146^k n^{\mathcal{O}(\log n)}$, where k is the solution size. Also, the NP-hardness of the DELETION TO LIST M -PARTITION problem when $m_{1,1} = m_{2,2} = m_{3,3} = *$, $m_{1,2} = 0, m_{1,3} = 1, m_{2,3} = *$ is open, although we give an FPT algorithm, parameterized by

the solution size. It would be interesting to complete the classification of these problems, as well as the parameterized dichotomy of DELETION TO LIST M -PARTITION for all matrices of order 4. We are also interested in optimising the running time of our algorithms, and studying the kernelization complexity of these problems in the future.

References

- 1 Noga Alon and Michael Tarsi. Colorings and orientations of graphs. *Combinatorica*, 12(2):125–134, 1992.
- 2 Bengt Aspvall, Michael F. Plass, and Robert E. Tarjan. A Linear-Time Algorithm for Testing the Truth of Certain Quantified Boolean Formulas. *Information Processing Letters*, 8(3):121–123, 1979.
- 3 Armen S. Asratian, Tristan M. J. Denley, and Roland Häggkvist. *Bipartite graphs and their applications*, volume 131. Cambridge University Press, 1998.
- 4 Andreas Brandstädt, Feodor F. Dragan, Van Bang Le, and Thomas Szymczak. On stable cutsets in graphs. *Discrete Applied Mathematics*, 105(1-3):39–50, 2000. doi:10.1016/S0166-218X(00)00197-9.
- 5 Kathie Cameron, Elaine M. Eschen, Chinh T. Hoàng, and R. Sritharan. The Complexity of the List Partition Problem for Graphs. *SIAM J. Discrete Math.*, 21(4):900–929, 2007.
- 6 Jianer Chen, Yang Liu, Songjian Lu, Barry O’Sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *Journal of the ACM*, 55(5):21:1–21:19, 2008.
- 7 Rajesh Chitnis, László Egri, and Dániel Marx. List H-Coloring a Graph by Removing Few Vertices. *Algorithmica*, 78(1):110–146, 2017.
- 8 Marek Cygan, Fedor V. Fomin, Alexander Golovnev, Alexander S. Kulikov, Ivan Mihajlin, Jakub Pachocki, and Arkadiusz Socała. Tight Bounds for Graph Homomorphism and Subgraph Isomorphism. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 1643–1649, 2016.
- 9 Elias Dahlhaus, David S. Johnson, Christos H. Papadimitriou, Paul D. Seymour, and Mihalis Yannakakis. The Complexity of Multiway Cuts (Extended Abstract). In *ACM Symposium on Theory of Computing (STOC)*, pages 241–251, 1992.
- 10 Josep Díaz, Maria Serna, and Dimitrios M. Thilikos. (H,C,K)-coloring: Fast, easy, and hard cases. In *Mathematical Foundations of Computer Science (MFCS)*, pages 304–315, 2001.
- 11 Josep Díaz, Maria Serna, and Dimitrios M Thilikos. Recent results on parameterized H-colorings. *Discrete Mathematics and Theoretical Computer Science*, 63:65–86, 2004.
- 12 László Egri, Andrei Krokhin, Benoit Larose, and Pascal Tesson. The complexity of the list homomorphism problem for graphs. *Theory of Computing Systems*, 51(2):143–178, 2012.
- 13 Paul Erdős, Arthur L. Rubin, and Herbert Taylor. Choosability in graphs. In *Proc. West Coast Conf. on Combinatorics, Graph Theory and Computing, Congressus Numerantium*, volume 26, pages 125–157, 1979.
- 14 Tomás Feder and Pavol Hell. List Homomorphisms to Reflexive Graphs. *Journal of Combinatorial Theory, Series B*, 72(2):236–250, 1998.
- 15 Tomás Feder and Pavol Hell. Full Constraint Satisfaction Problems. *SIAM J. Comput.*, 36(1):230–246, 2006. doi:10.1137/S0097539703427197.
- 16 Tomás Feder, Pavol Hell, and Jing Huang. List Homomorphisms and Circular Arc Graphs. *Combinatorica*, 19(4):487–505, October 1999.
- 17 Tomás Feder, Pavol Hell, Sulamita Klein, and Rajeev Motwani. List Partitions. *Journal of Discrete Mathematics*, 16(3):449–478, 2003.
- 18 Lester R. Ford and Delbert R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8(3):399–404, 1956.
- 19 Michael R. Garey and David S. Johnson. Computers and intractability: A guide to the theory of NP-completeness. *Computers and Intractability*, page 340, 1979.

- 20 Michael R. Garey, David S. Johnson, and Larry J. Stockmeyer. Some Simplified NP-Complete Graph Problems. *Theor. Comput. Sci.*, 1(3):237–267, 1976. doi:10.1016/0304-3975(76)90059-1.
- 21 Andreas Göbel, Leslie Ann Goldberg, Colin McQuillan, David Richerby, and Tomoyuki Yamakami. Counting List Matrix Partitions of Graphs. *SIAM J. Comput.*, 44(4):1089–1118, 2015. doi:10.1137/140963029.
- 22 Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs*. Elsevier, 2004.
- 23 Melven R. Krom. The decision problem for a class of first-order formulas in which all disjunctions are binary. *Mathematical Logic Quarterly*, 13(1-2):15–20, 1967.
- 24 Marek Kubale. Some results concerning the complexity of restricted colorings of graphs. *Discrete Applied Mathematics*, 36(1):35–46, 1992.
- 25 S. Føldes and P. L. Hammer. Split graphs. *South-Eastern Conference on Combinatorics, Graph Theory and Computing (SEICCGTC)*, pages 311–315, 1977.
- 26 Daniel Lokshantov, N. S. Narayanaswamy, Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. Faster Parameterized Algorithms Using Linear Programming. *Transactions on Algorithms*, 11(2):15:1–15:31, 2014.
- 27 Dániel Marx. Parameterized graph separation problems. *Theoretical Computer Science*, 351(3):394–406, 2006.
- 28 Vadim G. Vizing. Vertex colorings with given colors. *Diskret. Analiz*, 29:3–10, 1976.
- 29 Margit Voigt. List colourings of planar graphs. *Discrete Mathematics*, 120(1-3):215–219, 1993.
- 30 Mihalis Yannakakis. Node-and Edge-deletion NP-complete Problems. In *ACM Symposium on Theory of Computing (STOC)*, pages 253–264, 1978.

The Generalized Microscopic Image Reconstruction Problem

Amotz Bar-Noy

City University of New York (CUNY), USA
amotz@sci.brooklyn.cuny.edu

Toni Böhnlein

Bar Ilan University, Ramat-Gan, Israel
toni.bohnlein@biu.ac.il

Zvi Lotker

Ben Gurion University of the Negev, Beer Sheva, Israel
Bar Ilan University, Ramat-Gan, Israel
zvi.lotker@gmail.com

David Peleg

Weizmann Institute of Science, Rehovot, Israel
david.peleg@weizmann.ac.il

Dror Rawitz

Bar Ilan University, Ramat-Gan, Israel
dror.rawitz@biu.ac.il

Abstract

This paper presents and studies a generalization of the *microscopic image reconstruction* problem (*MIR*) introduced by Frosini and Nivat [7, 12]. Consider a specimen for inspection, represented as a collection of points typically organized on a grid in the plane. Assume each point x has an associated physical value ℓ_x , which we would like to determine. However, it might be that obtaining these values precisely (by a *surgical probe*) is difficult, risky, or impossible. The alternative is to employ *aggregate* measuring techniques (such as EM, CT, US or MRI), whereby each measurement is taken over a larger window, and the exact values at each point are subsequently extracted by computational methods.

In this paper we extend the MIR framework in a number of ways. First, we consider a generalized setting where the inspected object is represented by an arbitrary graph G , and the vector $\ell \in \mathbb{R}^n$ assigns a value ℓ_v to each node v . A probe centered at a vertex v will capture a window encompassing its entire neighborhood $N[v]$, i.e., the outcome of a probe centered at v is $\mathcal{P}_v = \sum_{w \in N[v]} \ell_w$. We give a criterion for the graphs for which the extended MIR problem can be solved by extracting the vector ℓ from the collection of probes, $\bar{\mathcal{P}} = \{\mathcal{P}_v \mid v \in V\}$.

We then consider cases where such reconstruction is impossible (namely, graphs G for which the probe vector \mathcal{P} is inconclusive, in the sense that there may be more than one vector ℓ yielding \mathcal{P}). Let us assume that surgical probes (whose outcome at vertex v is the exact value of ℓ_v) are technically available to us (yet are expensive or risky, and must be used sparingly). We show that in such cases, it may still be possible to achieve reconstruction based on a *combination* of a collection of standard probes together with a suitable set of surgical probes. We aim at identifying the minimum number of surgical probes necessary for a unique reconstruction, depending on the graph topology. This is referred to as the **MINIMUM SURGICAL PROBING** problem (MSP).

Besides providing a solution for the above problems for arbitrary graphs, we also explore the range of possible behaviors of the **MINIMUM SURGICAL PROBING** problem by determining the number of surgical probes necessary in certain specific graph families, such as perfect k -ary trees, paths, cycles, grids, tori and tubes.

2012 ACM Subject Classification Mathematics of computing

Keywords and phrases Discrete mathematics, Combinatorics, Reconstruction algorithm, Image reconstruction, Graph spectra, Grid graphs



© Amotz Bar-Noy, Toni Böhnlein, Zvi Lotker, David Peleg, and Dror Rawitz;
licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 42; pp. 42:1–42:15

Leibniz International Proceedings in Informatics



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

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.42

Funding This work was supported by US-Israel BSF grant 2018043.

Amotz Bar-Noy: ARL Cooperative Grant, ARL Network Science CTA, W911NF-09-2-0053

Dror Rawitz: ISF grant no. 497/14

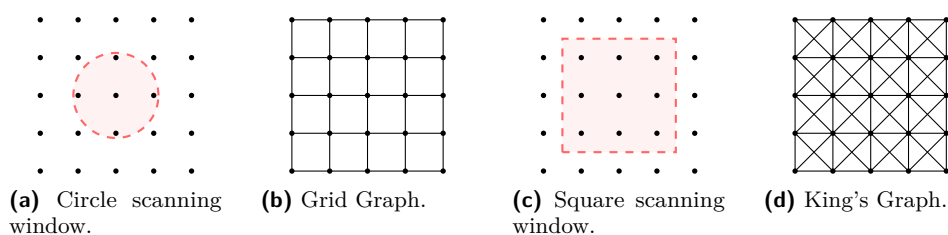
1 Introduction

Background and motivation. Imaging technologies are used increasingly widely in a variety of medical, engineering, and scientific application domains. Imagine a specimen for inspection, represented as a collection of points organized in (2- or 3-dimensional) space. Assume each point x has an associated physical value ℓ_x (e.g., atom density, brightness, etc). Our goal is to determine these values at all points in the specimen. However, it is often the case that obtaining these values through a direct and precise inspection (hereafter referred to as a *surgical probe*) is complicated, prohibitively expensive, potentially risky, or even impossible. A commonly used alternative is to employ *aggregate* measuring techniques (such as EM, CT, US or MRI), whereby measurements are taken over a larger area (rather than a single point), and the (exact or approximate) values at each point are subsequently extracted by computational methods. For example, a microscope with a scanning window can be used for inspecting the specimen by systematically going over it and *probing* (i.e., taking a measurement in) each window. The measurement taken from each window centered at point x , hereafter denoted by \mathcal{P}_x , consists of the sum of the observed values at all the points in the observed window. (This is sometimes referred to as the *luminosity* of the window.) The goal is then to use the measurements obtained by a sufficiently diverse collection of probes in order to deduce the original values ℓ_x at each point x in the specimen.

This general problem has been extensively studied as the *discrete tomography* reconstruction problem (*DTR*). For a survey, see Herman and Kuba [10]. The *microscopic image reconstruction* problem (*MIR*) was then introduced by Frosini and Nivat in [7, 12] as a natural extension of the DTR problem. In both problems, the specimen is represented by a 2-dimensional grid (see Figure 1b) whose points, $x = (i, j)$, for $i \in \{1, \dots, n_1\}$ and $j \in \{1, \dots, n_2\}$, are assigned nonnegative integer¹ values $\ell_{i,j}$. In the DTR problem, the window of a probe is typically an entire row or column (i.e., the probe can be thought of as performed by a ray piercing the specimen from one side to the other). In the MIR problem, it is assumed that the microscope's scanning window is a segment of the plane (e.g., a circle or a rectangle). For example, assume for the sake of illustration that the window corresponds to a circle of radius 1 (see Figure 1a). Then the input can be thought of as an $n_1 \times n_2$ integer matrix, and the output is an $n_1 \times n_2$ integer matrix. A similar setting can be described with a square scanning window as shown in Figure 1c. A window consisting of a node and its eight neighbors in the grid is depicted by the king's graph, illustrated in Figure 1d.

In this paper we extend the MIR framework of [7] in a number of ways. First, we consider a generalized setting where the inspected object is represented by an arbitrary simple undirected connected graph $G = (V, E)$ with vertex set $V = \{1, \dots, n\}$. Given a graph G , the vector $\ell \in \mathbb{R}^n$ is an assignment of a value ℓ_v to each node v . Given a graph G and a vector $\ell \in \mathbb{R}^n$, define $\ell(U) \triangleq \sum_{v \in U} \ell_v$, for every $U \subseteq V$. Here, a probe centered at a vertex v captures a window encompassing its entire neighborhood, $N[v] \triangleq \{w \mid (v, w) \in E\} \cup \{v\}$, i.e., the outcome of a probe centered at v is $\mathcal{P}_v \triangleq \ell(N[v]) = \sum_{w \in N[v]} \ell_w$. For example, in

¹ In fact, [7] assume only Boolean values, 0 or 1.



■ **Figure 1** Scanning windows for the grid.

the case of a grid specimen, $N[v]$ may contain all vertices at distance at most d (according to any norm L_p) from v . Our first question is to determine the class of graphs for which the extended MIR problem can be solved, namely, for which it is possible to extract the vector ℓ from the collection of probes at all vertices, $\bar{\mathcal{P}} = \{\mathcal{P}_v \mid v \in V\}$.

Note, however, that in some cases, this type of reconstruction is not possible. Given a graph G and a probe vector \mathcal{P} , it may be possible that the outcome of the measurements is inconclusive, in the sense that there may be several (or even infinitely many) vectors ℓ that would yield the same probe vector \mathcal{P} . For example, consider the case where G consists of two nodes and an edge between them. In this case, the same probe vector $\mathcal{P} = (p_1, p_2)$ is obtained for any vector (ℓ_1, ℓ_2) such that $\ell_1 + \ell_2 = p_1 = p_2$.

This leads to our next extension of the problem. Let us assume that surgical probes (whose outcome at vertex v is the exact value of ℓ_v) are technically available to us, yet are so expensive or risky that we must use them sparingly. In cases where a unique reconstruction based on standard (aggregate) probes alone is not possible, it may still be possible to achieve a reliable reconstruction based on a combination of a comprehensive collection of standard probes together with a (hopefully small) set of surgical probes. Hence, our second goal is to identify the minimum number of surgical probes necessary for a unique reconstruction, depending on the graph topology. Formally, we consider the following MINIMUM SURGICAL PROBING problem (MSP). Given a graph G and a vector \mathcal{P} , the goal is to find the actual vector ℓ that generated \mathcal{P} , using as few surgical probes as possible.

Our results. In Section 2 we present an efficient algorithm for solving the MINIMUM SURGICAL PROBING problem. We show that one can compute the minimum number of surgical probes necessary for any graph and determine a subset of the vertices which need to be probed. The general problem can be formulated as a system of linear equations. The adjacency matrix of our graph (whose main diagonal is set to 1) determines the coefficient matrix and the probe vector \mathcal{P} is the right hand side. We use techniques from linear algebra to solve the problem.

While these results allow us to determine the number of surgical probes necessary for every graph, it is interesting to explore and chart the range of possible behaviors of the problem, by identifying the number of surgical probes necessary for some specific graph families. Towards this goal, we consider (in Section 3) the behavior of the problem on trees. We first show that ℓ can be uncovered on any n -vertex tree using $\lfloor \frac{n}{2} \rfloor - 1$ many surgical probes, and that this number is tight if n is odd. In contrast we show that on the class of perfect k -ary trees, no surgical probes are required to uncover ℓ .

We continue pursuing this line of investigation by considering (in Section 4) *Cartesian products* of paths and cycles, resulting in *grids*, *tubes* and *tori*. Furthermore, Section 5 deals with the *Strong product* of two path graphs, which is known as the king's graph (see Figure 1d). Grid graphs are interesting as they have the topology which was studied in previous papers on discrete tomography. A probe \mathcal{P} has a circular scanning window. Similarly, in the king's graph a probe \mathcal{P} has a square scanning window.

42:4 The Generalized Microscopic Image Reconstruction Problem

As we will see, the number of required surgical probes is related to the rank of our graph's adjacency matrix which in turn is related to its eigenvalues. The eigenvalues of adjacency matrices are studied in spectral graph theory. Simple expressions to determine the eigenvalues for adjacency matrices of path and cycle graphs are known. This and the fact that Cartesian and Strong products preserve the eigenvalues of their factor's adjacency matrices allow us to derive expressions for the eigenvalues of the grids, tubes, tori and king's graph adjacency matrices. We use these expressions to determine the number of surgical probes in a more efficient way than our general result allows for.

Table 1 lists the number of surgical probes that are sufficient to discover ℓ for grids, tubes, tori and king's graphs. To express our results, we introduce the following indicator variables:

$$\mathcal{I}_b^a(n) \triangleq \begin{cases} 1 & \text{if } n \bmod a \equiv b \\ 0 & \text{otherwise.} \end{cases}$$

Surprisingly, only a constant number of surgical probes are needed for any grid, tube, or torus. On the other hand, the king's graph may require as many as $n_1 + n_2 - 1$ probes. In addition, in all the above graphs the number of probes may be zero, depending on the graph dimensions. For example, when both n_1 and n_2 are multiples of 30, a grid of size $n_1 \times n_2$ does not require surgical probes. Similarly, when n_1 and n_2 are multiples of 3, no surgical probes are required for the king's graph. Hence, when given control on the dimensions (say, in a design phase), one may fix the dimensions such that no surgical probes are required.

Note that, these results give us only the number of surgical probes that are sufficient to uncover ℓ . In general, we need to resort to our result from Section 2 to find a set of vertices that needs to be probed. For paths and cycles, however, it is possible to find these vertices directly and uncover ℓ in linear time.

■ **Table 1** The table shows the number of surgical probes. In the case of a tube n_1 is the length of the path while $n_2 \geq 3$ is the length of the cycle.

Graph	# surgical probes	at most
Grid	$\mathcal{I}_2^3(n_1)\mathcal{I}_1^2(n_2) + \mathcal{I}_1^2(n_1)\mathcal{I}_2^3(n_2) + 2\mathcal{I}_4^5(n_1)\mathcal{I}_4^5(n_2)$	4
Path ($n_2 = 1$)	$\mathcal{I}_2^3(n_1)$	1
Tube	$2\mathcal{I}_1^2(n_1)\mathcal{I}_0^3(n_2) + \mathcal{I}_2^3(n_1)\mathcal{I}_0^2(n_2) + 2\mathcal{I}_2^3(n_1)\mathcal{I}_0^4(n_2) + 4\mathcal{I}_4^5(n_1)\mathcal{I}_0^5(n_2)$	9
Cycle ($n_1 = 1$)	$2\mathcal{I}_0^3(n_2)$	2
Torus	$4\mathcal{I}_0^3(n_1)\mathcal{I}_0^4(n_2) + 4\mathcal{I}_0^4(n_1)\mathcal{I}_0^3(n_2) + 2\mathcal{I}_0^2(n_1)\mathcal{I}_0^6(n_2) + 2\mathcal{I}_0^6(n_1)\mathcal{I}_0^2(n_2) + 8\mathcal{I}_0^5(n_1)\mathcal{I}_0^5(n_2)$	20
King's graph	$\mathcal{I}_2^3(n_1)n_2 + \mathcal{I}_2^3(n_2)n_1 - \mathcal{I}_2^3(n_1)\mathcal{I}_2^3(n_2)$	$n_1 + n_2 - 1$

Related Work. Most important for our work is the extension of the DTR problem by Frosini and Nivat [7, 12]. They introduced the problem of reconstructing a binary matrix from a rectangular scan instead of the row and column sums. A polynomial time algorithm is presented that solves the reconstruction problem for a class of matrices. Battaglini, Frosini and Rinaldi [4] extended this by studying scans of different shapes like diamond shapes. Relations to tiling are discovered.

A concept similar to surgical probing was considered by Frosini, Nivat and Rinaldi [8]. They analyze a variation where a grid is scanned by the means of two rectangular windows. To solve the reconstruction problem, they assume that a small number of the values associated with the grid points are known in advance.

A combination of DTR and MIR is studied by Alpers and Gritzmann [1]. Their goal is to reconstruct a matrix from column and row sums with additional windows constraints. In [3] they study applications to image reconstruction.

Gritzmann et al. [9] show that determining the minimum number of prescribed values making the DTR problem unique is in general a NP-hard problem. Prescribed values are an analogue to surgical probes. The hardness results from the integer values in the DTR problem. Alpers and Gritzmann [2] discuss uniqueness problems for a dynamic variation of DTR. Here, the application is to track particles over time.

2 Algorithm for Solving Minimum Surgical Probing

In this section, we show how to solve the MINIMUM SURGICAL PROBING problem. We start with some preliminaries. Denote the $n \times n$ identity matrix by I_n . Given a matrix A , let $\text{rank}(A)$ denote its rank, and let $\Lambda(A)$ denote its set of eigenvalues. For an eigenvalue $\lambda \in \Lambda(A)$, denote by $\phi(\lambda, A)$ the multiplicity of λ in $\Lambda(A)$.

Given a graph G , let $A_G = \{a_{ij}\}$ be G 's $n \times n$ adjacency matrix, i.e.,

$$a_{ij} = \begin{cases} 1, & (i, j) \in E, \\ 0, & \text{otherwise.} \end{cases}$$

Define $\bar{A}_G \triangleq A_G + I_{|V|}$ as the adjacency matrix whose main diagonal is set to 1.

A probe vector \mathcal{P} is induced by $\ell \in \mathbb{R}^n$ if the following is satisfied:

$$\bar{A}_G \cdot \ell = \mathcal{P}. \tag{1}$$

► **Lemma 1.** *The multiplicity of the eigenvalue -1 in the matrix A_G is*

$$\phi(-1, A_G) = \phi(0, \bar{A}_G) = |V| - \text{rank}(\bar{A}_G).$$

Proof. If \bar{A}_G has an eigenvalue 0, then it is singular. Indeed, the multiplicity of the eigenvalue 0 is the dimension on the kernel (null-space) $\ker(\bar{A}_G)$ (cf. [5]). The eigenspace of an eigenvalue λ consists of the solutions to $(\bar{A}_G - \lambda I_{|V|})x = 0$. Therefore, the eigenspace of eigenvalue 0 is the kernel of \bar{A}_G . If 0 is an eigenvalue of \bar{A}_G , then -1 is an eigenvalue of A_G , and the multiplicity is the same. ◀

We now present our general result.

► **Theorem 2.** *Consider a graph G and a probe vector \mathcal{P} .*

1. *If the adjacency matrix \bar{A}_G has full rank, i.e., $\text{rank}(\bar{A}_G) = |V|$, then ℓ can be uncovered in polynomial time without using any surgical probes.*
2. *Otherwise, the minimum number of surgical probes needed to uncover ℓ is $s = \phi(-1, A_G)$. Moreover, a set of s nodes whose surgical probes uncover ℓ can be computed in polynomial time.*

Proof. As mentioned above, the label vector $\ell \in \mathbb{R}^{|V|}$ satisfies the system of linear equations given in (1). Therefore, system (1) has at least one solution. If the adjacency matrix \bar{A}_G has full rank, i.e., $\text{rank}(\bar{A}_G) = |V|$, then system (1) has a unique solution. A standard method, like the Gauss-Jordan elimination, can be used to uncover ℓ without any surgical probe in polynomial time.

42:6 The Generalized Microscopic Image Reconstruction Problem

If $\text{rank}(\bar{A}_G) < |V|$, then system (1) has (infinitely) many solutions. The column space $C(\bar{A}_G)$ of \bar{A}_G contains all vectors $\bar{A}_G \cdot x$. It is spanned by a maximal subset of independent columns of \bar{A}_G , and its dimension is $\text{rank}(\bar{A}_G)$. The kernel $\ker(\bar{A}_G)$ of \bar{A}_G consists of all solutions to the homogeneous system $\bar{A}_G \cdot x = 0$; its dimension is $s \triangleq |V| - \text{rank}(\bar{A}_G)$. To describe all solutions of system (1), let x be a vector such that $\bar{A}_G \cdot x = \mathcal{P}$. Then,

$$\bar{A}_G \cdot (x + x_0) = \bar{A}_G \cdot x + \bar{A}_G \cdot x_0 = \mathcal{P} + 0 = \mathcal{P}$$

for any $x_0 \in \ker(\bar{A}_G)$.

To uncover ℓ , we need to remove this ambiguity by surgical probes. One approach is to select a maximal subset of independent columns U of \bar{A}_G . The columns of U are a basis of $C(\bar{A}_G)$, and $|U| = \text{rank}(\bar{A}_G)$. Let $U' = \bar{A}_G \setminus U$ be the remaining columns. Thus, $|U'| = s$. Probing the vertices that correspond to the columns of U' returns $\bar{\ell} \in \mathbb{R}^s$. Now, we can solve the system

$$U \cdot x = \mathcal{P} - U' \cdot \bar{\ell} \tag{2}$$

to uncover the unknown entries of ℓ . If ℓ is a solution of system (1), then system (2) has at least one solution. Since U is a set of linearly independent columns, this solution is unique. If we select more than $\text{rank}(\bar{A}_G)$ columns to form U , system (2) does not have a unique solution.

A structured way to select a maximal subset of independent columns of \bar{A}_G is the *reduced row echelon form*. The reduced row echelon form can be computed in polynomial time using the Gauss-Jordan elimination method (cf. [11]). The output is a partition of the columns into *pivot-columns* and *free-columns*. The pivot-columns form a basis of $C(\bar{A}_G)$. We probe the vertices corresponding to the free-columns. Back-substitution takes care of uncovering the missing entries of ℓ (if \bar{A}_G was augmented by \mathcal{P} before the elimination process). ◀

Note that the rank of \bar{A}_G can be as low as one, e.g., in the case of the complete graph K_n we have $\text{rank}(\bar{A}_{K_n}) = 1$. Hence, to solve MINIMUM SURGICAL PROBING for K_n a maximum number of $n - 1$ surgical probes is required. However, as we will see in the following sections, there are many graphs for which we can uncover ℓ without any surgical probes.

In classic DTR problems the labels are binary or integer vectors. If we add respective constraints on ℓ to System (1), Theorem 2 provides an upper bound on the number of required surgical probes.

3 Minimum Surgical Probing in Trees

In this section we consider the MINIMUM SURGICAL PROBING problem in trees. A first class of trees that requires no surgical probes are stars.

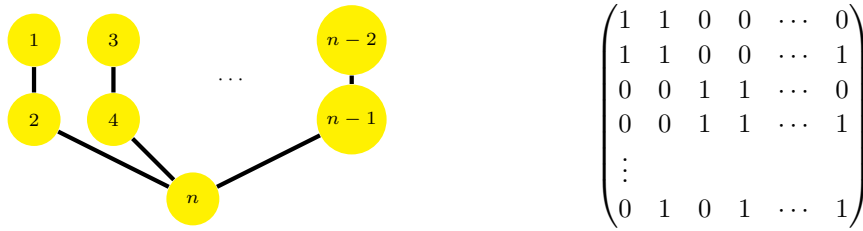
► **Lemma 3.** *A star graph requires no surgical probes to uncover ℓ .*

Proof. Let G be a star with $n \geq 3$ vertices $\{1, \dots, n\}$ where vertex n is the center. The label ℓ_n can be computed from the neighborhood probes as follows:

$$\frac{1}{n-2}(\sum_{i=1}^{n-1} \mathcal{P}_i - \mathcal{P}_n) = \frac{1}{n-2}(\sum_{i=1}^{n-1} (\ell_n + \ell_i) - (\ell_n + \sum_{i=1}^{n-1} \ell_i)) = \ell_n.$$

Given ℓ_n , the remaining labels can be computed using $\ell_i = \mathcal{P}_i - \ell_n$, for $i < n$. ◀

The following theorem shows an upper bound on the number of surgical probes that are required to uncover bipartite graphs. Trees are always connected and bipartite.



(a) Spider with $\frac{n-1}{2}$ legs.

(b) Adj. matrix of a spider.

■ **Figure 2** A spider graph and its adjacency matrix.

► **Theorem 4.** *Let $G = (L \cup R, E)$ be a connected, bipartite graph with $n \geq 3$ nodes. Then ℓ can be uncovered using $\lfloor \frac{n}{2} \rfloor - 1$ many surgical probes.*

Proof. Assume without loss of generality that $|L| \leq |R|$. Observe that there must be a node $v \in L$ such that $deg(v) > 1$ since G is connected. Now, surgical probe $L \setminus \{v\}$ and uncover the labels of all nodes in $R \setminus N(v)$. This can be done since the induced graph $T[(L \setminus \{v\}) \cup (R \setminus N(v))]$ is bipartite and we know all the labels of one of the blocks. Since the graph that is induced by the nodes with unknown labels is a star, we are done due to Lemma 3.

If n is even, we perform at most $\frac{n}{2} - 1 = \frac{n-2}{2}$ many surgical probes. If n is odd, we perform at most $\frac{n-1}{2} - 1 = \frac{n-3}{2}$ many surgical probes. ◀

We can show that the given upper bound is tight for trees where the number of vertices is odd.

► **Theorem 5.** *There exist n -vertex trees, for odd n , that require $\lfloor \frac{n}{2} \rfloor - 1$ surgical probes.*

Proof. Consider a spider G with $\frac{n-1}{2}$ legs as shown in Figure 2a. The adjacency matrix is given in Figure 2b. Elementary row operations can be used to show that $rank(\bar{A}_G) = \frac{n+3}{2}$. To see this, subtract each odd row $i < n$ from its succeeding row $i + 1$. The claim follows from Theorem 2. ◀

Another class of trees that require no surgical probes are perfect k -ary trees. Recall that a perfect k -ary tree is a tree where all internal nodes have k children and all leaves have the same depth.

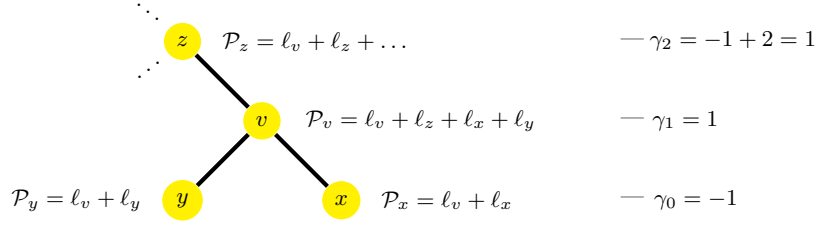
► **Theorem 6.** *Let $T = (V, E)$ be a perfect k -ary tree. Then, no surgical probes are needed to uncover ℓ .*

Proof. We prove the theorem by induction on the height h of T . For a vertex v we denote the set of its children by $ch(v)$. As a base case, we consider $h = 1$. This yields a $(k + 1)$ -vertex star graph, and due to Lemma 3, no surgical probes are needed to uncover ℓ .

Now, let $h \geq 2$ and let r be the root of T . We start by calculating the root label, ℓ_r , based on the given neighborhood probes. To do that, we express ℓ_r as a function of the probe results over the entire tree. This is done as follows. Arrange the tree vertices in levels, with the leaves on level 0 and the root r on level h . Let

$$\gamma_j = \begin{cases} -1 & j = 0, \\ 1 & j = 1, \\ -\gamma_{j-1} - k \cdot \gamma_{j-2} & j \geq 2, \end{cases} \quad \text{and} \quad b = \gamma_h + k \cdot \gamma_{h-1}. \quad (3)$$

Next, define a coefficient $a_v = \gamma_j$ for every vertex v on level j .



■ **Figure 3** Part of a perfect binary tree. Vertices x and y are leaves. The label l_x appears only in \mathcal{P}_x and \mathcal{P}_v . Their coefficients sum up to 0, i.e., the label cancels out. Analogously, the labels l_y and l_v cancel out.

▷ **Claim 7.** The root label satisfies
$$b \cdot \ell_r = \sum_{v \in V} a_v \mathcal{P}_v .$$

Proof. Consider the sum on the right hand side. Observe that for every vertex v except the root r , the contribution of the label l_v to this sum is cancelled out. To see this, note the following observations:

- The label of a leaf v appears once in \mathcal{P}_v and once in \mathcal{P}_z where z is v 's parent, so it is cancelled out in the sum since the coefficients of these probes are $a_v = -1$ and $a_z = 1$.
- The label of a nonleaf v on level 1 appears in \mathcal{P}_v , \mathcal{P}_z , and \mathcal{P}_x for $x \in ch(v)$ where z is v 's parent, so it is cancelled out since the coefficients of these probes are $a_x = -1$ (k times in the sum), $a_v = 1$, and $a_z = k - 1$.
- The label of a vertex v on level $2 \leq j < h$ appears in \mathcal{P}_v , \mathcal{P}_z , and \mathcal{P}_x for $x \in ch(v)$ where z is v 's parent, so it is cancelled out in the sum since the coefficients of these probes are $a_x = \gamma_{j-1}$ (k times in the sum), $a_v = -\gamma_{j-1} - k \cdot \gamma_{j-2}$, and $a_z = -\gamma_j - k \cdot \gamma_{j-1} = -(-\gamma_{j-1} - k \cdot \gamma_{j-2}) - k \cdot \gamma_{j-1} = -(k - 1) \cdot \gamma_{j-1} + k \cdot \gamma_{j-2}$.

See Figure 3 for an example of a perfect 2-ary (binary) tree.

It follows that the only label that remains in the sum is the root label ℓ_r . This label appears in \mathcal{P}_r and \mathcal{P}_x for $x \in ch(r)$. The coefficients of these probes are $a_x = \gamma_{h-1}$ and $a_r = \gamma_h$. Hence, after all other labels are canceled out, the sum simplifies to $(\gamma_h + k \cdot \gamma_{h-1})\ell_r = b\ell_r$. \triangleleft

▷ **Claim 8.** $b \neq 0$.

Proof. By Eq. (3), $b = -\gamma_{h+1}$. The recursion of Eq. (3) solves to the following explicit formula²:

$$\gamma_j = \frac{2^{-j-1}}{\sqrt{1-4 \cdot k}} \cdot \left((-1 - \sqrt{1-4 \cdot k})^{j+1} - (-1 + \sqrt{1-4 \cdot k})^{j+1} \right) .$$

Hence $b = -\gamma_{h+1} = 0$ if and only if $-\sqrt{1-4 \cdot k} = \sqrt{1-4 \cdot k}$, which is false. \triangleleft

Claims 7 and 8 enable us to extract ℓ_r from the neighborhood probes \mathcal{P} without using any surgical probes. Subsequently, we use the inductive argument to calculate the rest of the labels, proving the claim for h . To do this, we remove r from the tree T , and get k smaller trees T_i whose roots are r_i for $i \in [k]$. As the trees T_i are trees of height $h - 1$, we can use the inductive hypothesis to solve for ℓ on each of the subtrees T_i . The only correction required is that before solving the labels on the subtrees T_i , we need to fix the probe results on the

² the explicit formula can be computed with a computer algebra system

roots, r_i , since probing the vertex r_i in T_i yields a different value than in T (in which r_i has another neighbor, r). Hence, to get the correct labels of these roots in T_i , we need to subtract ℓ_r from \mathcal{P}_{r_i} . More precisely, for $i \in [k]$, let us denote by $\hat{\mathcal{P}}_{r_i}$ the expected outcome of the probe operation on r_i had it been applied to the tree T_i . Then $\hat{\mathcal{P}}_{r_i} = \mathcal{P}_{r_i} - \ell_r$. ◀

4 Mesh graphs and Cartesian Products

In this section we consider mesh graphs, which are Cartesian products of simpler graphs. For example, grid graphs which were studied in previous papers on discrete tomography can be presented as the Cartesian product of two path graphs. In addition to grid graphs, we consider the Cartesian product of a path and a cycle (a tube) and two cycles (a torus).

Graph Products. Let us start with some definitions and notation, following [5]. Given two graphs, $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, the *Cartesian product* of G_1 and G_2 , denoted $G_1 \square G_2$, is the graph $G = (V, E)$ where $V = V_1 \times V_2$ and

$$E = \{(v, u), (v', u) : (v, v') \in E_1\} \cup \{(v, u), (v, u') : (u, u') \in E_2\} .$$

The adjacency matrices of the path and cycle graphs are connected to the product graphs' adjacency matrices by the Kronecker product and Kronecker sum. Given two square matrices A and B of respective sizes n and m , the *Kronecker product* and *Kronecker sum* of A and B are defined, respectively, as

$$A \otimes B \triangleq [a_{ij} \bar{B}] \quad \text{and} \quad A \oplus B \triangleq (A \otimes I_m) + (I_n \otimes B) .$$

The Kronecker sum of the adjacency matrices of two graphs is the adjacency matrix of the Cartesian product graph, i.e., $A_{G_1 \square G_2} = (A_{G_1} \otimes I_{|V_2|}) + (I_{|V_1|} \otimes A_{G_2})$.

The Kronecker sum preserves eigenvalues of its summands in the following way.

► **Theorem 9** ([5]). *Let G_1 and G_2 be graphs. Then, the set of eigenvalues of $A_{G_1 \square G_2}$ is the Minkowski sum of the set of eigenvalues of A_{G_1} and the set of eigenvalues of A_{G_2} , i.e., it is*

$$\Lambda(A_{G_1 \square G_2}) = \{\lambda + \mu \mid \lambda \in \Lambda(A_{G_1}), \mu \in \Lambda(A_{G_2})\} .$$

Cosine at Rational Angles. The required number of surgical probes for the graphs studied herein is determined by the number of solutions to equations that involve trigonometric functions. In particular, rational values of the cosine function at rational angles are of interest. A rational angle is a rational multiple of π . Conway and Jones [6] give a characterization of linear combinations of up to four cosine functions at rational angles that are rational.

► **Theorem 10** ([6]). *Suppose we have at most four distinct rational multiples of π lying strictly between 0 and $\frac{\pi}{2}$ for which some rational linear combination of their cosines is rational but no proper subset has this property. Then the appropriate linear combination is proportional to one from the following list:*

- $\cos(\frac{\pi}{3}) = \frac{1}{2}$
- $-\cos(\varphi) + \cos(\frac{\pi}{3} - \varphi) + \cos(\frac{\pi}{3} + \varphi) = 0 \quad (0 < \varphi < \frac{\pi}{6})$
- $\cos(\frac{\pi}{5}) - \cos(\frac{2\pi}{5}) = \frac{1}{2}$
- $\cos(\frac{\pi}{7}) - \cos(\frac{2\pi}{7}) + \cos(\frac{3\pi}{7}) = \frac{1}{2}$
- $\cos(\frac{\pi}{5}) - \cos(\frac{\pi}{15}) + \cos(\frac{4\pi}{15}) = \frac{1}{2}$
- $-\cos(\frac{2\pi}{5}) + \cos(\frac{2\pi}{15}) - \cos(\frac{7\pi}{15}) = \frac{1}{2}$
- $\cos(\frac{\pi}{7}) + \cos(\frac{3\pi}{7}) - \cos(\frac{\pi}{21}) + \cos(\frac{8\pi}{21}) = \frac{1}{2}$

42:10 The Generalized Microscopic Image Reconstruction Problem

- $-\cos(\frac{2\pi}{7}) + \cos(\frac{3\pi}{7}) + \cos(\frac{4\pi}{21}) + \cos(\frac{10\pi}{21}) = \frac{1}{2}$
- $\cos(\frac{\pi}{7}) - \cos(\frac{2\pi}{7}) + \cos(\frac{2\pi}{21}) - \cos(\frac{5\pi}{21}) = \frac{1}{2}$
- $-\cos(\frac{\pi}{15}) + \cos(\frac{2\pi}{15}) + \cos(\frac{4\pi}{15}) - \cos(\frac{7\pi}{15}) = \frac{1}{2}$

The angles are normalized due to the symmetry of the cosine function. The theorem does not cover the values of the cosine function at 0 and multiples of $\frac{\pi}{2}$. This is characterized by the following theorem.

► **Theorem 11** ([13]). *The only rational values of the circular trigonometric functions at rational multiples of π are 0, $\pm\frac{1}{2}$ and ± 1 for cosine and sine, 0 and ± 1 for tangent and cotangent, and ± 1 and ± 2 for secant and cosecant.*

Grids and Paths. We exploit the fact that mesh graphs are products of simpler graphs to determine the number of surgical probes without having to resort to Theorem 2.

► **Theorem 12.** *Let G be a grid graph of size $n_1 \times n_2$. The number of surgical probes that are sufficient to uncover ℓ is*

$$\mathcal{I}_1^2(n_1)\mathcal{I}_2^3(n_2) + \mathcal{I}_2^3(n_1)\mathcal{I}_1^2(n_2) + 2\mathcal{I}_4^5(n_1)\mathcal{I}_4^5(n_2).$$

In particular, the number of surgical probes to uncover ℓ for grid graphs of any size is at most 4.

Proof. Let G be a grid graph which is the Cartesian product of the two path graphs P_1 and P_2 , of length n_1 and n_2 , resp. By Lemma 1, we look for eigenvalues of A_G that are -1 . By Theorem 9, we need to identify eigenvalues of A_{P_1} and A_{P_2} that add up to -1 . The eigenvalues of A_P where P is a path of length n are given by $2\cos(\frac{\pi j}{n+1})$, for $j \in [1, n]$ (cf. [5]). Hence, we are looking for the number of solutions to the following equation:

$$2\cos\left(\frac{i}{n_1+1} \cdot \pi\right) + 2\cos\left(\frac{j}{n_2+1} \cdot \pi\right) = -1, \quad (4)$$

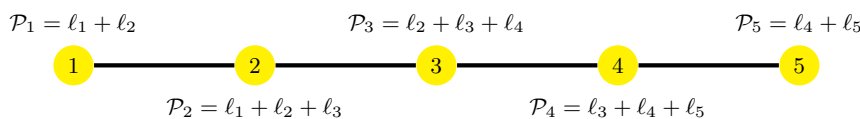
for $i \in [1, n_1]$ and $j \in [1, n_2]$.

Due to Theorem 11 the only rational values the cosine function takes at rational angles are 0, $\pm\frac{1}{2}$ and ± 1 . Since $0 < \frac{i}{n_1+1}, \frac{j}{n_2+1} < 1$, the values 0 and $-\frac{1}{2}$ are the only combination that can satisfy Eq. (4). For $0 < x < 1$, the equations $\cos(\pi x) = 0$ and $\cos(\pi x) = -\frac{1}{2}$ have solutions $x = \frac{1}{2}$ and $x = \frac{2}{3}$, respectively. Hence, Eq. (4) has a solution if $3i = 2(n_1 + 1)$ and $2j = n_2 + 1$. This is the case if $n_1 \bmod 3 \equiv 2$ and $n_2 \bmod 2 \equiv 1$. Here, we have that $\frac{2(n_1+1)}{3} \in [1, n_1]$ and $\frac{n_2+1}{2} \in [1, n_2]$. Due to symmetry, we get another solution if $n_1 \bmod 2 \equiv 1$ and $n_2 \bmod 3 \equiv 2$.

Theorem 10 characterizes linear combinations of cosine functions that have a rational value. Here, the values of a single cosine function are irrational. For a combination of two cosine functions, Theorem 10 states that there is only one such combination: $\cos(\frac{\pi}{5}) - \cos(\frac{2\pi}{5}) = \frac{1}{2}$. Due to symmetry, $\cos(\frac{\pi}{5}) = -\cos(\frac{4\pi}{5})$, and we derive

$$2\cos\left(\frac{4\pi}{5}\right) + 2\cos\left(\frac{2\pi}{5}\right) = -1. \quad (5)$$

Hence, Eq. (4) has a solution if $5i = 4(n_1 + 1)$ and $5j = 2(n_2 + 1)$. For $4(n_1 + 1)$ or $2(n_2 + 1)$ to become a multiple of 5, we need that $n_1 \bmod 5 \equiv 4$ and $n_2 \bmod 5 \equiv 4$. In both cases, $\frac{4(n_1+1)}{5}, \frac{2(n_1+1)}{5} \in [1, n_1]$ and $\frac{4(n_2+1)}{5}, \frac{2(n_2+1)}{5} \in [1, n_2]$ giving two solutions to Eq. (4) if both $n_1 \bmod 5 \equiv 4$ and $n_2 \bmod 5 \equiv 4$. ◀



■ **Figure 4** A path of length $n = 5$. One probe is needed, but not at $i = 3$.

There are many grids where no surgical probes are required to uncover ℓ (e.g. the 3×3 grid). In the worst case, a total of 4 surgical probes is required to uncover ℓ . This is the case if $n_1 = 30i - 1$ and $n_2 = 30j - 1$, for $i, j \in \mathbb{N}$.

When $n_2 = 1$, i.e., graph G is a path, we can also provide a linear time algorithm for uncovering ℓ .

► **Theorem 13.** *Let P be a path with n vertices, where $V = \{1, \dots, n\}$. If $n \bmod 3 \equiv 2$, then a single probe is needed, and it should be at a node i such that $i \bmod 3 \neq 0$. Otherwise, no surgical probes are needed. In both cases, the labels can be discovered in $O(n)$ time.*

Proof. The number of surgical probes follows from Theorem 12

To discover the labels along the path, we use the following procedure. First, the label ℓ_3 can be discovered using $\ell_3 = \mathcal{P}_2 - \mathcal{P}_1$. Iteratively, ℓ_{3i} , for $i = 2, \dots, \lfloor \frac{n}{3} \rfloor$, can be discovered using

$$\mathcal{P}_{3i-1} - \mathcal{P}_{3i-2} + \ell_{3i-3} = (\ell_{3i-2} + \ell_{3i-1} + \ell_{3i}) - (\ell_{3i-3} + \ell_{3i-2} + \ell_{3i-1}) + \ell_{3i-3} = \ell_{3i} .$$

If $n \bmod 3 \neq 2$, we discover either ℓ_n or ℓ_{n-1} in the last iteration. We may discover the other value due to $\mathcal{P}_n = \ell_{n-1} + \ell_n$. Given ℓ_n and ℓ_{n-1} , we may discover the rest of the labels going from right to left. If $n \bmod 3 \equiv 2$, then probe any vertex $i \in [n]$, such that $i \bmod 3 \neq 2$. The probe splits the path into two sub-paths of length n_1 and n_2 , where $n_1, n_2 \bmod 3 \neq 2$. The rest of the labels can be discovered as above, for each sub-path. See Figure 4. ◀

Tubes and Cycles. A tube graph is a Cartesian product of a path and a cycle graph. We denote the length of the path by n_1 and the length of the cycle by $n_2 \geq 3$.

► **Theorem 14.** *Let T be a tube graph of dimensions $n_1 \times n_2$. The number of surgical probes that are sufficient to uncover ℓ is*

$$2\mathcal{I}_1^2(n_1)\mathcal{I}_0^3(n_2) + \mathcal{I}_2^3(n_1)\mathcal{I}_0^2(n_2) + 2\mathcal{I}_2^3(n_1)\mathcal{I}_0^4(n_2) + 4\mathcal{I}_4^5(n_1)\mathcal{I}_0^5(n_2).$$

In particular, the number of surgical probes to uncover ℓ for any tube graph is at most 9.

Proof. Let T be a tube graph which is the Cartesian product of a path graphs P and a cycle C , of length n_1 and n_2 , resp. The eigenvalues of A_P are given by $2 \cos(\frac{\pi i}{n_1+1})$, for $i \in [1, n_1]$, and the eigenvalues of A_C are given by $2 \cos(\frac{2\pi j}{n_2})$, for $j \in [0, n_2 - 1]$ (cf. [5]). By Theorem 9 and Lemma 1, we are interested in solutions of the equation

$$2 \cos\left(\frac{i}{n_1+1} \cdot \pi\right) + 2 \cos\left(\frac{2j}{n_2} \cdot \pi\right) = -1. \tag{6}$$

Since $0 < \frac{\pi i}{n_1+1} < \pi$, $\cos(\frac{\pi i}{n_1+1})$ has the rational values $-\frac{1}{2}, 0, \frac{1}{2}$, and since $0 \leq \frac{2\pi j}{n_2} < 2\pi$, $\cos(\frac{2\pi j}{n_2})$, has the rational values $-1, -\frac{1}{2}, 0, \frac{1}{2}, 1$, due to Theorem 11. This gives us the solutions to Eq. (6) where both cosine functions have rational values. The following combinations satisfy Eq. (6):

42:12 The Generalized Microscopic Image Reconstruction Problem

- $\cos(\frac{\pi i}{n_1+1}) = -\frac{1}{2}$ and $\cos(\frac{2\pi j}{n_2}) = 0$. The equation $\cos(\pi x) = -\frac{1}{2}$ has one solution $x = \frac{2}{3}$, for $0 < x < 1$. Our question is reduced to $3i = 2(n_1 + 1)$. For $2(n_1 + 1)$ to become a multiple of 3, we need that $n_1 \bmod 3 \equiv 2$. In this case, $\frac{2}{3}(n_1 + 1) \in [1, n_1]$. The equation $\cos(2\pi x) = 0$ has two solutions $x = \frac{1}{4}$ and $x = \frac{3}{4}$, for $0 \leq x < 1$. Hence, either $4j = n_2$ or $4j = 3n_2$. It follows that if n_2 is a multiple of 4, we have that $\frac{n_2}{4}, \frac{3n_2}{4} \in [0, n_2 - 1]$. We get two solutions for Eq. (6) if $n_1 \bmod 3 \equiv 2$ and $n_2 \bmod 4 \equiv 0$.
- $\cos(\frac{\pi i}{n_1+1}) = 0$ and $\cos(\frac{2\pi j}{n_2}) = -\frac{1}{2}$. The equation $\cos(\pi x) = 0$ has one solution $x = \frac{1}{2}$, for $0 < x < 1$. Our question is reduced to $2i = n_1 + 1$. For $n_1 + 1$ to become a multiple of 2, we need that $n_1 \bmod 2 \equiv 1$. In this case, $\frac{1}{2}(n_1 + 1) \in [1, n_1]$. The equation $\cos(2\pi x) = -\frac{1}{2}$ has two solutions $x = \frac{1}{3}$ and $x = \frac{2}{3}$, for $0 \leq x < 1$. Hence, either $3j = n_2$ or $3j = 2n_2$. It follows that if n_2 is a multiple of 3, we have that $\frac{n_2}{3}, \frac{2n_2}{3} \in [0, n_2 - 1]$. We get two solutions for Eq. (6) if $n_1 \bmod 2 \equiv 1$ and $n_2 \bmod 3 \equiv 0$.
- $\cos(\frac{\pi i}{n_1+1}) = \frac{1}{2}$ and $\cos(\frac{2\pi j}{n_2}) = -1$. The equation $\cos(\pi x) = \frac{1}{2}$ has one solution $x = \frac{1}{3}$, for $0 < x < 1$. Our question is reduced to $3i = n_1 + 1$. For $n_1 + 1$ to become a multiple of 3, we need that $n_1 \bmod 3 \equiv 2$. In this case, $\frac{1}{3}(n_1 + 1) \in [1, n_1]$. The equation $\cos(2\pi x) = -1$ has one solution $x = \frac{1}{2}$, for $0 \leq x < 1$. Hence, we need that $2j = n_2$. It follows that if n_2 is a multiple of 2, we have that $\frac{n_2}{2} \in [0, n_2 - 1]$. We get one solution for Eq. (6) if $n_1 \bmod 3 \equiv 2$ and $n_2 \bmod 2 \equiv 0$.

Theorem 10 characterizes a linear combination of two cosines which yields four more solutions to Eq. (6). We derive Eq. (5) again: $2 \cos(\frac{4\pi}{5}) + 2 \cos(\frac{2\pi}{5}) = -1$. We have $\cos(\frac{4\pi}{5}) = \frac{1}{4}(-1 - \sqrt{5})$. The equation $\cos(\pi x) = \frac{1}{4}(-1 - \sqrt{5})$ has one solution $x = \frac{4}{5}$, for $0 < x < 1$. Our question is reduced to $5i = 4(n_1 + 1)$. For $n_1 + 1$ to become a multiple of 5, we need that $n_1 \bmod 5 \equiv 4$. In this case, $\frac{4}{5}(n_1 + 1) \in [1, n_1]$. We have $\cos(\frac{2\pi}{5}) = \frac{1}{4}(-1 + \sqrt{5})$. The equation $\cos(2\pi x) = \frac{1}{4}(-1 + \sqrt{5})$ has two solutions $x = \frac{1}{5}$ and $x = \frac{4}{5}$, for $0 \leq x < 1$. Hence, either $5j = n_2$ or $5j = 4n_2$. It follows that if n_2 is a multiple of 5, we have that $\frac{2n_2}{5}, \frac{4n_2}{5} \in [0, n_2 - 1]$. We get two solutions for Eq. (6) if $n_1 \bmod 5 \equiv 4$ and $n_2 \bmod 5 \equiv 0$.

Since n_1 and n_2 can switch, we get another pair of solutions. The equation $\cos(\pi x) = \frac{1}{4}(-1 + \sqrt{5})$ has one solution $x = \frac{2}{5}$, for $0 < x < 1$. Our question is reduced to $5i = 2(n_1 + 1)$. For $n_1 + 1$ to become a multiple of 5, we need that $n_1 \bmod 5 \equiv 4$. In this case, $\frac{4}{5}(n_1 + 1) \in [1, n_1]$. We have $\cos(\frac{2\pi}{5}) = \frac{1}{4}(-1 + \sqrt{5})$. The equation $\cos(2\pi x) = \frac{1}{4}(-1 - \sqrt{5})$ has two solutions $x = \frac{2}{5}$ and $x = \frac{3}{5}$, for $0 \leq x < 1$. Hence, either $5j = 2n_2$ or $5j = 3n_2$. It follows that if n_2 is a multiple of 5, we have that $\frac{2n_2}{5}, \frac{3n_2}{5} \in [0, n_2 - 1]$. In summary, we get four solutions for Eq. (6) if $n_1 \bmod 5 \equiv 4$ and $n_2 \bmod 5 \equiv 0$. ◀

► **Theorem 15.** *Let C be a cycle of length n . If $n \bmod 3 \equiv 0$, then two probes are needed, and they should be at nodes i, j such that $i - j \bmod 3 \not\equiv 0$. Otherwise, no surgical probes are needed. In both cases, the labels can be discovered in $O(n)$ time.*

Proof. The number of surgical probes follows from Theorem 14

To discover the labels of a cycle, we use the following

- If $n \bmod 3 \equiv 0$, then probe vertex 1 which can be understood as removing the vertex from the cycle. Hence, we are left with a path of length $n - 1$. Since $(n - 1) \bmod 3 \equiv 2$, we can use the procedure described in the proof of Theorem 13 to discover the remaining labels with one additional surgical probe.

- If $n \bmod 3 \equiv 1$, we compute ℓ_1 as follows:

$$\begin{aligned} 3\ell_1 &= (\ell_n + \ell_1 + \ell_2) + \sum_{i=1}^{\lfloor n/3 \rfloor} (\ell_{3i-2} - \ell_{3i-1} - \ell_{3i+1} + \ell_{3i+2}) \\ &= \mathcal{P}_1 + \sum_{i=1}^{\lfloor n/3 \rfloor} (\mathcal{P}_{3i-1} - 2\mathcal{P}_{3i} + \mathcal{P}_{3i+1}) . \end{aligned}$$

- If $n \bmod 3 \equiv 2$, we compute ℓ_1 as follows:

$$\begin{aligned} 3\ell_1 &= (-\ell_n + \ell_1 + \ell_2 + 2\ell_3) + \sum_{i=1}^{\lfloor n/3 \rfloor} (-\ell_{3i-1} - 2\ell_{3i} + \ell_{3i+2} + 2\ell_{3i+3}) \\ &= -\mathcal{P}_1 + 2\mathcal{P}_2 + \sum_{i=1}^{\lfloor n/3 \rfloor} (-\mathcal{P}_{3i} - \mathcal{P}_{3i+1} + 2\mathcal{P}_{3i+2}) . \end{aligned}$$

In the two latter cases we are left with a path where we can uncover the remaining labels without any surgical probes due to Theorem 13. ◀

Tori. A *torus* graph is a Cartesian product of two cycle graphs.

► **Theorem 16.** *Let T be a torus graph of dimensions $n_1 \times n_2$. The number of surgical probes that are sufficient to uncover ℓ is*

$$4\mathcal{I}_0^3(n_1)\mathcal{I}_0^4(n_2) + 4\mathcal{I}_0^4(n_1)\mathcal{I}_0^3(n_2) + 2\mathcal{I}_0^2(n_1)\mathcal{I}_0^6(n_2) + 2\mathcal{I}_0^6(n_1)\mathcal{I}_0^2(n_2) + 8\mathcal{I}_0^5(n_1)\mathcal{I}_0^5(n_2).$$

In particular, the number of surgical probes to uncover ℓ for any torus graph is at most 20.

Proof. Let T be a torus graph which is the Cartesian product of two cycles C_1 and C_2 of lengths n_1 and n_2 , resp. The eigenvalues of A_{C_i} are given by $2 \cos(\frac{2j\pi}{n_i})$, for $j \in [0, n_i - 1]$ (cf. [5]). By Theorem 9 and Lemma 1, we are interested in solutions of the equation

$$2 \cos\left(\frac{2i}{n_1} \cdot \pi\right) + 2 \cos\left(\frac{2j}{n_2} \cdot \pi\right) = -1 . \tag{7}$$

The two summands $\cos(\frac{2i\pi}{n_1})$ and $\cos(\frac{2j\pi}{n_2})$ have the rational values $-1, -\frac{1}{2}, 0, \frac{1}{2}, 1$, due to Theorem 11. This gives us the solutions to Eq. (6) where both cosine functions have rational values. We consider the combinations that satisfy Eq. (7):

- $\cos(\frac{2i\pi}{n_1}) = -\frac{1}{2}$ and $\cos(\frac{2j\pi}{n_2}) = 0$. The equation $\cos(2\pi x) = -\frac{1}{2}$ has two solutions $x = \frac{1}{3}$ and $x = \frac{2}{3}$, for $0 \leq x < 1$. It follows that if n_1 is a multiple of 3, we have that $\frac{n_1}{3}, \frac{2n_1}{3} \in [0, n_1 - 1]$. The equation $\cos(2\pi x) = 0$ has two solutions $x = \frac{1}{4}$ and $x = \frac{3}{4}$, for $0 \leq x < 1$. It follows that if n_2 is a multiple of 4, we have that $\frac{n_2}{4}, \frac{3n_2}{4} \in [0, n_2 - 1]$. We get four solutions for Eq. (7) if $n_1 \bmod 3 \equiv 0$ and $n_2 \bmod 4 \equiv 0$.
- $\cos(\frac{2i\pi}{n_1}) = 0$ and $\cos(\frac{2j\pi}{n_2}) = -\frac{1}{2}$. Similarly, we get four solutions if $n_1 \bmod 4 \equiv 0$ and $n_2 \bmod 3 \equiv 0$.
- $\cos(\frac{2i\pi}{n_1}) = \frac{1}{2}$ and $\cos(\frac{2j\pi}{n_2}) = -1$. The equation $\cos(2\pi x) = \frac{1}{2}$ has two solutions $x = \frac{1}{6}$ and $x = \frac{5}{6}$, for $0 \leq x < 1$. It follows that if n_1 is a multiple of 6, we have that $\frac{n_1}{6}, \frac{5n_1}{6} \in [0, n_1 - 1]$. The equation $\cos(2\pi x) = -1$ has one solution $x = \frac{1}{2}$, for $0 \leq x < 1$. It follows that if n_2 is a multiple of 2, we have that $\frac{n_2}{2} \in [0, n_2 - 1]$. We get two solutions for Eq. (7) if $n_1 \bmod 2 \equiv 0$ and $n_2 \bmod 6 \equiv 0$.
- $\cos(\frac{2i\pi}{n_1}) = -1$ and $\cos(\frac{2j\pi}{n_2}) = \frac{1}{2}$. Similarly, we get two solutions if $n_1 \bmod 6 \equiv 0$ and $n_2 \bmod 2 \equiv 0$.

Theorem 10 characterizes a linear combination of two cosines which yields eight more solutions to Eq. (7). We derive Eq. (5) once again: $2 \cos\left(\frac{4\pi}{5}\right) + 2 \cos\left(\frac{2\pi}{5}\right) = -1$. We have $\cos\left(\frac{4\pi}{5}\right) = -\frac{1+\sqrt{5}}{4}$. The equation $\cos(2\pi x) = -\frac{1+\sqrt{5}}{4}$ has two solutions $x = \frac{2}{5}$ and $x = \frac{3}{5}$, for $0 \leq x < 1$. It follows that if n_1 is a multiple of 5, we have that $\frac{2n_1}{5}, \frac{3n_1}{5} \in [0, n_1 - 1]$. We have $\cos\left(\frac{2\pi}{5}\right) = \frac{-1+\sqrt{5}}{4}$. The equation $\cos(2\pi x) = \frac{-1+\sqrt{5}}{4}$ has two solutions $x = \frac{1}{5}$ and $x = \frac{4}{5}$, for $0 \leq x < 1$. It follows that if n_2 is a multiple of 5, we have that $\frac{n_2}{5}, \frac{4n_2}{5} \in [0, n_2 - 1]$. Since n_1 and n_2 can switch roles, in total, we get eight solutions for Eq. (7) if $n_1 \bmod 5 \equiv 0$ and $n_2 \bmod 5 \equiv 0$. ◀

5 King's Graph and Strong Products

Again, let us start with some definitions. The *Strong product* of two graphs G_1 and G_2 , denoted by $G_1 \boxtimes G_2$, is the graph $G = (V, E)$ where $V = V_1 \times V_2$ and

$$E = \{(v, u), (v', u) : (v, v') \in E_1\} \cup \{(v, u), (v, u') : (u, u') \in E_2\} \cup \{(v, u), (v', u') : (v, v') \in E_1 \text{ and } (u, u') \in E_2\} .$$

The adjacency matrix of $G_1 \boxtimes G_2$ is defined as $A_{G_1 \boxtimes G_2} = \bar{A}_{G_1} \otimes \bar{A}_{G_2} - I_{|V_1||V_2|}$.

► **Theorem 17** ([5]). *Let G_1 and G_2 be graphs. Then, the set of eigenvalues of $A_{G_1 \boxtimes G_2}$ is*

$$\Lambda(A_{G_1 \boxtimes G_2}) = \{(\lambda + 1)(\mu + 1) - 1 \mid \lambda \in \Lambda(A_{G_1}), \mu \in \Lambda(A_{G_2})\} .$$

A king's graph is the Strong product of two paths (see Figure 1d). Underlying a king's graph has the topology of a grid graph. Here, the neighborhood of a vertex corresponds to a rectangular scanning window (see Figure 1c). Again, we use the product-property to determine the number of surgical probes directly, without using Theorem 2.

► **Theorem 18.** *Let G be a king's graph of dimensions $n_1 \times n_2$. The number of surgical probes that are sufficient to uncover ℓ is $\mathcal{I}_2^3(n_1)n_2 + \mathcal{I}_2^3(n_2)n_1 - \mathcal{I}_2^3(n_1)\mathcal{I}_2^3(n_2)$.*

Proof. Due to Theorem 17 the number of required probes is given by the solutions of the equation:

$$\left(2 \cos\left(\frac{i}{n_1 + 1} \cdot \pi\right) + 1\right) \left(2 \cos\left(\frac{j}{n_2 + 1} \cdot \pi\right) + 1\right) = 0 , \quad (8)$$

for $i = [1, n_1]$ and $j = [1, n_2]$. Equation (8) has solutions if one of the factors becomes zero. From the proof of Theorem 12, we know that the equation $2 \cos\left(\frac{i\pi}{n_1 + 1}\right) = -1$ has one solution if $n_1 \bmod 3 \equiv 2$. In this case, we get a solution of Eq. (8) for each value of j if $n_2 \bmod 3 \not\equiv 2$. By symmetry we get n_1 solutions if $n_2 \bmod 3 \equiv 2$ and $n_1 \bmod 3 \not\equiv 2$. If $n_1, n_2 \bmod 3 \equiv 2$, we get $n_1 + n_2 - 1$ solutions for Eq. (8). ◀

We see that king's graphs behave quite differently compared to grid graphs. Here, the number of required surgical probes can be as large as $n_1 + n_2 - 1$.

6 Future directions

We extended the MIR framework by representing the inspected object by an undirected graph. A probe corresponds to a measurement taken over the node and its neighborhood. There are many other potentially interesting types of probes in this general model of graphs. For example, when the specimen is a grid and a probe at a node contains all nodes at distance

at most d from this node. We gave a closed form formula for the number of surgical probes for the case where $d = 1$ (grid graph and king's graph). It would be interesting to obtain such a formula for general d .

In the setting introduced here, the surgical probes have the same costs at each node. However, parts of a specimen might be less accessible and therefore more expensive to probe. It might be interesting to study a variation where a surgical probe at node i implies costs c_i , and the goal is now to uncover the labels with minimum total costs.

Moreover, considering directed graphs instead of un-directed graphs leads to a type of probes that are not symmetric. However, this implies that the adjacency matrix is no longer symmetric and Lemma 1 does not longer hold (see example on page 3 of [5]).

Another possible direction is to determine the minimum number of surgical probes in case ℓ is a binary or integer vector. As mentioned earlier, our results provide an upper bound on the number of required surgical probes. This problem is most likely NP-hard as the work by Gritzmann et al. [9] suggests.

Finally, we conjecture that Theorem 6 can be extended to full k -ary trees.

References

- 1 Andreas Alpers and Peter Gritzmann. Reconstructing binary matrices under window constraints from their row and column sums. *Fundamenta Informaticae*, 155(4):321–340, 2017.
- 2 Andreas Alpers and Peter Gritzmann. Dynamic discrete tomography. *Inverse Problems*, 34(3):034003, 2018.
- 3 Andreas Alpers and Peter Gritzmann. On double-resolution imaging and discrete tomography. *SIAM Journal on Discrete Mathematics*, 32(2):1369–1399, 2018.
- 4 Daniela Battaglino, Andrea Frosini, and Simone Rinaldi. A decomposition theorem for homogeneous sets with respect to diamond probes. *Computer Vision and Image Understanding*, 117(4):319–325, 2013.
- 5 Andries E. Brouwer and Willem H. Haemers. *Spectra of Graphs*. Springer, 2011.
- 6 John Conway and Alice J. Jones. Trigonometric diophantine equations (On vanishing sums of roots of unity). *Acta Arithmetica*, 30(3):229–240, 1976.
- 7 Andrea Frosini and Maurice Nivat. Binary matrices under the microscope: A tomographical problem. *Theor. Comput. Sci.*, 370(1-3):201–217, 2007.
- 8 Andrea Frosini, Maurice Nivat, and Simone Rinaldi. Scanning integer matrices by means of two rectangular windows. *Theoretical Computer Science*, 406(1-2):90–96, 2008.
- 9 Peter Gritzmann, Barbara Langfeld, and Markus Wiegmann. Uniqueness in discrete tomography: three remarks and a corollary. *SIAM Journal on Discrete Mathematics*, 25(4):1589–1599, 2011.
- 10 Gabor T Herman and Attila Kuba. *Discrete tomography: Foundations, algorithms, and applications*. Springer Science & Business Media, 2012.
- 11 Carl D Meyer. *Matrix analysis and applied linear algebra*, volume 71. Siam, 2000.
- 12 Maurice Nivat. Sous-ensembles homogènes de \mathbb{Z}^2 et pavages du plan. *Comptes Rendus Mathématique*, 335(1):83–86, 2002.
- 13 Ivan Niven. *Numbers: rational and irrational*, volume 1. Random House New York, 1961.

Stabilization Time in Minority Processes

Pál András Papp

ETH Zürich, Switzerland
apapp@ethz.ch

Roger Wattenhofer

ETH Zürich, Switzerland
wattenhofer@ethz.ch

Abstract

We analyze the stabilization time of minority processes in graphs. A minority process is a dynamically changing coloring, where each node repeatedly changes its color to the color which is least frequent in its neighborhood. First, we present a simple $\Omega(n^2)$ stabilization time lower bound in the sequential adversarial model. Our main contribution is a graph construction which proves a $\Omega(n^{2-\epsilon})$ stabilization time lower bound for any $\epsilon > 0$. This lower bound holds even if the order of nodes is chosen benevolently, not only in the sequential model, but also in any reasonable concurrent model of the process.

2012 ACM Subject Classification Mathematics of computing → Graph coloring; Theory of computation → Distributed computing models; Theory of computation → Self-organization

Keywords and phrases Minority process, Benevolent model

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.43

Related Version An archive version is available at <https://arxiv.org/abs/1907.02131>.

1 Introduction

If you google “bad wifi”, one advice you will get for sure is to choose the least crowded frequency in order to minimize interference with your neighbors. Unfortunately, this least crowded frequency may change again if some of your neighbors do the same.

Frequency allocation is a familiar example of *minority processes* in graphs: given a graph, a set of colors, and an initial coloring of the nodes with these colors, a minority process is a process where each node, when given the chance to act, modifies its color to a color that has the smallest number of occurrences in its neighborhood. This results in a dynamically changing coloring, which is essentially a form of distributed automata. Minority processes arise in various fields of economics [12] or social science [3] when players are motivated to differentiate from each other, but they also emerge in cellular biology [4] or crystallization mechanics [2].

A minority process is said to stabilize when no node has an incentive to change its color anymore. The aim of the paper is to understand how long it takes until such a minority process reaches a stable state. We study the process in several different models, some of them sequential, some concurrent. In sequential models, when only one node at a time can change its color, stabilization time depends on the choice of the order of nodes. Hence, the model can further be subdivided into three cases, depending on whether the order of acting nodes is specified benevolently (trying to minimize stabilization time), adversarially (trying to maximize stabilization time), or randomly.

On the other hand, in concurrent models, multiple nodes are allowed to switch their color at the same time. However, if two (or more) neighboring nodes continuously keep on forcing each other to switch their color, the system may never stabilize. The simplest such example is a graph of two connected nodes that have the same initial color, and keep on switching



© Pál András Papp and Roger Wattenhofer;

licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 43; pp. 43:1–43:19

Leibniz International Proceedings in Informatics



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

to the same new color in every step. We also study concurrent models that exclude this behavior, as it is unrealistic in many application areas where neighbors are unlikely to switch at the exact same time.

In any model where simultaneous neighboring switches are excluded, it is easy to prove a $O(n^2)$ upper bound on stabilization time for minority processes. Initially, some (maybe even all) of the at most $O(n^2)$ edges in the graph are monochromatic (i.e., they have a *conflict*). When a node switches its color to the minority color in its neighborhood (but its neighbors do not change color in the same step), then the number of conflicts on the adjacent edges strictly decrease. Since the original number of conflicts is $O(n^2)$ and the overall number of conflicts decreases by 1 at least in each step, the number of steps is limited to $O(n^2)$.

However, this raises a natural question: are there example graphs that exhibit this naive upper bound? Or is there a significantly lower (e.g. linear) upper bound on stabilization time in some models? While these questions are already answered for the “dual” problem of majority processes (when nodes switch to the most frequent color in their neighborhood), for the case of minority processes, they have remained open so far.

The main contributions of the paper are constructions that prove lower bounds on stabilization time of minority processes. As a warm-up, we present a simple example in Section 4 which shows that in the sequential adversarial model, stabilization may take $\Theta(n^2)$ steps. Our main result is a construction proving that stabilization can also take superlinear time in the sequential benevolent case. We first present a graph and an initial coloring in Section 5 where any selectable sequence lasts for $\Omega(n^{3/2})$ steps. Then in Section 6, we outline how a recursive application of this technique leads to a stabilization time of $\Omega(n^{2-\epsilon})$ for any $\epsilon > 0$, almost matching the upper bound of $O(n^2)$. This is an interesting contrast to majority processes, where stabilization time is bounded by $O(n)$ in the benevolent case. Furthermore, our construction shows that this almost-quadratic lower bound holds not only in the sequential model, but also in any reasonable concurrent setting.

2 Related work

While there is a wide variety of results on both minority and majority processes, majority processes have been studied much more extensively. Recently, [5] has shown that stabilization time in majority processes can be superlinear both in the synchronous model, and in the sequential model if the order is chosen by an adversary. However, [5] has also shown that stabilization always happens in $O(n)$ time in the sequential benevolent model. In case of majority processes in weighted graphs, a $2^{\Theta(n)}$ lower bound on stabilization time was also shown in [11].

Other aspects of majority processes have also been studied thoroughly, especially in the synchronous model. Results on majority processes include basic properties [8], their behavior on random graphs [6], complexity results on determining stabilization time [10], minimal sets of nodes that dominate the process [7], and the existence of stable states in the process [1].

In contrast to this, the dynamics of minority processes has received less attention. The stabilization of minority processes has only been studied in special classes of graphs, including tori, cycles, trees and cliques [14, 15, 16]. These studies are mostly conducted only in the synchronous or the sequential random model. More importantly, these results study a different variant of the minority process, which considers the closed neighborhood of nodes, and thus can result in significantly larger (possibly exponential) stabilization time, even in the unweighted case. An experimental study of the processes on grids is also available in [14].

In weighted graphs, it has recently been shown in [13] that stabilization of minority processes can take $2^{\Theta(n)}$ steps in various models, matching a straightforward exponential upper bound in the weighted case. However, the constructions of [13] use exponentially large node or edge weights to obtain these results; as such, the same techniques are not applicable in the unweighted case.

Besides these studies on the dynamics of the process, there are also numerous theoretical results on stable states in minority processes. These include complexity results on deciding the existence of different stable state variants [12], characterization of infinite graphs with a stable state [17], and analysis of price of anarchy in such states as local minima [12]. In the work of [9], it is also shown that slightly modified minority processes, based on distance-2 neighborhood of nodes, can provide better local minima at the cost of larger (but still polynomial) stabilization time.

However, in contrast to majority processes, the stabilization time of minority processes in general unweighted graphs has remained unresolved so far.

3 Definitions and background

3.1 Models

In the paper, we primarily focus on the following models:

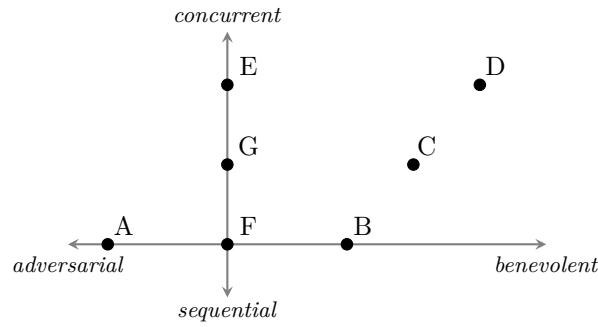
- A. Sequential adversarial:** In every step, only one node switches. The order of nodes is specified by an adversary who maximizes stabilization time.
- B. Sequential benevolent:** In every step, only one node switches. The order is specified by a benevolent player who minimizes stabilization time.
- C. Independent benevolent:** In every step, the benevolent player is allowed to choose any independent set of switchable nodes, and switch them simultaneously.
- D. Free benevolent:** In each step, the benevolent player is allowed to choose any set of switchable nodes, and switch them simultaneously.

However, our lower bounds extend to a range of other popular models:

- E. Concurrent synchronous:** In every step, all switchable nodes switch simultaneously.
- F. Sequential random:** In every step, only one node switches, chosen uniformly at random among the switchable nodes.
- G. Concurrent random:** In every step, every switchable node switches with probability p , independently from other nodes.

An intuitive illustration of these models is shown in Figure 1. The vertical axis shows how concurrent a model is, the horizontal shows how wide is the set of opportunities it grants the player to speed up / slow down stabilization. In the case of majority processes, models A and E are shown to take superlinear time to stabilize for some graphs, but model B always stabilizes in linear time [5]. However, we prove that for minority processes, even model B can take superlinear time. Models C and D grant even wider sets of possible (concurrent) moves for the benevolent player, which may drastically reduce the number of steps in some cases; however, we show that the same lower bound holds even if such moves are available.

Note that models A, B, C and F exhibit a natural $O(n^2)$ upper bound on stabilization time, as the overall number of conflicts decreases in each step by at least 1. On the other hand, models D, E or G may allow neighboring nodes to switch at the same time, and thus in these models, some nodes may keep on endlessly changing colors. However, our constructions specifically ensure that connected nodes are never switchable at the same time, and thus for these particular graphs, the process stabilizes in any of the models.



■ **Figure 1** Properties of the listed models.

Through most of the analysis in the paper, we focus on the sequential models. We first show a simple construction with $\Theta(n^2)$ stabilization time in model A. We then present a more complex construction to first show $\Omega(n^{3/2})$, and then $\Omega(n^{2-\epsilon})$ stabilization time in model B. It then follows from a few observations that these latter constructions also have the same stabilization time in models C and D. Since model D provides the widest set of opportunities from all models, this implies the same lower bound for each of the listed models.

3.2 Preliminaries

Throughout the paper, we consider simple, unweighted, undirected graphs. Graphs are denoted by G , their number of nodes by n , and the maximum degree in the graph by Δ .

Given a graph G on the vertex set V , an *independent set* is a subset of V such that no two nodes in this subset are connected. A *coloring* of the graph with k colors is the assignment of one of the colors (numbers) from $\{1, 2, \dots, k\}$ to each of the nodes. If two nodes share an edge and are assigned the same color, then the nodes have a *conflict* on this edge.

Our process consists of discrete time steps (*states*), where we have a current coloring of the graph in every state. When a node v is currently colored c_1 , but there exists a color c_2 such that the neighborhood of v contains strictly less nodes colored c_2 than nodes colored c_1 , then the node is *switchable* (since the node could reduce its number of conflicts by changing its color). The process of v changing its color is *switching*. Nodes always make locally optimal solutions, that is, they switch to the color which is least frequent in their neighborhood. In case of multiple optimal colors, related work on majority processes considers different tie-breaking rules. However, our constructions ensure that a tie can never occur, and thus our bounds hold for any tie-breaking strategy.

The *minority process* is a sequence of steps, where each step is described by a set of nodes that switch. Note that we only consider valid steps, where every chosen node is switchable.

A state is *stable* when no node in the graph is switchable; a system *stabilizes* if it reaches a stable state. *Stabilization time* is the number of steps until the process stabilizes. Note that in case of model E, papers studying majority processes often use a different definition of stabilization, based on periodicity. However, our constructions ensure that the process always ends in a stable state, thus for the graphs in the paper, the two definitions of stabilization are equivalent.

In our examples, we will consider the case of having only two available colors, black and white. However, as discussed in Section 3.3, our lower bounds are easy to generalize to any number of colors.

The restriction to two colors allows us to introduce some helpful terminology. Consider a node v at a given state of the process. If v has v_s neighbors with the same color as v , and v_o neighbors with the opposite color, the number $v_o - v_s$ is called the *balance* of v . Note that if one of the neighbors of v switches, then the balance of v either increases or decreases by 2 (which shows that the parity of the balance of v can never change). The definition also implies that v is switchable if and only if its balance is negative. Switching v changes the sign of its balance.

3.3 General tools in the constructions

Groups. We use the notion *group* to refer to a set of nodes that have the same initial color and the exact same set of neighbors (hence, groups are independent sets). Groups are, in fact, only a tool to consider certain nodesets together as one entity for simpler presentation. They will be shown as only one node with double borders in the figures, with the size of the group indicated in brackets.

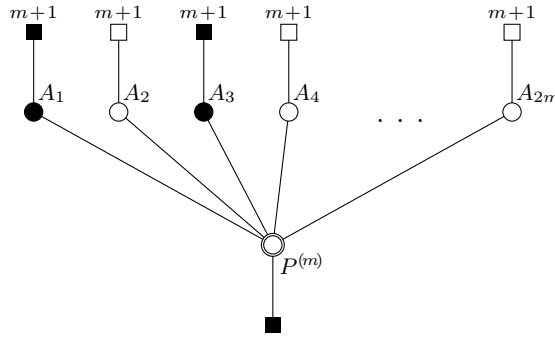
In the adversarial case, we will only consider sequences that switch groups together (i.e. consecutively in any order). In the benevolent case, groups will be switched together in the sense that if a node in the group switches, then all other nodes in the group will also switch before any neighbor of the group becomes switchable; this property is enforced by the graph construction. The more complicated definition in the benevolent case is due to the fact that we have to consider every possible sequence that the player can choose. Technically, in some sequences, a group might not be switched consecutively (it might be interrupted by switches in other, distant parts of the graph), but the outcome will still be equivalent to switching them consecutively.

Fixed nodes. Given a graph G , let us add two more set of nodes F_w, F_b to the graph such that $|F_w| = |F_b| = n + 1$, and v_w and v_b are connected for all $v_w \in F_w, v_b \in F_b$. Let the color of F_w and F_b initially be white and black, respectively. The nodes in F_w and F_b will be referred to as *fixed nodes*, and we will connect them to some of the nodes in our original graph. Note that these fixed nodes already have $n + 1$ neighbors of the opposite color, and can never have more neighbors of the same color (as they can have at most n neighbors G), so their color is indeed fixed and they can never switch.

Such fixed nodes are widely used in our construction; we can allow any node in G to have up to $\Delta + 1$ fixed neighbors of either color. The introduction of fixed nodes increases the graph size only by a constant factor (to $3n + 2$), so all lower bounds expressed as a function of n will still be of the same magnitude as a function of $3n + 2$. Therefore, for ease of presentation, we still use n to denote the number of nodes in the graph *without* the extra fixed nodes, and express our bounds as a function of n .

Fixed node neighbors are denoted by squares in the figures, with the multiplicity written beside the square (if more than 1). We always draw separate squares for distinct nodes, even though the corresponding fixed node sets might overlap. This is because fix node connections are thought of as a “property” of the node, introducing an offset into its initial balance.

Generalization to more colors. While the paper discusses the case of two colors, a simple idea allows a generalization to any constant number of colors k . Assume we have a construction G on n nodes, showing a lower bound on stabilization time with two colors; we can simply add sets of nodes F_3, F_4, \dots, F_k of size $\Delta + 1$ such that they form a complete multipartite graph, and connect all these new nodes to all nodes in G . Let us color the nodes in F_i with color i .



■ **Figure 2** Construction with an adversarial sequence of $\Theta(n^2)$ switches.

None of the original nodes in G will ever assume any of the colors 3, 4, ..., k , since they always have $\Delta + 1$ neighbors of these colors, while they have strictly less (at most Δ) neighbors of colors 1 and 2. Nodes in F_i will never have any incentive to switch, since they have no conflicts at all. Thus the process will behave as if the graph only consisted of G with colors 1 and 2. As the new nodes only increase the graph size by a constant factor, we receive an example with the same magnitude of running time, but with k colors.

With the same technique, our lower bound of $\Omega(n^{3/2})$ can also be generalized to the case of up to $\Theta(\sqrt{n})$ colors; details of this are discussed in Appendix B.

4 Sequential adversarial model

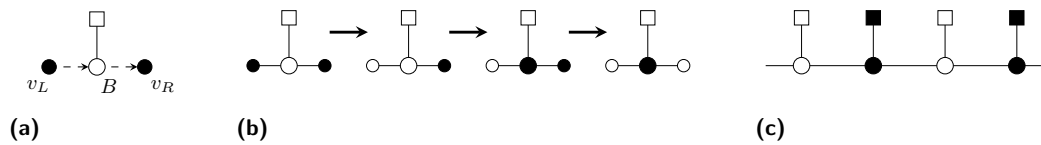
We first present a simple example where model A takes $\Omega(n^2)$ steps. Our construction, shown in Figure 2, consist of a group P of size m (for some parameter m), initially colored white, and $2m$ distinct nodes A_1, A_2, \dots, A_{2m} , such that A_i is initially colored black for odd values of i and white for even i . Let us connect all nodes A_i to P , and add one more fixed black node that is connected only to P . Finally, let us connect each A_i to $m + 1$ fixed nodes of the same color as A_i . Recall that although the figure shows multiple squares, there are in fact only $n + 1$ fixed black and $n + 1$ fixed white nodes in the graph altogether.

In this graph, P has a balance of 1 initially, while black A_i have a balance of -1 and white A_i have a balance of $-(2m + 1)$. Note that even after execution begins, until A_i is switched for the first time, it will have $m + 1$ fixed neighbors of the same color and at most m neighbors of the opposite color (depending on the current color of P), and thus a negative balance. Therefore, each A_i is switchable anytime if it has not been switched before.

Consider the following sequence of adversarial moves in this graph: the player first decides to switch A_1 , then P , then A_2 , then P again, then A_3, P, \dots, A_{2m} , and finally P again. As each A_i is used only once, they are clearly all switchable. As for P , its balance first changes from 1 to -1 , when changing A_1 to white, but increases back to 1 when we switch P itself. Then it changes to -1 once again after changing A_2 , so it is switchable again, and so on: each time we switch an A_i , we change it to the same color that P currently has, decreasing P 's balance to -1 , which increases back to 1 again as we switch P . Therefore, this strategy is indeed a sequence of valid switches.

Since P contains m nodes and is switched $2m$ times in this sequence, this alone contributes to $2m^2$ switches. Altogether, we have $3m$ nodes in the graph (without fixed nodes), allowing us a choice of $m = \frac{n}{3}$. This gives us a sequence with at least $\frac{2}{9}n^2$ steps.

► **Theorem 1.** *There exists a graph construction with $\Omega(n^2)$ stabilization time in model A.*



■ **Figure 3** Simple relay gadget (a), the steps of its operation (b), and a chain of relays (c).

5 Construction for benevolent models

We now present a construction with $\Omega(n^{3/2})$ stabilization time in benevolent models. Note that it is much more involved to find an example where benevolent models take $\omega(n)$ steps, since in such a construction, we have to ensure that any possible sequence lasts for a long time. In order to have an easy-to-analyze construction, our graph will, at any point in time, contain only one, or a small set of nodes that are switchable, and switching this or these nodes enables the next such set of nodes (i.e., makes them switchable). This way, the switchable point “propagates” through the graph, and the benevolent player has no other valid move than to follow this path of propagation that has been designed into the graph.

The general idea behind the construction is to have a linearly long chain of nodes which is propagated through multiple times. After each such round, the propagation enters a different branch of further nodes; this branch resets the chain for the following round, and then also triggers the following round of propagation (as outlined later in Figure 11).

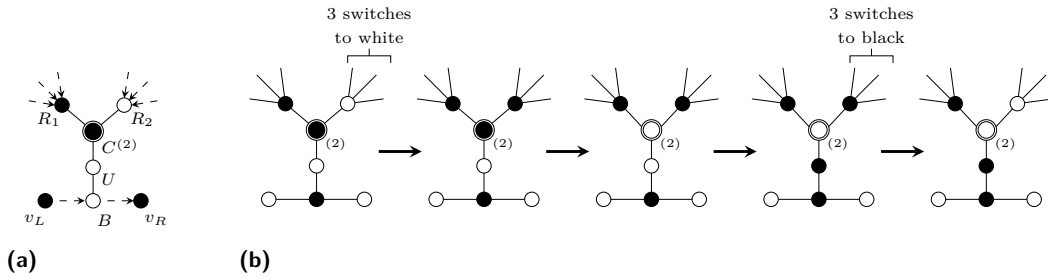
Due to the complexity of the construction, we do not describe it directly; instead, we define smaller functional elements (*gadgets*) that execute a certain task. We then use these gadgets as building blocks to put our example graph together. This section outlines the tasks and main properties of the gadgets; a detailed description and analysis of each gadget can be found in Appendix B. While the concrete gadget designs are specific to minority processes, they are built on general ideas and techniques for benevolent models; as such, we hope they may inspire similar solutions in the analysis of related processes or cellular automata.

When describing a gadget, the edges connecting the gadget to other nodes in the graphs are drawn as dashed lines in the figures, with the external node usually denoted by v (possibly with some subscript). Although our graph is undirected, we often refer to such edges as *input* or *output* edges of the gadget, and also show this direction in the figures. This will refer to the role that the external node plays in the functionality of the gadget. That is, whenever the gadget is used in our constructions, it is triggered by (some of) its input nodes switching, and upon completing its task, the gadget makes (some of) its output nodes switchable.

Naturally, as in the entire graph, the role of the two colors is always interchangeable within the gadgets. Therefore, we only present each such gadget in one color variant.

Due to the complexity of the construction, we have also verified its correctness through implementing the process. A discussion of these simulations is available in Appendix C.

Simple relay. As our most basic tool to propagate the only possible point of switching, we use the *simple relay* shown in Figure 3a. A simple relay consists of a base node B , connected to a fixed node of the same color, and two further nodes outside of the gadget, which initially have the opposite color as B . Until neither of v_L and v_R switch, B has positive balance and cannot switch either. However, as soon as v_L switches to the color of B , B becomes switchable, and as B switches, this propagates the point of change to its other neighbor v_R (as shown in Figure 3b).



■ **Figure 4** Rechargeable relay gadget (a) and the steps of its operation (b).

Note that connecting alternating-colored relays into a chain already gives a simple example of linear stabilization time (see Figure 3c). If the leftmost (white) relay’s base node is connected to a fixed white node, then the only available sequence of moves is to switch the base nodes in the relays one by one from left to right, resulting in a sequence of n steps.

Through the concept of input and output nodes, relays essentially allow us to connect other, more sophisticated gadgets in our constructions. If some gadget has an output node v_1 and another gadget has an input node v_2 , we can add a chain of relays between v_1 and v_2 , ensuring that once v_1 switches, it will be followed by v_2 eventually. Due to this role, relays are not shown explicitly in our final overview figure of the construction, but only represented by arrows, indicating the direction of propagation between more complex gadgets.

Rechargeable relay. A more sophisticated version of a relay is the *rechargeable relay* shown in Figure 4a. In such a relay, node B is extended by an upper node U , a control group C of size 2, and two recharge nodes R_1, R_2 , the role of which are interchangeable. Besides v_L and v_R , the nodes R_1 and R_2 also have edges to some external nodes. It is always ensured that the initial balance of R_1 and R_2 from these upper neighbors (that is, with C ignored) is exactly 3.

As in case of a simple relay, if v_L switches, then B itself can switch, followed by v_R . Now assume that in this “used” phase of the relay, some outside circumstance changes 3 neighbors of node R_2 from black to white, and thus its balance changes from the current value of 5 to -1 (the relay is *recharged*). Then R_2 can switch to black, making C and in turn U switch, too. Finally, assume that some other outside circumstance then changes the balance of R_2 from 5 to -1 again (known as *resetting* the relay); then R_2 will switch back to white (with a new balance of 1), and we end up in the initial state of a rechargeable relay of the opposite color. The steps of the process are shown in Figure 4b.

This is exactly the essence of this gadget: it is a relay which can be used the same way multiple times. Connecting such gadgets into a chain in the same fashion as Figure 3c, we get a chain that can propagate the point of change not only once, but multiple times if “recharged” through their upper connections between two such propagations.

Recharging system. The rechargeable relay suggests that it is useful to have a tool to “recharge” some nodes, i.e. to decrease their balance by switching some of their neighbors to the color they currently have. To execute this task efficiently on many nodes, we present a *recharging system*.

For the first version of this gadget, assume a setting where there is a set X of m black nodes, and we want to decrease the balance of each of these nodes by 2 (i.e., change exactly one white neighbor of each of them to black). A *basic recharging system*, shown in Figure 5,

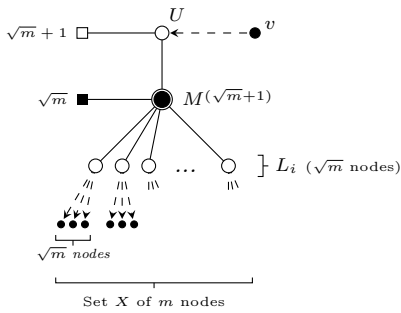


Figure 5 Basic recharging system.

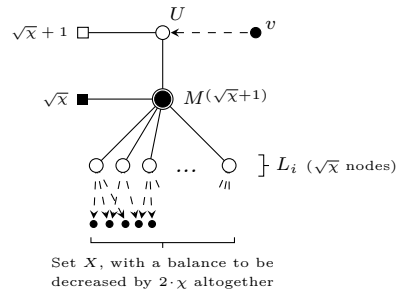


Figure 6 Generalized recharging system.

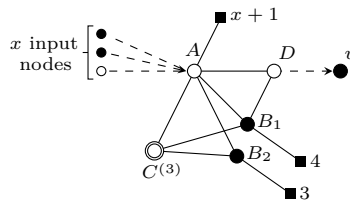


Figure 7 AND gate.

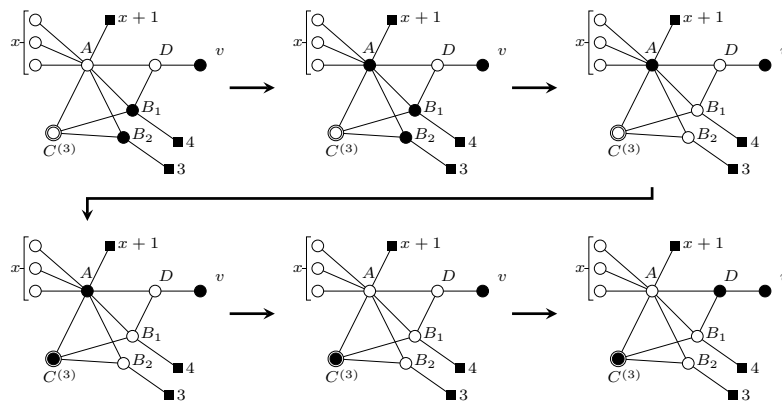
can execute this task while using only $O(\sqrt{m})$ nodes. The gadget is organized into 3 levels: a single node U in the upper level, a group M of $\sqrt{m} + 1$ nodes in the middle level, and \sqrt{m} distinct nodes L_i in the lower level. Each lower level node is connected to \sqrt{m} different nodes in X , thus exactly covering the nodes of X . The gadget operates in a top-to-bottom fashion: once v switches, U turns black, followed by M turning white. Once all nodes in M are switched, the nodes L_i all decide to switch, too.

The key idea in the design of the gadget is that each node L_i has strictly more neighbors in M than in X . This ensures that as long as M is black, the nodes L_i always have a positive balance, regardless of the current color of their neighbors in X . Therefore, no node in the gadget can ever switch before the node U is triggered.

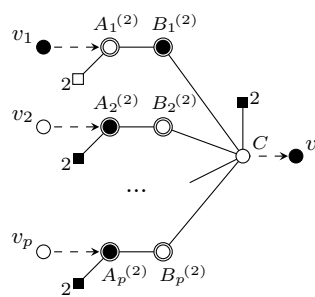
We can use this insight to create a similar gadget for a more general setting. Assume that we similarly have a set X of m black nodes, but instead of decreasing their balance by 2, we want to decrease the balance of each node in X by some specific (possibly different) even value, denoted by $2x_1, 2x_2, \dots, 2x_m$ (i.e., for the j^{th} node in X , we want to change x_j of its white neighbors to black). Let us denote the sum $\sum_{j=1}^m x_j$ of these values by χ .

We can achieve this using a similar construction, shown in Figure 6. In this *generalized recharging system*, we allow multiple nodes L_i to be connected to the same node in X : if a node in X has a corresponding value $2x_j$, then it has exactly x_j neighbors in the lower level of the system. This ensures that once all the nodes L_i switch, the new balance of each node in X is exactly as desired. The number of nodes in the gadget can be minimized by placing $\sqrt{\chi}$ nodes L_i in the lower level, each with $\sqrt{\chi}$ neighbors in X ; this way, the overall number of edges going into the set X from the gadget is exactly χ as required. To ensure that the neighborhood of each L_i is dominated by M , we choose the size of group M to be $\sqrt{\chi} + 1$.

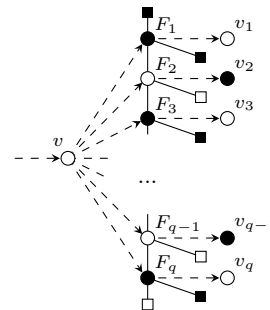
AND gate. Another ingredient we use is an AND gate. As its name suggests, this gadget has x input edges from a set of nodes X , and once all nodes in X have switched to the same color (say, white), the gadget triggers a change in another part of the graph.



■ **Figure 8** Operation of an AND gate. In the end, node D switches to black, making v switchable.



■ **Figure 9** Join gadget.



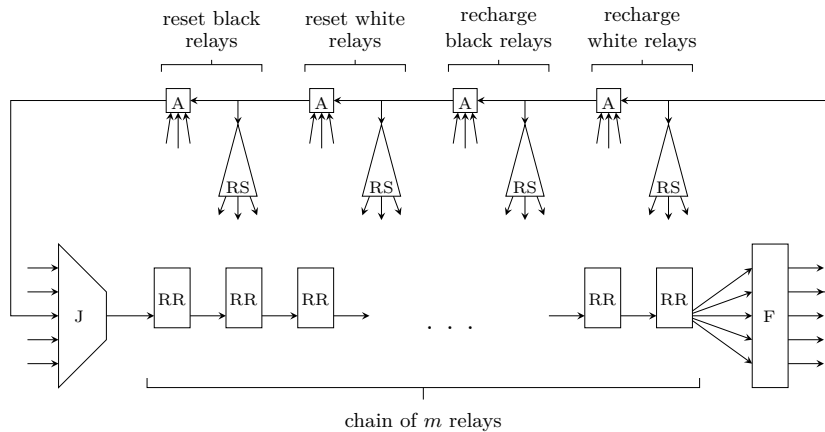
■ **Figure 10** Fork gadget.

Note that we could achieve this functionality with a single node, by carefully setting its initial balance such that it switches exactly when all inputs have the desired color. However, AND gates are used to “check” the state of specific nodes in the construction, and as such, it is unfortunate that this check also affects the nodes that are being checked: once the node in this simple AND gate switches, the balance of all input nodes in X will increase by 2. It would be much better to have a gadget that can perform this task without having any effect on the nodes in X .

For this purpose, consider the gadget in Figure 7, which is connected to the nodes in X on the input side and a black node v on the output side. Once all nodes in X are white, node A switches, followed by B_1 and B_2 , and then by C . With C switched, A decides to switch back to its original color white. However, since now both A and B_1 are white, this finally switches D to black, triggering a change in the output node v (Figure 8). The usefulness of the gadget lies in the fact that by the end of this sequence, A is switched back to its original color, and thus the balance of nodes in X is again the same as it was in the beginning.

Join and fork gadgets. Finally, we need two small gadgets in the construction to fork and join the control sequence at the ends of our main relay chain.

A join gadget, shown in Figure 9, connects a specific number of input nodes v_i to an output node v . When an input node v_i switches, then so does A_i and then B_i in the corresponding input branch, which also switches C and triggers node v . Then when v_{i+1} later switches at some point, the same thing happens to the next input branch and C again, only with the two colors swapping roles. Thus if the nodes v_1, v_2, \dots are switched one after another in this order, then each of these input switches make the output node v switch again.



■ **Figure 11** Overview of the whole construction, with one branch shown in detail. Rechargeable relays (RR), Recharging systems (RS), AND gates (A), Joins (J) and Forks (F) are explicitly shown.

The fork gadget of Figure 10, on the other hand, is responsible for receiving triggers from a given input node v , and directing the propagation to a new branch (a new output node v_i) every time. When v first switches, only v_1 becomes switchable. Similarly, after v is switched for the i^{th} time, only v_i becomes switchable, and thus the gadget triggers the i^{th} branch of output.

Assembling the pieces. Our final graph construction (shown in Figure 11) has two defining parameters m and r . The base of the construction is a chain of m rechargeable relays, connected to a join gadget of r branches and a fork gadget of $r-1$ branches. For each $i \in \{1, \dots, r-1\}$, we add a sequence of gadgets (a *branch*) to connect the i^{th} output of the fork to the $i+1^{\text{th}}$ input of the join gadget, which is responsible for recharging the relay chain.

Each branch consists of recharging systems connected to our main chain. First let us consider the rechargeable relays where node U is currently white (either the even or the odd ones; relays at positions of the same parity are all in the same state). We first need a recharging system to recharge all these relays, and then we need another system to reset the relays. We need similarly 2 recharging systems for the other half of the relays which are in the opposite color phase.

Finally, we need to force the player to indeed execute these changes on the relays. For that, we insert an AND gate after each recharging system, which checks if all switchable nodes have indeed been switched before moving on. The output of the AND gate is then used to enable the next recharging systems (or the next input of the join gadget).

This construction ensures that the player has no other choice than to go through the relay chain, follow the next branch from the fork, recharge and reset all the relays, and start going through the relay chain again. Since the chain consists of m relays and it is traversed r times in this process, the switches in the chain add up to $m \cdot r$ steps altogether.

Of course, one also needs to introduce a starting point (initially switchable node) into the construction. This can be done by replacing v_1 in the join gadget by a fixed white node.

Let us consider the number of nodes in the construction. Since rechargeable relays consist of constantly many nodes, the size of the relay chain is $O(m)$. The size of the join and fork gadgets is $O(r)$. Finally, each of the $r-1$ recharging branches consist of constantly many recharging systems, AND gates and simple relays; since the latter two have constant

43:12 Stabilization Time in Minority Processes

size, branch size is dominated by the size of the recharging systems. Each such system is connected to $\frac{m}{2}$ relays, and thus needs to reduce the balance of $O(m)$ nodes by a constant value of 6. This implies that each recharging system needs $O(\sqrt{m})$ nodes.

This shows that we can choose $r = \Theta(\sqrt{m})$ and $m = \Theta(n)$ for our parameters. Our graph then contains $O(m) + O(r) + r \cdot O(\sqrt{m}) = O(n)$ nodes, so it is indeed a valid setting with the proper choice of constants.

To investigate runtime, it is enough to consider the switches in the main relay chain. Each of the $\Theta(n)$ relays has a base node that is switched $\Theta(\sqrt{n})$ times, adding up to a total of $\Omega(n^{3/2})$ switches.

► **Theorem 2.** *There exists a graph construction with $\Omega(n^{3/2})$ stabilization time in model B.*

Note that in the previous construction, whenever any of the base nodes of the relay chain are switchable, there is no other switchable node in the entire graph. This implies that even in the independent benevolent case, the player has no other option than to select this single node, so the number of minimal switches is $\Omega(n^{3/2})$ even if we assume the independent benevolent model.

In fact, one can observe that the construction also ensures that regardless of the choices of the player, the set of switchable nodes is always an independent set at any point in the process. Hence models C and D are in fact the same in this graph, and thus the lower bound also holds for model D. This then implies the same bound for all the remaining models.

► **Corollary 3.** *There is a graph construction with $\Omega(n^{3/2})$ stabilization time in models C–G.*

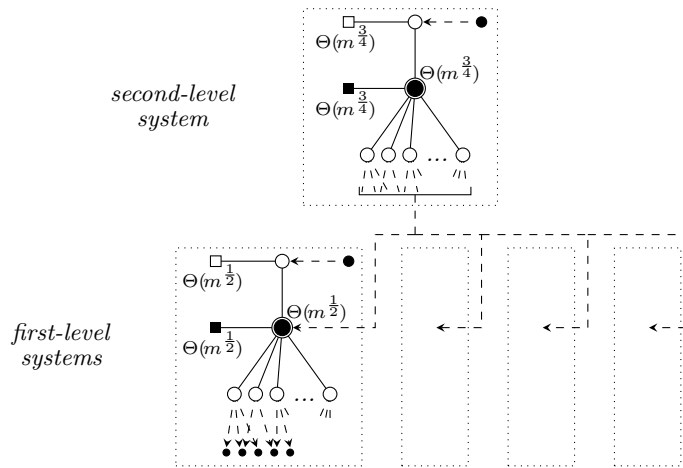
6 Recursive construction

We now briefly outline the modification idea that provides the almost tight lower bound of $\Omega(n^{2-\epsilon})$. A more detailed discussion of the construction can be found in Appendix A.

The key idea is to make the recharging systems themselves also rechargeable, so that they can recharge the same output nodes repeatedly. Note that once a recharging system has been used, the color of its nodes is exactly that of a recharging system of the opposite color. Thus, if we reset the balance of each node in the system to its initial value, we can use the system again to recharge the same output nodes again. More specifically, given a used recharging system, we need to restore the balance of M and U to 1 in order to obtain a recharging system of the opposite color; then by triggering U again, we can use the system to recharge the nodes in X once more.

Therefore, we can add a layer of *second-level* recharging systems to recharge all the original (first-level) systems in the graph after all first-level system have been used, as illustrated in Figure 12. Recall that decreasing the sum of balances in a set of nodes by χ requires a recharging system of $O(\sqrt{\chi})$ nodes. We have $\Theta(\sqrt{m})$ first-level systems in our graph, each consisting of $\Theta(\sqrt{m})$ nodes, with a balance of $\Theta(\sqrt{m})$ after use; to reset each node in these systems to their default balance of 1, with $\chi = \Theta(m^{3/2})$, a second-level system requires $\sqrt{\chi} = \Theta(m^{3/4})$ nodes.

In order to keep the overall number of nodes in second-level systems in $O(m)$, we add $\Theta(m^{1/4})$ distinct second-level systems to our graph. When used, each of these second-level systems recharges all systems on the first level, which in turn allows us to propagate through the main relay chain $\Theta(m^{1/2})$ times again. Therefore, with $\Theta(m^{1/4})$ second-level systems in the construction, the first two levels already allow us to traverse the main relay chain $\Theta(m^{1/2}) \cdot \Theta(m^{1/4})$ times.



■ **Figure 12** Connection of a second-level recharging system to first-level recharging systems. For simplicity, only the recharging of group M is shown (node U also has to be recharged).

We can continue this technique in a recursive manner. Assume that we have $\Theta(m^{1/(2^i)})$ distinct i^{th} -level systems in the construction, each consisting of $\Theta(m^{1-1/(2^i)})$ nodes (which, therefore, all have a balance of $\Theta(m^{1-1/(2^i)})$ after they have been used). We can then use an $(i + 1)^{\text{th}}$ -level recharging system to recharge all of these i^{th} -level systems; since we now have $\chi = \Theta(m^{1/(2^i)}) \cdot \Theta(m^{1-1/(2^i)}) \cdot \Theta(m^{1-1/(2^i)}) = \Theta(m^{(2^{i+1}-1)/(2^i)})$, this requires a next level system of $\sqrt{\chi} = \Theta(m^{(2^{i+1}-1)/(2^{i+1})}) = \Theta(m^{1-1/(2^{i+1})})$ nodes. In order to keep the nodes in this new level also in $O(m)$, we only add $\Theta(m^{1/(2^{i+1})})$ systems to the $(i + 1)^{\text{th}}$ -level.

Generally, these higher-level recharging systems fit into our construction in the following way. Every time when first-level systems have all been used, an extra branch is added to the construction, which uses one of the second-level systems to recharge the entire first level (and does not influence the relay chain). Similarly, whenever we would need such a second-level branch but all of them has been used, a third-level branch is added to recharge all second-level systems, and the required second-level branch is only visited after traversing this third-level branch.

Following the recursive pattern, we obtain a construction that allows us to traverse the main relay chain $\Theta(m^{1/2}) \cdot \Theta(m^{1/4}) \cdot \Theta(m^{1/8}) \cdot \dots$ times altogether. If the number of levels go to infinity with m increasing, then for any $\epsilon > 0$, there is an m large enough that the number of relay chain traversals is at least $\Theta(m^{1-\epsilon})$. Since the relay chain consists of $\Theta(m)$ nodes, this leads to a stabilization time of $\Theta(m^{2-\epsilon})$.

If we have $\Theta(m^{1/(2^i)})$ recharging systems on the i^{th} level, this setting allows us to add $\Theta(\log \log m)$ levels until the number of systems on a level decreases to a constant value.

Now let us analyze the number of nodes in the graph. On each level, the systems contain $\Theta(m)$ nodes altogether, so the number of nodes in recharging systems adds up to $\Theta(m \log \log m)$ over all levels. One can easily show that the size of the graph is dominated by these nodes. The number of branches controlling first-level systems is $\Theta(m^{1/2} \cdot m^{1/4} \cdot m^{1/8} \cdot \dots) = O(m)$, the number of branches controlling second-level systems is only $\Theta(m^{1/4} \cdot m^{1/8} \cdot \dots) = O(m^{1/2})$, and so on, the number of i^{th} -level branches is $O(m^{1/2^{i-1}})$. Summing these up, the number of branches altogether is still $O(m)$. Apart from recharging systems, each branch contains constantly many nodes only (in the form of simple relays, AND gates, and the corresponding parts of the fork and join gadgets). This shows that the number of nodes outside of the recharging system is only $O(m)$ altogether, thus the number of nodes in the entire graph is indeed $\Theta(m \log \log m)$.

This allows for a choice of $m = \Theta(\frac{n}{\log \log n})$, leading to a stabilization time of $\Omega(\frac{n^{2-\epsilon}}{(\log \log n)^{2-\epsilon}})$. Since this bound holds for any $\epsilon > 0$, we can easily remove the logarithmic factors: a lower bound of $\Omega(n^{2-\epsilon})$ follows from the same construction for any $\hat{\epsilon} < \epsilon$. Thus the construction shows that the number of steps is $\Omega(n^{2-\epsilon})$.

Similarly to the non-recursive case, this lower bound holds in all of our models, since propagations over the relay chain are still only possible sequentially.

► **Theorem 4.** *For any $\epsilon > 0$, there exists a graph construction with $\Omega(n^{2-\epsilon})$ stabilization time in models B–G.*

References

- 1 Cristina Bazgan, Zsolt Tuza, and Daniel Vanderpooten. Satisfactory graph partition, variants, and generalizations. *European Journal of Operational Research*, 206(2):271–280, 2010.
- 2 Olivier Bodini, Thomas Fernique, and Damien Regnault. Crystallization by stochastic flips. In *Journal of Physics: Conference Series*, volume 226, page 012022. IOP Publishing, 2010.
- 3 Zhigang Cao and Xiaoguang Yang. The fashion game: Network extension of matching pennies. *Theoretical Computer Science*, 540:169–181, 2014.
- 4 Jacques Demongeot, Julio Aracena, Florence Thuderoz, Thierry-Pascal Baum, and Olivier Cohen. Genetic regulation networks: circuits, regulons and attractors. *Comptes Rendus Biologies*, 326(2):171–188, 2003.
- 5 Silvio Frischknecht, Barbara Keller, and Roger Wattenhofer. Convergence in (social) influence networks. In *International Symposium on Distributed Computing*, pages 433–446. Springer, 2013.
- 6 Bernd Gärtner and Ahad N Zehmakan. Color war: Cellular automata with majority-rule. In *International Conference on Language and Automata Theory and Applications*, pages 393–404. Springer, 2017.
- 7 Bernd Gärtner and Ahad N Zehmakan. Majority model on random regular graphs. In *Latin American Symposium on Theoretical Informatics*, pages 572–583. Springer, 2018.
- 8 Eric Goles and Jorge Olivos. Periodic behaviour of generalized threshold functions. *Discrete Mathematics*, 30(2):187–189, 1980.
- 9 Sandra M Hedetniemi, Stephen T Hedetniemi, KE Kennedy, and Alice A Mcrae. Self-stabilizing algorithms for unfriendly partitions into two disjoint dominating sets. *Parallel Processing Letters*, 23(01):1350001, 2013.
- 10 Dominik Kaaser, Frederik Mallmann-Trenn, and Emanuele Natale. Brief Announcement: On the Voting Time of the Deterministic Majority Process. *Distributed*, page 647, 2015.
- 11 Barbara Keller, David Peleg, and Roger Wattenhofer. How Even Tiny Influence Can Have a Big Impact! In *International Conference on Fun with Algorithms*, pages 252–263. Springer, 2014.
- 12 Jeremy Kun, Brian Powers, and Lev Reyzin. Anti-coordination games and stable graph colorings. In *International Symposium on Algorithmic Game Theory*, pages 122–133. Springer, 2013.
- 13 Pál András Papp and Roger Wattenhofer. Stabilization Time in Weighted Minority Processes. In *36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.
- 14 Damien Regnault, Nicolas Schabanel, and Éric Thierry. Progresses in the Analysis of Stochastic 2D Cellular Automata: A Study of Asynchronous 2D Minority. In Luděk Kučera and Antonín Kučera, editors, *Mathematical Foundations of Computer Science 2007*, pages 320–332. Springer Berlin Heidelberg, 2007.
- 15 Damien Regnault, Nicolas Schabanel, and Éric Thierry. On the analysis of “simple” 2d stochastic cellular automata. In *International Conference on Language and Automata Theory and Applications*, pages 452–463. Springer, 2008.

- 16 Jean-Baptiste Rouquier, Damien Regnault, and Éric Thierry. Stochastic minority on graphs. *Theoretical Computer Science*, 412(30):3947–3963, 2011.
- 17 Saharon Shelah and Eric C Milner. Graphs with no unfriendly partitions. *A tribute to Paul Erdős*, pages 373–384, 1990.

A Further discussion of the recursive construction

While the main idea of the recursive construction has been outlined above, there are some details worth discussing for completeness.

Since a second-level system can only be used to recharge nodes of the same color, every time we recharge all the first-level systems, we in fact need two second-level recharging systems, one of each color.

Recall that in addition to the group M , the balance of node U also has to be reset between two uses of a recharging system; however, we did not point this out when calculating the necessary size of systems, since besides M , a single extra node does not affect the magnitude. Earlier, we have noted that the recharging systems on a certain level consist of two classes of systems of different color; observe that the next level systems that recharge the groups M in one class can simultaneously be used to also recharge the nodes U in the other class. Alternatively (for simpler analysis), we can add an extra recharging system (of the same size) on each branch in order to separately recharge the nodes U on the level below.

Note that the number of total relay chain traversals, which is $\Theta(m^{1/2} \cdot m^{1/4} \cdot m^{1/8} \cdot \dots)$, is in fact only guaranteed to be at least $\Theta(m^{1-\epsilon})$ if the coefficients in these factors are sufficiently large. With an analysis of the constants in the process, one can show that the coefficient in each factor can indeed be chosen as 1, and thus these constant do not add up to dividing logarithmic factors when taking the product. However, this is in fact unnecessary, as any such logarithmic factor could also be removed simply by a smaller choice of ϵ .

Finally, note that in this recursive setting, recharging systems are slightly modified in the sense that they have multiple input nodes from multiple different branches, each connected to node U . However, this does not modify the behavior of U as long as its initial balance is readjusted to 1. This also requires a minor modification in the simple relays that are used as input nodes, since relays generally assume that their output node never switches before the relays themselves are triggered. This can be resolved by using a modified relay where the base node has an initial balance of 3, and thus it is enabled by two distinct simple relays on the branch.

B Detailed analysis of gadgets

Here we provide a more detailed description of the gadgets, and also comment on their behavior and their use in the construction.

Simple relay. The construction and behavior of the simple relay has already been described above. One thing to note is that in our construction, simple relays are always used only once: after node B switches, propagation never returns to the same part of the graph again, and thus node B will remain unswitchable for the rest of the process.

While we mostly use this original version of the gadget, we occasionally need relays with multiple output nodes instead of just one. This only requires a simple modification: besides connecting x extra (black) output nodes to node B , we also need to add x fixed (white) nodes in order to keep the initial balance of B unchanged.

43:16 Stabilization Time in Minority Processes

Chains of simple relays are mostly used to connect more complex gadgets in our construction. Note that depending on whether the input and output nodes in these gadgets are supposed to have the same or different initial colors, we only need a chain of length 1 or 2 for this, respectively.

Rechargeable relay. In a rechargeable relay, node B is connected to an upper node U instead of a fixed node. Node U is connected to a group C of two nodes, which is further connected to nodes R_1, R_2 . Initially, C has the opposite color as B and U , and one of R_1 and R_2 is white, the other is black. Node B has the same external neighbors as a simple relay. The recharge nodes can both have any set of external neighbors as long as their initial balance is 3 with C ignored (so with C included, the initial balance of R_1 and R_2 is then 1 and 5, respectively).

Note that since R_1 and R_2 have opposite colors, this recharging process can always be executed on a used relay through either R_1 or R_2 , depending on the current color of the nodes. We only need to select the recharge node that has the current color of U , and switch 3 of its neighbors (to U 's current color) for the recharging step, and then switch 3 of its neighbors (to the opposite color) for the resetting step.

Recharging system. In a basic recharging system, the node U is connected to the input node v , the group M , and to $\sqrt{n} + 1$ fixed white nodes. The middle level group M has a further edge to all nodes L_i , and is balanced by \sqrt{m} fixed black nodes. Finally, each node L_i has \sqrt{m} distinct neighbors in X , and thus each node in X is connected to exactly one lower-level node. For convenience, we assume that m is a square number.

A generalized recharging system is almost identical to this, except for the nodes L_i occasionally being connected to the same node. The connections between the lower level and X are not directly specified: we are free to choose which of the nodes L_i to connect to a specific node in X . Note, however, that the gadget design implicitly assumes that $x_j \leq \sqrt{\chi}$ for all nodes in X . This is naturally satisfied whenever we use the gadget in our constructions, since we always have $x_1 = x_2 = \dots = x_m$ with $|X| > x_j$. Also note that for convenience, we assume χ to be a square number.

Nodes in the upper and lower levels are initially white, while M and the input node v are initially black. The nodes X may assume any color, and also may switch multiple times before the recharging system is activated. However, the graph construction ensures that at the time when the gadget is activated (that is, when v switches), all nodes in X are currently colored black (i.e., we indeed use the system on rechargeable relays that can currently be recharged). The gadget design ensures that U and M have an initial balance of 1, while the nodes L_i have a balance of 1 at least, depending on the current color of their neighbors in X .

AND gate. The AND gate consist of 7 nodes. The input nodes of X are connected to node A , which is further connected to all other nodes in the gadget (B_1, B_2, D and the group C on 3 nodes). Nodes B_1 and B_2 are also connected to group C , node B_1 has an edge to node D , and node D is connected to some external black node v on the output side. Furthermore, A, B_1 and B_2 have $x + 1, 4$ and 3 fixed black neighbors, respectively.

One can check that each node has a positive balance as long as there exists a black node in X . Node A gets a balance of $x - 1$ from the nodes within the gadget, so it is not switchable unless all nodes in X are white. Nodes B_1, B_2, C and D all have an initial balance of 1.

After the gadget reaches its final stage (see Figure 8), no node in the gadget can ever change again, regardless of the states of X or v .

Note that for the described behavior of the gadget, we also need the fact that none of the nodes in X switch between the first and second switching of A . The switching of A only increases their balance (temporarily), so this is guaranteed if other neighbors of nodes in X do not interfere with the process. In the construction, we only use AND gates this way: whenever a node A becomes switchable in a gate, then that is the only switchable node in the entire graph, so no other nodes will switch until the propagation reaches v .

As long as this condition is fulfilled, we can connect any number of AND gates to a given node of the graph without affecting its behavior; we only have to make sure that we also add fixed node neighbors to restore the node's balance to the original value.

Join gadget. The join gadget consists of a central node C , and of p distinct 2-group *starter gadgets* of alternating color (we assume p to be even). Each starter gadget consists of two groups A_i and B_i , both of size 2 (with $i \in \{1, 2, \dots, p\}$). The two groups are connected to each other, and A_i has a further edge to the input node v_i , and two fixed nodes of the same color as its own. Finally, all B_i are connected to a central node C , which is in turn connected to an output node v . Node C also has two further fixed black connections.

Initially, A_i for odd i values, B_i for even i values, v_i for even i and node C are colored white; the remaining nodes are colored black. Nodes A_i have an initial balance of 1, nodes B_i have an initial balance of 1 or 3 (depending on parity), and C has an initial balance of 3.

Every time after v switches, the balance of C returns to its initial value of 3, so the switching of the next input node will trigger the same process through the next starter gadget.

Fork gadget. The fork gadget consists of q nodes F_1, \dots, F_q of alternating color, where we assume q to be an odd number. All F_i are connected to the same input node v , and each to a distinct output node v_i . They are also linked to each other, with F_i connected to F_{i+1} for all $i \in \{1, 2, \dots, q-1\}$. Also, node F_1 and F_q have a fixed neighbor colored black and white, respectively (imitating the role of the nonexistent nodes F_0 and F_{q+1}). Finally, each F_i has a further fixed neighbor of its original color. Initially, F_i is colored black for odd i and white for even i values.

The balance of F_1 and all white F_i -s is originally 1 in this setting, while the balance of black F_i -s (except for F_1) is 3. Hence when v first switches, only F_1 will become switchable (and switching it will propagate on through v_1). The next time v switches, it switches back to white; with v and F_1 both white, F_2 can now switch too. The pattern continues all the way to F_q : as F_{i-1} has already been switched before, as soon as v switches back to the color of F_i , F_i becomes switchable, too, enabling propagation on the next branch. After v_i switches (and remains that way), F_i is not switchable anymore, since v_i , F_{i-1} and its fixed neighbor all have the opposite color.

Note that since each switching F_i increases the current balance of v from 1 to 3, we need to switch two neighbors of v in each turn to make v switchable again. This is exactly what happens when v is the base node of the rightmost relay in the chain: between every consecutive switches of v , we switch both node U (by the recharging step) and node v_L (by propagation through the chain) in the relay, and thus v becomes switchable again.

Note that since it is connected to the fork gadget, the rightmost rechargeable relay in the chain is a modified one in the sense that its base node has not one, but q right-side neighbors, colored in alternating fashion. However, this fact does not change its behavior at all. The initial balance of the base node is still 1, and every time after v switches, it has one of its neighbors F_i switching in the opposite direction. That has exactly the same effect as if the right neighbor was simply a subsequent relay in the chain, triggered by v .

On the whole construction. For convenience, we assume in the construction that both m and r are even numbers.

Recharging systems and AND gates, as all other gadgets, are available in two color variants; in the overview of the construction, we did not discuss which variant is used in which case. However, the current state of each relay in each round is straightforward to calculate, so the necessary color of all recharging systems and AND gates can easily be determined.

Also, we have seen that AND gates are used to ensure that the given recharging or resetting operations have completely been executed. In order to achieve this, in case of the first systems (which recharge relays), the input edges of the gates can be connected to the upper nodes of the corresponding relays, since that is the last node to switch in the sequence. In case of the systems that reset relays, the aim is only to switch the corresponding recharge node of the relay, so we can connect the gates to the recharge nodes.

However, as each AND gate belongs to a certain branch of the construction, we also have to ensure that the AND gate is only activated when the propagation reaches this branch, and stays inactive as long as previous branches are being processed. Therefore, besides the specified nodes in the relays, the final input node of the AND gate is the node which was used to enable the recharging system in question (node v of Figure 5). This way, the gates ensure that *after* the recharging system is activated, propagation only continues if all the resulting switches were executed.

Generalization to $\omega(1)$ colors

One can observe that in the construction of Section 5, except for nodes A in the AND gates, all nodes in the graph have a degree of $O(\sqrt{n})$. We can slightly modify the construction and replace each of these AND gates with two levels of such gates, with $\Theta(\sqrt{n})$ distinct gates on the first level (each with $\Theta(\sqrt{n})$ input nodes), and a final gate that connects the outputs of these first-level gates. This gives us a construction with the same properties, but a maximum degree of $O(\sqrt{n})$.

This allows us to generalize the lower bound of $\Omega(n^{\frac{3}{2}})$ to the case of not only $O(1)$, but up to $O(\sqrt{n})$ colors. The technique for this is the same as in the case of $O(1)$ colors: we add a multipartite graph colored with the additional colors, and connect each of its nodes to each original node. With $\Delta = O(\sqrt{n})$ established, it suffices to have $\Theta(\sqrt{n})$ nodes in each of the color classes. Therefore, using only $\Theta(n)$ additional nodes, we can extend the graph by a multipartite graph on $\Theta(\sqrt{n})$ color classes, each consisting of only $\Theta(\sqrt{n})$ nodes.

C Notes on simulations

Due to its complexity, we have also verified the correctness of the non-recursive construction of Section 5 through implementing it and running a simulation of the minority process. Note that in general, it is difficult to simulate a minority process in a benevolent model, since all possible switching sequences would have to be examined to find the one with the smallest number of steps.

Fortunately, the task is significantly simpler in our case, due to the properties of the construction. The key observation in our graph is that whenever propagation is split into multiple parallel threads (that is, when there are multiple switchable nodes at the same time), then propagation on any of these threads does not influence propagation on other threads at all. Specifically, the nodes on separate threads do not have common neighbors except for the beginning and end of such threads; i.e. when a switching node splits the propagation to multiple threads, or when threads are joined in an AND-like fashion, meaning

that a common neighbor only becomes switchable when propagation has been finished in all of the threads. This implies that throughout the process, these threads can be handled completely independently from each other, and the order in which they are processed is irrelevant. Note that this is also the property of the construction which ensures that the set of switchable nodes is an independent set in any state.

If we exploit this property, the process can be simulated easily by always choosing an arbitrary one of the switchable nodes in the graph, knowing that the choice of nodes will not influence the outcome. To verify correctness in such a simulation, we only have to check that in each step of the process, the set of nodes that become switchable is exactly the set of nodes determined by the analysis. Note that the opposite does not happen in our construction: the switching of a node never makes another switchable node unswitchable (this would also contradict the property that switchable nodes form an independent step in any state).

When examining concrete instances of our construction, we used the parameter r as the input to determine the size of the instance. For a given input value of r (always an even number), we have chosen $m = 2 \cdot (r - 1)^2$, which fits our preconditions on both magnitudes and parity. All other details of the construction are already determined above; the only additional thing to note is that whenever different gadgets are connected through a chain of simple relays, we always use the smallest possible such chain in the implementation.

The simulations verified that the analysis of the construction is correct, and thus stabilization time is indeed $\Omega(n^{3/2})$ in model B. Table 1 illustrates the number of steps for some choices r , along with the resulting number of nodes in the construction. One can observe that the number of steps indeed grows superlinearly in n .

■ **Table 1** Number of steps on some specific graphs.

Input (r)	Nodes (n)	Steps
2	99	112
4	469	772
8	1 929	5 884
16	7 729	47 404
24	17 369	161 372
30	27 119	316 568
40	48 169	754 108
60	108 269	2 559 188
80	192 369	6 084 268
100	300 469	11 905 348
120	432 569	20 598 428

Parameterized Complexity of Stable Roommates with Ties and Incomplete Lists Through the Lens of Graph Parameters

Robert Brederbeck 

Technische Universität Berlin, Chair of Algorithmics and Computational Complexity, Germany
robert.bredereck@tu-berlin.de

Klaus Heeger 

Technische Universität Berlin, Chair of Algorithmics and Computational Complexity, Germany
heeger@tu-berlin.de

Dušan Knop 

Technische Universität Berlin, Chair of Algorithmics and Computational Complexity, Germany
Department of Theoretical Computer Science, Faculty of Information Technology,
Czech Technical University in Prague, Prague, Czech Republic
dusan.knop@fit.cvut.cz

Rolf Niedermeier 

Technische Universität Berlin, Chair of Algorithmics and Computational Complexity, Germany
rolf.niedermeier@tu-berlin.de

Abstract

We continue and extend previous work on the parameterized complexity analysis of the NP-hard STABLE ROOMMATES WITH TIES AND INCOMPLETE LISTS problem, thereby strengthening earlier results both on the side of parameterized hardness as well as on the side of fixed-parameter tractability. Other than for its famous sister problem STABLE MARRIAGE which focuses on a bipartite scenario, STABLE ROOMMATES WITH INCOMPLETE LISTS allows for arbitrary acceptability graphs whose edges specify the possible matchings of each two agents (agents are represented by graph vertices). Herein, incomplete lists and ties reflect the fact that in realistic application scenarios the agents cannot bring *all* other agents into a *linear* order. Among our main contributions is to show that it is $W[1]$ -hard to compute a maximum-cardinality stable matching for acceptability graphs of bounded treedepth, bounded tree-cut width, and bounded feedback vertex number (these are each time the respective parameters). However, if we “only” ask for perfect stable matchings or the mere existence of a stable matching, then we obtain fixed-parameter tractability with respect to tree-cut width but not with respect to treedepth. On the positive side, we also provide fixed-parameter tractability results for the parameter feedback edge set number.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Graph algorithms analysis; Mathematics of computing → Matchings and factors

Keywords and phrases Stable matching, acceptability graph, fixed-parameter tractability, $W[1]$ -hardness, treewidth, treedepth, tree-cut width, feedback set numbers

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.44

Related Version A full version of this paper can be found at <http://arxiv.org/abs/1911.09379>.

Funding Main work was done while all authors were with TU Berlin.

Klaus Heeger: Supported by DFG Research Training Group 2434 “Facets of Complexity”.

Dušan Knop: Supported by the DFG, project MaMu (NI 369/19).



© Robert Brederbeck, Klaus Heeger, Dušan Knop, and Rolf Niedermeier;
licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 44; pp. 44:1–44:14

Leibniz International Proceedings in Informatics



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

1 Introduction

The computation of stable matchings is a core topic in the intersection of algorithm design and theory, algorithmic game theory, and computational social choice. It has numerous applications – the research goes back to the 1960s. The classic (and most prominent from introductory textbooks) problem STABLE MARRIAGE, which is known to be solvable in linear time, relies on complete bipartite graphs for the modeling with the two sides representing the same number of “men” and “women”. Herein, each side expresses preferences (linear orderings aka rankings) over the opposite sex. Informally, stability then means that no two matched agents have reason to break up. STABLE ROOMMATES, however, is not restricted to a bipartite setting: given is a set V of agents together with a preference list \mathcal{P}_v for every agent $v \in V$, where a preference list \mathcal{P}_v is a strict (linear) order on $V \setminus \{v\}$. The task is to find a *stable matching*, that is, a set of pairs of agents such that each agent is contained in at most one pair and there is no blocking edge (i.e., a pair of agents who strictly prefer their mates in this pair to their partners in the matching; naturally, we assume that agents prefer to be matched over being unmatched). Such a matching can be computed in polynomial time [18]. We refer the reader to the monographs [16, 25] for a general discussion on STABLE ROOMMATES. Recent practical applications of STABLE ROOMMATES and its variations also to be studied here range from kidney exchange to connections in peer-to-peer networks [10, 33, 34].

If the preference lists \mathcal{P}_v for all agents v are complete, then the graph-theoretic model behind is trivial – a complete graph reflects that every agent ranks all other agents. In the more realistic scenario that an agent may only rank part of all other agents, the corresponding graph, referred to as *acceptability graph*, is no longer a complete graph but can have an arbitrary structure. We assume that the acceptability is symmetric, that is, if an agent v finds an agent u acceptable, then also agent u finds v acceptable. Moreover, to make the modeling of real-world scenarios more flexible and realistic, one also allows ties in the preference lists (rankings) of the agents, meaning that tied agents are considered equally good. Unfortunately, once allowing ties in the preferences, STABLE ROOMMATES already becomes NP-hard [24, 32], indeed this is true even if each agent finds at most three other agents acceptable [7]. Hence, in recent works specific (parameterized) complexity aspects of STABLE ROOMMATES WITH TIES AND INCOMPLETE LISTS (SRTI) have been investigated [1, 3, 5]. In particular, while Brederick et al. [3] studied restrictions on the structure of the preference lists, Adil et al. [1] initiated the study of structural restrictions of the underlying acceptability graph, including the parameter treewidth of the acceptability graph. We continue Adil et al.’s line of research by systematically studying three variants (“maximum”, “perfect”, “existence”) and by extending significantly the range of graph parameters under study, thus gaining a fairly comprehensive picture of the parameterized complexity landscape of STABLE ROOMMATES WITH TIES AND INCOMPLETE LISTS.

Notably, while previous work [1, 15] argued for the (also practical) relevance for studying the structural parameters treewidth and vertex cover number, our work extends this to further structural parameters that are either stronger than vertex cover number or yield more positive algorithmic results than possible for treewidth. We study the arguably most natural optimization version of STABLE ROOMMATES with ties and incomplete lists, referred to as MAX-SRTI:

MAX-SRTI

Input: A set V of agents and a profile $\mathcal{P} = (\mathcal{P}_v)_{v \in V}$.

Task: Find a maximum-cardinality stable matching or decide that none exists.

In addition to MAX-SRTI, we also study two NP-hard variants. The input is the same, but the task either changes to finding a *perfect* stable matching – this is PERFECT-SRTI – or to finding just *any* stable matching – this is SRTI-EXISTENCE.¹

PERFECT-SRTI

Input: A set of agents V and a profile $\mathcal{P} = (\mathcal{P}_v)_{v \in V}$.

Task: Find a perfect stable matching or decide that none exists.

SRTI-EXISTENCE

Input: A set of agents V and a profile $\mathcal{P} = (\mathcal{P}_v)_{v \in V}$.

Task: Find a stable matching or decide that none exists.

Related Work. On bipartite acceptability graphs, where STABLE ROOMATES is called STABLE MARRIAGE, MAX-SRTI admits a polynomial-time factor- $\frac{2}{3}$ -approximation [29]. However, even on bipartite graphs it is NP-hard to approximate MAX-SRTI by a factor of $\frac{29}{33}$, and MAX-SRTI cannot be approximated by a factor of $\frac{3}{4} + \epsilon$ for any $\epsilon > 0$ unless VERTEX COVER can be approximated by a factor strictly smaller than two [36]. Note that, as we will show in our work, SRTI-EXISTENCE is computationally hard in many cases, so good polynomial-time or even fixed-parameter approximation algorithms for MAX-SRTI seem out of reach.

PERFECT-SRTI was shown to be NP-hard even on bipartite graphs [19]. This holds also for the more restrictive case when ties occur only on one side of the bipartition, and any preference list is either strictly ordered or a tie of length two [24]. As SRTI-EXISTENCE is NP-hard for complete graphs, all three problems considered in this paper are NP-hard on complete graphs (as every stable matching is a maximal matching). This implies paraNP-hardness for all parameters which are constant on cliques, including distance to clique, cliquewidth, neighborhood diversity, the number of uncovered vertices, and modular width.

Following up on work by Bartholdi III and Trick [2], Brederbeck et al. [3] showed NP-hardness and polynomial-time solvability results for SRTI-EXISTENCE under several restrictions constraining the agents' preference lists.

On a fairly general level, there is quite some work on employing methods of parameterized algorithmics in the search for potential islands of tractability for in general NP-hard stable matching problems [1, 5, 6, 26, 28]. More specifically, Marx and Schlotter [26] showed that MAX-SRTI is W[1]-hard when parameterized by the number of ties. They observed that it is NP-hard even if the maximum length of a tie is constant but showed that MAX-SRTI is fixed-parameter tractable when parameterized by the combined parameter “number of ties and maximum length of a tie”. Meeks and Rastegari [30] considered a setting where the agents are partitioned into different types having the same preferences. They show that the problem is FPT in the number of types. Mnich and Schlotter [31] defined STABLE MARRIAGE WITH COVERING CONSTRAINTS, where the task is to find a matching which matches a given set of agents, and minimizes the number of blocking pairs among all these matchings. They showed the NP-hardness of this problem and investigated several parameters such as the number of blocking pairs or the maximum degree of the acceptability graph.

¹ In the following, we consider a slightly different formulation of these problems: We assume that the input consists of the acceptability graph and rank functions. This is no restriction, as one can transform a set of agents and a profile to an acceptability graph and rank functions and vice versa in linear time.

Most importantly for our work, however, Adil et al. [1] started the research on structural restrictions of the acceptability graph, which we continue and extend. Their result is an XP-algorithm for the parameter treewidth; indeed, they did not show $W[1]$ -hardness for this parameter, leaving this as an open question. This open question was solved (also for the bipartite case) by Gupta et al. [15], who further considered various variants (such as sex-equal or balanced) of stable marriage with respect to two variants of treewidth.² Moreover, Adil et al. [1] showed that MAX-SRTI is fixed-parameter tractable when parameterized by the size of the solution (that is, the cardinality of the set of edges in the stable matching) and that MAX-SRTI restricted to planar acceptability graphs then is fixed-parameter tractable even with subexponential running time.³

Our Contributions. We continue the study of algorithms for MAX-SRTI and its variants based on structural limitations of the acceptability graph. In particular, we extend the results of Adil et al. [1] in several ways. For an overview on our results we refer to Figure 1. We highlight a few results in what follows. We observe that Adil et al.’s dynamic programming-based XP-algorithm designed for the parameter treewidth⁴ indeed yields fixed-parameter tractability for the combined parameter treewidth and maximum degree. We complement their XP result and the above mentioned results by showing that MAX-SRTI is $W[1]$ -hard for the graph parameters treedepth, tree-cut width, and feedback vertex set. Notably, all these graph parameters are “weaker” [22] than treewidth and these mutually independent results imply $W[1]$ -hardness with respect to treewidth; the latter was also shown in the independent work of Gupta et al. [15].

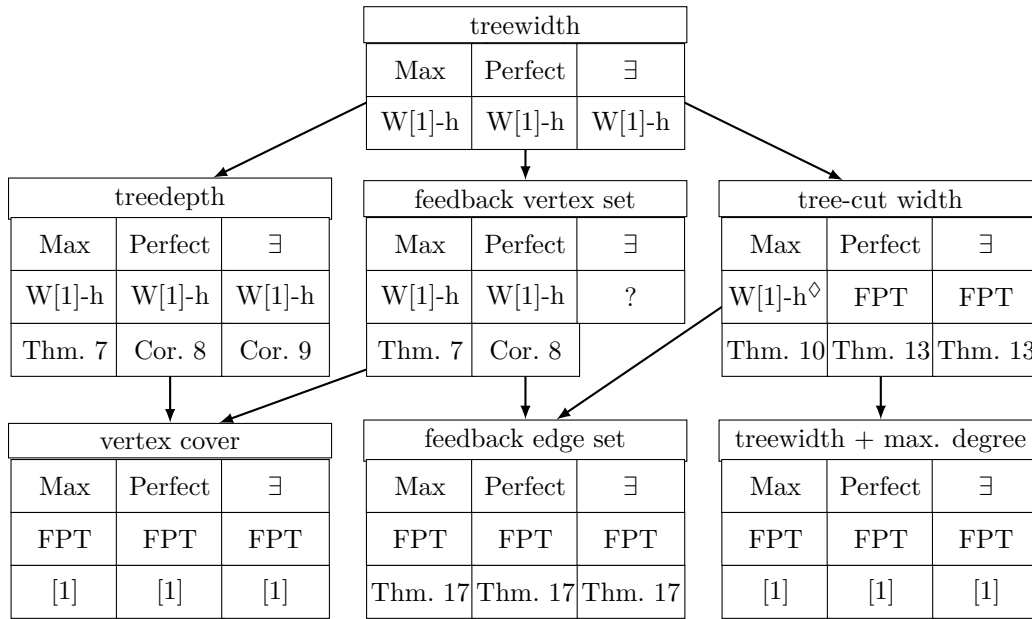
For the two related problems PERFECT-SRTI and SRTI-EXISTENCE, on the contrary we show fixed-parameter tractability with respect to the parameter tree-cut width. These results confirm the intuition that tree-cut width, a recently introduced [35] and since then already well researched graph parameter [11, 12, 13, 20, 27] “lying between” treewidth and the combined parameter “treewidth and maximum vertex degree”, is a better suited structural parameter for edge-oriented problems than treewidth is. Moreover, we extend our $W[1]$ -hardness results to PERFECT-SRTI and SRTI-EXISTENCE parameterized by treedepth and to PERFECT-SRTI parameterized by the feedback vertex number.

In summary, we provide a fairly complete picture of the (graph-)parameterized computational complexity landscape for the three studied problems – see Figure 1 for an overview of our results. Among other things Figure 1 for the parameter tree-cut width depicts a surprising complexity gap between MAX-SRTI on the one side ($W[1]$ -hardness) and PERFECT-SRTI and SRTI-EXISTENCE (fixed-parameter tractability) on the other side. Finally, Figure 1 leaves as an open question the parameterized complexity of SRTI-EXISTENCE with respect to the parameter feedback vertex set number which we conjecture to be answered with $W[1]$ -hardness.

² Indeed, without knowing the work of Gupta et al. [15] our work initially was strongly motivated by Adil et al.’s [1] open question for treewidth. To our surprise, although the Adil et al. [1] paper has been revised six months after the publication of Gupta et al. [15], it was not mentioned by Adil et al. [1] that this open question was answered by a subset of the authors, namely Gupta et al. [15].

³ More precisely, Adil et al. state their result for the parameter “size of a maximum matching of the acceptability graph”, which is only by a factor at most two greater than the size of a stable matching.

⁴ It only gives containment in XP for this parameter, and only this is stated by Adil et al. [1].



■ **Figure 1** Results for graph-structural parameterizations of STABLE ROOMMATES WITH TIES AND INCOMPLETE LISTS. Max means MAX-SRTI, Perfect means PERFECT-SRTI, and ∃ means SRTI-EXISTENCE. The symbol [◇] indicates the existence of an FPT factor- $\frac{1}{2}$ -approximation algorithm (see Corollary 16). The arrows indicate dependencies between the different parameters. An arrow from a parameter p_1 to a parameter p_2 means that there is a computable function $f: \mathbb{N} \rightarrow \mathbb{N}$ such that for any graph G we have $p_1(G) \leq f(p_2(G))$. Consequently, fixed-parameter tractability for p_1 then implies fixed-parameter tractability for p_2 , and W[1]-hardness for p_2 then implies W[1]-hardness for p_1 .

2 Preliminaries

For a positive integer n let $[n] := \{1, 2, 3, \dots, n\} = \{x \in \mathbb{N} : x \leq n\}$. We write vectors \mathbf{h} in boldface, and access their entries (coordinates) via $\mathbf{h}(e)$.

For a graph G and a vertex $v \in V(G)$, let $\delta_G(v)$ be the set of edges incident to v . If the graph G is clear from the context, then we may just write $\delta(v)$. For a subset of edges $M \subseteq E(G)$ and a vertex $v \in V(G)$, we define $\delta_M(v) := \delta_G(v) \cap M$. We denote the maximum degree in G by $\Delta(G)$, i.e., $\Delta(G) := \max_{v \in V(G)} |\delta_G(v)|$. For a tree T rooted at a vertex r and a vertex $v \in V(T)$, we denote by T_v the subtree rooted at v . For a graph G and a subset of vertices X (a subset of edges F), we define $G - X$ ($G - F$) to be the graph arising from G by deleting all vertices in X and all edges incident to a vertex from X (deleting all edges in F). For a graph G and a set of vertices $X \subseteq V(G)$, the graph arising by *contracting* X is denoted by $G_{/X}$; it is defined by replacing the vertices in X by a single vertex. Thus, we have $V(G_{/X}) := (V(G) \setminus X) \cup \{v_X\}$ and $E(G_{/X}) := \{\{v, w\} \in E(G) : v, w \notin X\} \cup \{\{v, v_X\} : \{v, x\} \in E(G) : v \notin X, x \in X\}$. Unless stated otherwise, $n := |V(G)|$, and $m := |E(G)|$.

We define the directed graph \overleftrightarrow{G} by replacing each edge $\{v, w\} \in E(G)$ by two directed ones in opposite directions, i.e. (v, w) and (w, v) .

Note that the acceptability graph for a set of agents V and a profile \mathcal{P} is always simple, while a graph arising from a simple graph through the contraction of vertices does not need to be simple.

A parameterized problem consists of the problem instance I (in our setting the STABLE ROOMMATE instance) and a parameter value k (in our case always a number measuring some aspect in acceptability graph). An FPT-algorithm for a parameterized problem is an algorithm that runs in time $f(k)|I|^{O(1)}$, where f is some computable function. That is, an FPT algorithm can run in exponential time, provided that the exponential part of the running time depends on the parameter value only. If such an algorithm exists, the parameterized problem is called fixed-parameter tractable for the corresponding parameter. There is also a theory of hardness of parameterized problems that includes the notion of W[1]-hardness. If a problem is W[1]-hard for a given parameter, then it is widely believed not to be fixed-parameter tractable for the same parameter.

The typical approach to showing that a certain parameterized problem is W[1]-hard is to reduce to it a known W[1]-hard problem, using the notion of a parameterized reduction. In our case, instead of using the full power of parameterized reductions, we use standard many-one reductions that ensure that the value of the parameter in the output instance is bounded by a function of the parameter of the input instance.

The Exponential-Time Hypothesis (ETH) of Impagliazzo and Paturi [17] asserts that there is a constant $c > 1$ such that there is no $c^{o(n)}$ time algorithm solving the SATISFIABILITY problem, where n is the number of variables. Chen et al. [4] showed that assuming ETH, there is no $f(k) \cdot n^{o(k)}$ time algorithm solving k -(MULTICOLORED) CLIQUE, where f is any computable function and k is the size of the clique we are looking for. For further notions related to parameterized complexity and ETH refer to [8].

2.1 Profiles and preferences

Let V be a set of agents. A *preference list* \mathcal{P}_v for an agent v is a subset $P_v \subseteq V \setminus \{v\}$ together with an ordered partition $(P_v^1, P_v^2, \dots, P_v^k)$ of P_v . A set P_v^i with $|P_v^i| > 1$ is called a *tie*. The *size of a tie* P_v^i is its cardinality, i.e., $|P_v^i|$. For an agent $v \in V$, the *rank function* is $\text{rk}_v: P_v \cup \{v\} \rightarrow \mathbb{N} \cup \{\infty\}$ with $\text{rk}_v(x) := i$ for $x \in P_v^i$, and $\text{rk}_v(v) = \infty$.

We say that v *prefers* $x \in P_v$ *over* $y \in P_v$ if $\text{rk}_v(x) < \text{rk}_v(y)$. If $\text{rk}_v(x) = \text{rk}_v(y)$, then v *ties* x and y . For a set V of agents, a set $\mathcal{P} = (\mathcal{P}_v)_{v \in V}$ of preference lists is called a *profile*. The corresponding *acceptability graph* G consists of vertex set $V(G) := V$ and edge set $E(G) := \{\{v, w\} : v \in P_w \wedge w \in P_v\}$.

A subset $M \subseteq E(G)$ of pairwise non-intersecting edges is called a matching. If $\{x, y\} \in M$, then we denote the corresponding partner y of x by $M(x)$ and set $M(x) := x$ if x is unmatched, that is, if $\{y \in V(G) : \{x, y\} \in M\} = \emptyset$. An edge $\{v, w\} \in E(G)$ is *blocking for* M if $\text{rk}_v(w) < \text{rk}_v(M(v))$ and $\text{rk}_w(v) < \text{rk}_w(M(w))$; we say that v, w constitutes a *blocking pair for* M . A matching $M \subseteq E(G)$ is *stable* if there are no blocking pairs, i.e., for all $\{v, w\} \in E(G)$, we have $\text{rk}_v(w) \geq \text{rk}_v(M(v))$ or $\text{rk}_w(v) \geq \text{rk}_w(M(w))$.

Note that the literature contains several different stability notions for a matching in the presence of ties. Our stability definition is frequently called *weak stability*.⁵

2.2 Structural graph parameters

We consider the (graph-theoretic) parameters treewidth, tree-cut width, treedepth, feedback vertex number, feedback edge number, vertex cover number, and the combined parameter “treewidth + maximum vertex degree” (also called degree-treedepth in the literature).

⁵ Manlove [23] discusses other types of stability – strong stability and super-strong stability.

A set of vertices $S \subseteq V(G)$ is a *feedback vertex set* if $G - S$ is a forest and the *feedback edge set* is a subset $F \subseteq E(G)$ of edges such that $G - F$ is a forest. We define the *feedback vertex (edge) number* $\text{fvs}(G)$ ($\text{fes}(G)$) to be the cardinality of a minimum feedback vertex (edge) set of G . A vertex cover is a set of vertices intersecting with every edge of G , and the vertex cover number $\text{vc}(G)$ is the size of a minimum vertex cover. The *treedepth* $\text{td}(G)$ is the smallest height of a rooted tree T with vertex set $V(G)$ such that for each $\{v, w\} \in V(G)$ we have that either v is a descendant of w in T or w is a descendant of v in T .

Treewidth intuitively measures the tree-likeness of a graph. It can be defined via structural decompositions of a graph into pieces of bounded size, which are connected in a tree-like fashion, called *tree decompositions*.

Tree-Cut Width. Tree-cut width has been introduced by Wollan [35] as tree-likeness measure between treewidth and treewidth combined with maximum degree. A family of subsets X_1, \dots, X_k of a finite set X is a *near-partition* of X if $X_i \cap X_j = \emptyset$ for all $i \neq j$ and $\bigcup_{i=1}^k X_i = X$. Note that $X_i = \emptyset$ is possible (even for several distinct i). A *tree-cut decomposition* of a graph G is a pair (T, \mathcal{X}) which consists of a tree T and a near-partition $\mathcal{X} = \{X_t \subseteq V(G) : t \in V(T)\}$ of $V(G)$. A set in the family \mathcal{X} is called a *bag* of the tree-cut decomposition. Given a tree node t , let T_t be the subtree of T rooted at t . For a node $t \in V(T)$, we denote by Y_t the set of vertices induced by T_t , i.e. $Y_t := \bigcup_{v \in V(T_t)} X_v$.

For an edge $e = \{u, v\} \in E(T)$, we denote by $T_u^{\{u,v\}}$ and $T_v^{\{u,v\}}$ the two connected components in $T - e$ which contain u respectively v . These define a partition

$$\left(\bigcup_{t \in T_u^{\{u,v\}}} X_t, \bigcup_{t \in T_v^{\{u,v\}}} X_t \right)$$

of $V(G)$. We denote by $\text{cut}(e) \subseteq E(G)$ the set of edges of G with one endpoint in $\bigcup_{t \in T_u^{\{u,v\}}} X_t$ and the other one in $\bigcup_{t \in T_v^{\{u,v\}}} X_t$.

A tree-cut decomposition is called *rooted* if one of its nodes is called the root r . For any node $t \in V(T) \setminus \{r\}$, we denote by $e(t)$ the unique edge incident to t on the r - t -path in T . The *adhesion* $\text{adh}_T(t)$ is defined as $|\text{cut}(e(t))|$ for each $t \neq r$, and $\text{adh}_T(r) := 0$.

The *torso of a tree-cut decomposition* (T, \mathcal{X}) at a node t , denoted by H_t , can be constructed from G as follows: If T consists of a single node, then the torso of $t \in V(T)$ is G . Else let C_t^1, \dots, C_t^ℓ be the connected components of $T - t$. Let $Z_i := \bigcup_{v \in V(C_t^i)} X_v$. Then, the torso arises from G by contracting each $Z_i \subseteq V(G)$ for $1 \leq i \leq \ell$.

The operation of *suppressing a vertex* v of degree at most two consists of deleting v and, if v has degree exactly two, then adding an edge between the two neighbors of v . The torso-size $\text{tor}(t)$ is defined as the number of vertices of the graph arising from the torso H_t by exhaustively suppressing all vertices of degree at most two.

The *width of a tree-cut decomposition* (T, \mathcal{X}) is defined as $\max_{t \in V(T)} \{\text{adh}(t), \text{tor}(t)\}$. The *tree-cut width* $\text{tcw}(G)$ of a graph G is the minimum width of a tree-cut decomposition of G .

Nice tree-cut decompositions. Similarly to nice tree decompositions [21], each tree-cut decomposition can be transformed into a nice tree-cut decomposition. Nice tree-cut decompositions have additional properties which help simplifying algorithm design. Besides the definition of nice tree-cut decompositions, in the following we provide some of its properties.⁶

⁶ The properties used here are stated (without a proof) by Ganian et al. [11].

► **Definition 1** ([11]). Let (T, \mathcal{X}) be a tree-cut decomposition. A node $t \in V(T)$ is called light if $\text{adh}(t) \leq 2$ and all outgoing edges from Y_t end in X_p , where p is the parent of t , and heavy otherwise (see Figure 2 for an example).

► **Theorem 2** ([11, Theorem 2]). Let G be a graph with $\text{tcw}(G) = k$. Given a tree-cut decomposition of G of width k , one can compute a nice tree-cut decomposition (T, \mathcal{X}) of G of width k with at most $2|V(G)|$ nodes in cubic time.

► **Lemma 3** ([11, Lemma 2]). Each node t in a nice tree-cut decomposition of width k has at most $2k + 1$ heavy children.

In what follows, we will assume that a nice tree-cut decomposition of the input graph is given. Computing the tree-cut width of a graph is NP-hard [20], but there exists an algorithm, given a graph G and an integer k , either finds a tree-cut decomposition of width at most $2k$ or decides that $\text{tcw}(G) > k$ in time $2^{O(k^2 \log k)} n^2$. Furthermore, Giannopoulou et al. [14] gave a constructive proof of the existence of an algorithm deciding whether the tree-cut width of a given graph G is at most k in $f(k)n$ time, where f is a computable recursive function. Very recently, Ganian et al. [13] performed experiments on computing optimal tree-cut decompositions using SAT-solvers.

► **Lemma 4.** Let T be a forest. Then $\text{tcw}(T) = 1$.

Proof. As clearly $\text{tcw}(T) \leq \text{tcw}(T + F)$ for any set of edges F , we may assume without loss of generality that T is a tree.

We define $X_t = \{t\}$ for all $t \in V(T)$, and consider the tree-cut decomposition (T, \mathcal{X}) , and pick an arbitrary vertex r to be the root of T .

As T is a tree, we have $\text{adh}(t) = 1$ for all $t \neq r$.

Furthermore, for each $t \in V(T)$, all vertices but t contained in the torso of t can be suppressed, and thus $\text{tor}(t) \leq 1$. ◀

► **Lemma 5.** Let G be a graph. Then $\text{tcw}(G + e) \leq \text{tcw}(G) + 2$ for any edge e .

Proof. Consider a tree-cut decomposition (T, \mathcal{X}) of G . This is also a tree-cut decomposition of $G + e$.

Clearly, the adhesion of any node of T can increase by at most 1.

The torso-size of a vertex can also increase by at most 2, as e can prevent at most both of its endpoints from being suppressed. ◀

► **Corollary 6.** Let G be a graph, and k the feedback edge number. Then $\text{tcw}(G) \leq 2k + 1$.

Proof. This directly follows from Lemmas 4 and 5. ◀

3 Hardness Results

All our hardness results are based on parameterized reductions from the MULTICOLORED CLIQUE problem, a well-known W[1]-hard problem. The so-called vertex selection gadgets are somewhat similar to those of Gupta et al. [15], however, the other gadgets in our reductions are different. Here we only discuss the main dissimilarities of the reductions we present here and the one of Gupta et al. [15]. We use one gadget for each edge whereas the reduction presented therein uses a single gadget for all edges between two color classes. This subtle difference allows us to bound not only treewidth of the resulting graph but rather both treedepth and the size of a feedback vertex set. It is worth noting that it is not clear whether the reduction of Gupta et al. [15] can, with some additional changes and work, yield hardness

for these parameters as well or not. On the other hand, we use some consistency gadget which is essentially a triangle (while the graph resulting from the reduction of Gupta et al. [15] is bipartite). Furthermore, in our reduction all of the vertices have either strictly ordered preferences or a tie between (the only) two agents they find acceptable. We defer details and further comments to a full version of this paper due to space reasons.

► **Theorem 7** (★). MAX-SRTI *parameterized by treedepth and feedback vertex set is W[1]-hard. Moreover, there is no $n^{o(\text{td}(G))}$ time algorithm for MAX-SRTI, unless ETH fails.*

Note that such a maximum stable matching corresponding to a clique of size k leaves only $2k(k-1)$ vertices uncovered. Thus, by adding $2k(k-1)$ vertices connected to all other vertices, we also get the W[1]-hardness for PERFECT-SRTI:

► **Corollary 8** (★). PERFECT-SRTI *parameterized by treedepth and feedback vertex set is W[1]-hard.*

From this, we get the W[1]-hardness of SRTI-EXISTENCE for treedepth by adding for each vertex a 3-cycle, ensuring that this vertex is matched (similarly to the 3-cycles c_{ij} , c'_{ij} , c''_{ij} for the vertex c_{ij} in the consistency gadgets).

► **Corollary 9** (★). SRTI-EXISTENCE *parameterized by treedepth is W[1]-hard.*

A different reduction partially using similar ideas and techniques yields the W[1]-hardness of MAX-SRTI for the parameter tree-cut width:

► **Theorem 10** (★). MAX-SRTI *parameterized by tree-cut width is W[1]-hard.*

4 Tractability Results

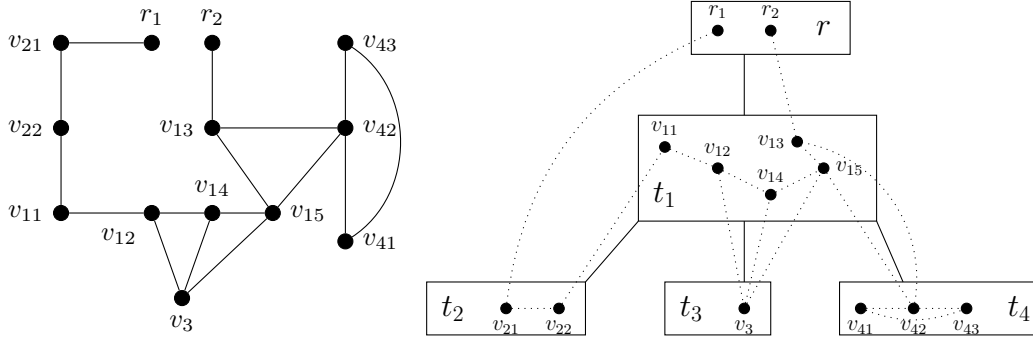
We present an FPT-algorithm for PERFECT-SRTI and SRTI-EXISTENCE. Given a tree-cut decomposition of the acceptability graph, we use dynamic programming to decide whether a solution exists or not. In the dynamic programming table for a node t we store information whether there exists a matching M for the set Y_t of vertices from the bags of the subtree of the tree-cut decomposition rooted in t . We allow that M is not stable in G but require that the blocking pairs are incident to vertices outside Y_t , and for some of the edges $\{v, w\}$ in $\text{cut}(t)$, we require the endpoint v in Y_t to be matched at least as good as he ranks w .

DP Tables. Before we describe the idea behind the table entries we store in our dynamic programming procedure, we introduce the following relaxation of matching stability.

► **Definition 11.** *Let (T, \mathcal{X}) be a nice tree-cut decomposition of G . For a node $t \in V(T)$, the closure of t ($\text{clos}(t)$) is the set of vertices in Y_t together with their neighbors, that is, $\text{clos}(t) := Y_t \cup N_G(Y_t)$. We say that a matching M on $\text{clos}(t)$ for some $t \in V(T)$ complies with a vector $\mathbf{h} \in \{-1, 0, 1\}^{\text{cut}(t)}$ if the following conditions hold:*

- *for each edge $e \in \text{cut}(t)$, we have $e \in M$ if and only if $\mathbf{h}_e = 0$;*
- *for each $e = \{v, w\} \in \text{cut}(t)$ with $v \in Y_t$ and $\mathbf{h}_e = 1$, we have $\text{rk}_v(M(v)) \leq \text{rk}_v(w)$, i.e. v ranks its partner (not being w by the previous condition) in M at least as good as w ; and*
- *every blocking pair contains a vertex from $V(G) \setminus Y_t$ not matched in M .*

Intuitively, if we set $\mathbf{h}_t(e) = 1$ for an edge $e = \{v, x\}$ in $\text{cut}(t)$ with $x \in X_t$, then we are searching for a matching M (in $G[\text{clos}(t)]$) for which we can guarantee that $\text{rk}_x(M(x)) \leq \text{rk}_x(v)$. Consequently, we know that e will not be blocking in an extension of such a matching. Contrary, if we set $\mathbf{h}_t(e) = -1$, then we allow x to prefer v over its partner (in particular, x



■ **Figure 2** An example of a graph G (left part of the picture) and its nice tree-cut decomposition (T, \mathcal{X}) (not of minimal width). The vertices of G are the circles, while the nodes of T are the rectangles. For a node $t \in V(T)$, the bag X_t contains exactly those vertices inside the rectangle. In the right picture, the solid edges are the edges of T , while the dotted edges are from G . The nodes t_1 and t_4 are light, while t_2 (because there is an edge connecting a vertex in t_2 to a vertex in r) and t_3 (because $\text{adh}(t_3) = 3$) are heavy.

may remain unmatched). Thus, for an extension of such a matching in order to maintain stability we have to secure that $\text{rk}_v(M(v)) \leq \text{rk}_v(x)$, since otherwise e will be blocking. Observe that if a matching complies with \mathbf{h} for a vector $\mathbf{h} \in \{-1, 0, 1\}^{\text{cut}(t)}$ with $\mathbf{h}(e) = 1$ for some edge $e \in \text{cut}(t)$, then it complies with $\hat{\mathbf{h}} \in \{-1, 0, 1\}^{\text{cut}(t)}$ which is the same as \mathbf{h} but for e is set to -1 (formally, $\hat{\mathbf{h}}(f) = \mathbf{h}(f)$ for $f \neq e$ and $\hat{\mathbf{h}}(e) = -1$). Clearly, any matching complying with \mathbf{h} complies with $\hat{\mathbf{h}}$, since the latter is more permissive.

For a node $t \in T$ its dynamic programming table is τ_t and it contains an entry for every $\mathbf{h} \in \{-1, 0, 1\}^{\text{cut}(t)}$. An entry $\tau_t[\mathbf{h}]$ is a matching $M \subseteq E(G[Y_t]) \cup \text{cut}(t)$ if M complies with \mathbf{h} . If no such matching for \mathbf{h} exists, then we set $\tau_t[\mathbf{h}] = \emptyset$. Note that the size of the table τ_t for a node t is upper-bounded by $3^{\text{tcw}(G)}$.

► **Example 12.** The graph and tree-cut decomposition are depicted in Figure 2:

For t_1 and $\mathbf{h}^1(\{v_{12}, v_{21}\}) = 0$ and $\mathbf{h}^1(\{v_{12}, r_2\}) = 1$, all stable matchings containing $\{t_1, v_{21}\}$ and $\{v_{21}, v_{22}\}$ are complying with \mathbf{h} . For t_2 and $\mathbf{h}^2(\{v_{21}, r_1\}) = 1$ and $\mathbf{h}^2(\{v_{22}, v_{11}\}) = -1$, the matching $M = \{\{v_{21}, v_{22}\}\}$ is complying with \mathbf{h}^2 . For t_3 and any vector \mathbf{h}^3 with $\mathbf{h}^3(\{v_3, v_{12}\}) = 1$, no matching complies with \mathbf{h}^3 : In such a matching, v_3 must be ranked at least as good as $\text{rk}_{v_3}(v_{12}) = 1$, but not to v_{12} , which is impossible. For t_4 and $\mathbf{h}^4(\{v_{42}, v_{15}\}) = 0 = \mathbf{h}^4(\{v_{42}, v_{13}\})$, no matching complying with \mathbf{h}^4 exists, as such a matching must match v_{42} to both v_{13} and v_{15} .

► **Theorem 13** (\star). PERFECT-SRTI and SRTI-EXISTENCE can be solved in $2^{O(k \log k)} n^{O(1)}$ time, where $k := \text{tcw}(G)$.

Proof Sketch. Let (T, \mathcal{X}) be a nice tree-cut decomposition of G of width k . We will first explain the algorithm for SRTI-EXISTENCE, and in the end we highlight how this algorithm can be adapted to PERFECT-SRTI. We compute the values $\tau_t[\mathbf{h}]$ by bottom-up induction over the tree T .

For a leaf $t \in V(T)$ and a vector \mathbf{h} , we enumerate all matchings M_t on $G[X_t \cup N(Y_t)]$. We check whether M_t complies with \mathbf{h} . If we find such a matching, then we store one of these matchings in $\tau_t[\mathbf{h}]$, and else set $\tau_t[\mathbf{h}] = \emptyset$. As $|X_t \cup N(Y_t)| \leq 2k$, and G is simple, the number of matchings is bounded by $2^{O(k \log k)}$.

The induction step, that is, computing the table entries for the inner nodes of the tree-cut decomposition is the most-involved part and sketched below.

For the root $r \in V(T)$, we have $Y_r = V(G)$ and $\text{cut}(r) = \emptyset$. Thus, a matching on $Y_r = V(G)$ complying with $\mathbf{h}^r \in \{-1, 0, 1\}^\emptyset$ is just a stable matching (note that \mathbf{h}^r is unique). Hence, G contains a stable matching if and only if $\tau_r[\mathbf{h}^r] \neq \emptyset$.

The induction step is executed for each $t \in V(T)$ and each $\mathbf{h} \in \{-1, 0, 1\}^{\text{cut}(t)}$, and therefore at most $n3^k$ times. As each execution takes $2^{O(k \log k)} n^{O(1)}$ time, the total running time of the algorithm is bounded by $2^{O(k \log k)} n^{O(1)}$.

To solve PERFECT-SRTI, we store in any dynamic programming table τ_t only matchings such that every vertex inside Y_t is matched.

Induction Step. In what follows we sketch how to solve the induction step.

Induction Step

Input: The acceptability graph G , rank functions rk_v for all $v \in V(G)$, a tree-cut decomposition (T, \mathcal{X}) , a node $t \in V(T)$, a vector $\mathbf{h} \in \{-1, 0, 1\}^{\text{cut}(t)}$, for each child c of t and each $\mathbf{h}^c \in \{-1, 0, 1\}^{\text{cut}(c)}$ the value $\tau_c[\mathbf{h}^c]$.

Task: Compute $\tau_t[\mathbf{h}]$.

Before we give the proof idea we first give the definition of light children classes. Intuitively, two light children of a node t are in the same class if, with respect to t , they behave in a similar way, that is, their neighborhood in X_t is the same and their table entries are compatible. In order to properly define the later notion we first need to introduce few auxiliary definitions. To simplify the notation, we assume that the edges of G are enumerated, that is, we have $E(G) = \{e_1, e_2, \dots, e_m\}$. For a vector $\mathbf{h} \in \{-1, 0, 1\}^{\text{cut}(t)}$, the i -th coordinate will always be the coordinate of the edge with the i -th lowest index in $\text{cut}(t)$.

► **Definition 14.** Let $t \in V(T)$ be a node. We define the signature $\text{sig}(t)$ to be the set $\{\mathbf{h} \in \{-1, 0, 1\}^{\text{cut}(t)} : \tau_t[\mathbf{h}] \neq \emptyset\}$.

Let c, d be light children of t . We write $c \diamond d$ if and only if $\text{sig}(c) = \text{sig}(d)$ and $N(c) = N(d)$, where we define $N(c) := N_G(Y_c)$ for each $c \in V(T)$.

It follows immediately that \diamond is an equivalence relation on light children of t . Furthermore, since each class of \diamond is identified by its signature and neighborhood in X_t , there are $O(k^2)$ classes of \diamond . Let $\mathcal{C}(c)$ denote the equivalence class of the light child c of t and let $N(\mathcal{C}) \subseteq X_t$ be the set of neighbors of the class \mathcal{C} of \diamond (i.e., $N(\mathcal{C})$ is the set of neighbors $N(Y_c) \subseteq X_t$ for $c \in \mathcal{C}$). Furthermore, let $\text{sig}(\mathcal{C})$ denote the signature of the class \mathcal{C} and similarly let $\text{sig}_x(\mathcal{C})$ denote the signature of \mathcal{C} with respect to its neighbor $x \in N(\mathcal{C})$.

► **Observation 15.** If \mathcal{C} is a class with $|\mathcal{C}| \geq 3$ and $(-1, -1) \notin \text{sig}(\mathcal{C})$, then there is no stable matching in G .

Proof Idea (Induction Step). Due to space reasons, we defer proof details to a full version of this paper. First, we will guess which edges incident to heavy children are in the matching M to be computed and which are not. Note that there are at most $k(2k+1)$ edges incident to heavy children of t , since their adhesion is at most k . Thus, we fix a matching between vertices in X_t and heavy children and what remains is to combine the guessed matching with matchings in their graphs; note that we can also guess these, however, this results in $(3^k)^{O(k)}$ guesses. Instead of trying all of the possibilities we prove that it is possible to reduce the number of heavy children matchings we try to extend to $k^{O(k)}$. It is worth noting that these choices will result in some further constraints the matching in the light children must fulfill.

Then for every class of the equivalence \diamond we guess whether its neighbor(s) in X_t are matched to it (i.e., matched to a vertex in a child or two in this class) or not. Let \mathcal{N} denote the guessed matching. Note that there are $k^{O(k)}$ such choices, since each vertex $v \in X_t$ is “adjacent” to at most $k+1$ classes of \diamond (i.e., there are at most k classes such that v is adjacent to a vertex in all of the children contained in this class) and choosing a class (or deciding not to be adjacent to any light child) for every vertex in X_t yields the claimed bound. We show that if a class of \diamond with $N(\mathcal{C}) = \{x\}$ is selected to be matched with its neighbor x , then it is possible to match x to the best child in this class (the one containing the top choice for x among these children); provided there exists a solution which is compatible with such a choice. We do this by showing a rather simple exchange argument.

Having resolved heavy children and light children with only one neighbor in X_t it remains to deal with children with two neighbors. We generalise the exchange argument we provide for classes with one neighbor. Then, we prove that many combinations of \mathcal{N} and a signature of a class \mathcal{C} allows us to reduce the number of children in \mathcal{C} in which we have to search for a partner of a vertex in $N(\mathcal{C})$ to a constant (in fact, four). We call such classes the *good classes*. However, there are classes where this is not possible (call these the *bad classes*). Consequently, there are only 4^k possibilities how to match vertices in X_t to good classes of children. Finally, we characterise the bad classes and use this characterisation to show how to reduce the question of existence of a (perfect) matching complying with \mathbf{h} and obeying all the constraints of heavy children to an instance of 2-SAT (similarly to Feder [9]). ◀

► **Corollary 16** (\star). *A factor- $\frac{1}{2}$ -approximation for MAX-SRTI can be computed in $2^{O(k \log k)} n^{O(1)}$ time, where $k := \text{tcw}(G)$.*

Using standard techniques and the polynomial-time algorithm for graphs of bounded treewidth by Adil et al. [1], we obtain an FPT-algorithm for MAX-SRTI (and therefore also PERFECT-SRTI and SRTI-EXISTENCE) parameterized by feedback edge number:

► **Theorem 17** (\star). *MAX-SRTI can be solved in $2^{\text{fes}(G)} n^{O(1)}$ time.*

5 Conclusion

Taking the viewpoint of parameterized graph algorithmics, we investigated the line between fixed-parameter tractability and W[1]-hardness of STABLE ROOMMATES WITH TIES AND INCOMPLETE LISTS. Studying parameterizations mostly relating to the “tree-likeness” of the underlying acceptability graph, we arrived at a fairly complete picture (refer to Figure 1) of the corresponding parameterized complexity landscape. There is a number of future research issues stimulated by our work. First, we did not touch on questions about polynomial kernelizability of the fixed-parameter tractable cases. Indeed, for the parameter feedback edge number we believe that a polynomial kernel should be possible. Another issue is how tight the running time for our fixed-parameter algorithm for the parameter tree-cut width k is; more specifically, can we show that our exponential factor $k^{O(k)}$ is basically optimal or can it be improved to say $2^{O(k)}$? Also the case of SRTI-EXISTENCE parameterized by feedback vertex number remained open (see Figure 1). Based on preliminary considerations, we conjecture it to be W[1]-hard. Clearly, there is still a lot of room to study STABLE ROOMMATES WITH TIES AND INCOMPLETE LISTS through the lens of further graph parameters. On a general note, we emphasize that so far our investigations are on the purely theoretic and classification side; practical algorithmic considerations are left open for future research.

References

- 1 Deeksha Adil, Sushmita Gupta, Sanjukta Roy, Saket Saurabh, and Meirav Zehavi. Parameterized algorithms for stable matching with ties and incomplete lists. *Theor. Comput. Sci.*, 723:1–10, 2018.
- 2 John J. Bartholdi III and Michael Trick. Stable Matching with Preferences Derived from a Psychological Model. *Oper. Res. Lett.*, 5(4):165–169, 1986.
- 3 Robert Brederbeck, Jiehua Chen, Ugo Paavo Finnendahl, and Rolf Niedermeier. Stable Roommate with Narcissistic, Single-Peaked, and Single-Crossing Preferences. In *Proc. of ADT '17*, volume 10576 of *LNCS*, pages 315–330. Springer, 2017.
- 4 Jianer Chen, Benny Chor, Mike Fellows, Xiuzhen Huang, David Juedes, Iyad A. Kanj, and Ge Xia. Tight lower bounds for certain parameterized NP-hard problems. *Information and Computation*, 201(2):216–231, 2005.
- 5 Jiehua Chen, Danny Hermelin, Manuel Sorge, and Harel Yedidsion. How Hard Is It to Satisfy (Almost) All Roommates? In *Proc. of ICALP '18*, volume 107 of *LIPICs*, pages 35:1–35:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.
- 6 Jiehua Chen, Rolf Niedermeier, and Piotr Skowron. Stable Marriage with Multi-Modal Preferences. In *Proc. of EC '18*, pages 269–286. ACM, 2018.
- 7 Ágnes Cseh, Robert W. Irving, and David F. Manlove. The Stable Roommates Problem with Short Lists. *Theory Comput. Syst.*, 63(1):128–149, 2019.
- 8 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.
- 9 Tomás Feder. A new fixed point approach for stable networks and stable marriages. *J. Comput. Syst. Sci.*, 45(2):233–284, 1992.
- 10 Anh-Tuan Gai, Dmitry Lebedev, Fabien Mathieu, Fabien de Montgolfier, Julien Reynier, and Laurent Viennot. Acyclic Preference Systems in P2P Networks. In *Proc. of Euro-Par '07*, volume 4641 of *LNCS*, pages 825–834. Springer, 2007.
- 11 Robert Ganian, Eun Jung Kim, and Stefan Szeider. Algorithmic Applications of Tree-Cut Width. In *Proc. of MFCS '15*, volume 9235 of *LNCS*, pages 348–360. Springer, 2015.
- 12 Robert Ganian, Fabian Klute, and Sebastian Ordyniak. On Structural Parameterizations of the Bounded-Degree Vertex Deletion Problem. In *Proc. of STACS '18*, volume 96 of *LIPICs*, pages 33:1–33:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.
- 13 Robert Ganian, Neha Lodha, Sebastian Ordyniak, and Stefan Szeider. SAT-encodings for treecut width and treedepth. In *Proc. of ALENEX '19*, pages 117–129. SIAM, 2019.
- 14 Archontia C. Giannopoulou, O-joung Kwon, Jean-Florent Raymond, and Dimitrios M. Thilikos. Lean Tree-Cut Decompositions: Obstructions and Algorithms. In *Proc. of STACS '19*, volume 126 of *LIPICs*, pages 32:1–32:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019.
- 15 Sushmita Gupta, Saket Saurabh, and Meirav Zehavi. On Treewidth and Stable Marriage. *CoRR*, abs/1707.05404, 2017. arXiv:1707.05404.
- 16 Dan Gusfield and Robert W. Irving. *The Stable Marriage Problem - Structure and Algorithms*. MIT Press, 1989.
- 17 Russell Impagliazzo and Ramamohan Paturi. On the Complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.
- 18 Robert W. Irving. An Efficient Algorithm for the “Stable Roommates” Problem. *J. Algorithms*, 6(4):577–595, 1985.
- 19 Kazuo Iwama, Shuichi Miyazaki, Yasufumi Morita, and David Manlove. Stable Marriage with Incomplete Lists and Ties. In *Proc. of ICALP '99*, volume 1644 of *LNCS*, pages 443–452. Springer, 1999.
- 20 Eun Jung Kim, Sang-il Oum, Christophe Paul, Ignasi Sau, and Dimitrios M. Thilikos. An FPT 2-Approximation for Tree-Cut Decomposition. *Algorithmica*, 80(1):116–135, 2018.
- 21 Ton Kloks. *Treewidth, Computations and Approximations*, volume 842 of *LNCS*. Springer, 1994. doi:10.1007/BFb0045375.

- 22 Christian Komusiewicz and Rolf Niedermeier. New Races in Parameterized Algorithmics. In *Proc. of MFCS '12*, volume 7464 of *LNCS*, pages 19–30. Springer, 2012.
- 23 David Manlove. The structure of stable marriage with indifference. *Discrete Appl. Math.*, 122(1-3):167–181, 2002.
- 24 David Manlove, Robert W. Irving, Kazuo Iwama, Shuichi Miyazaki, and Yasufumi Morita. Hard variants of stable marriage. *Theor. Comput. Sci.*, 276(1-2):261–279, 2002.
- 25 David F. Manlove. *Algorithmics of Matching Under Preferences*, volume 2 of *Series on Theoretical Computer Science*. WorldScientific, 2013.
- 26 Dániel Marx and Ildikó Schlotter. Parameterized Complexity and Local Search Approaches for the Stable Marriage Problem with Ties. *Algorithmica*, 58(1):170–187, 2010.
- 27 Dániel Marx and Paul Wollan. Immersions in Highly Edge Connected Graphs. *SIAM J. Discrete Math.*, 28(1):503–520, 2014.
- 28 Dániel Marx and Ildikó Schlotter. Stable assignment with couples: Parameterized complexity and local search. *Discrete Optim.*, 8(1):25–40, 2011.
- 29 Eric McDermid. A $3/2$ -Approximation Algorithm for General Stable Marriage. In *Proc. of ICALP '09*, pages 689–700. Springer, 2009.
- 30 Kitty Meeks and Baharak Rastegari. Stable Marriage with Groups of Similar Agents. In *Proc. of WINE '18*, volume 11316 of *LNCS*, pages 312–326. Springer, 2018. doi:10.1007/978-3-030-04612-5_21.
- 31 Matthias Mnich and Ildikó Schlotter. Stable Marriage with Covering Constraints-A Complete Computational Trichotomy. In *Proc. of SAGT '17*, volume 10504 of *LNCS*, pages 320–332. Springer, 2017. doi:10.1007/978-3-319-66700-3_25.
- 32 Eytan Ronn. NP-complete stable matching problems. *J. Algorithms*, 11(2):285–304, 1990.
- 33 Alvin E. Roth, Tayfun Sönmez, and M. Utku Ünver. Pairwise kidney exchange. *J. Econ. Theory*, 125(2):151–188, 2005.
- 34 Alvin E. Roth, Tayfun Sönmez, and M. Utku Ünver. Efficient Kidney Exchange: Coincidence of Wants in Markets with Compatibility-Based Preferences. *Am. Econ. Rev.*, 97(3):828–851, June 2007.
- 35 Paul Wollan. The structure of graphs not admitting a fixed immersion. *J. Comb. Theory, Ser. B*, 110:47–66, 2015. doi:10.1016/j.jctb.2014.07.003.
- 36 H. Yanagisawa. *Approximation Algorithms for Stable Marriage Problems*. PhD thesis, Kyoto University, 2007.

Path and Ancestor Queries over Trees with Multidimensional Weight Vectors

Meng He

Faculty of Computer Science, Dalhousie University, Canada
mhe@cs.dal.ca

Serikzhan Kazi

Faculty of Computer Science, Dalhousie University, Canada
skazi@dal.ca

Abstract

We consider an ordinal tree T on n nodes, with each node assigned a d -dimensional weight vector $\mathbf{w} \in \{1, 2, \dots, n\}^d$, where $d \in \mathbb{N}$ is a constant. We study path queries as generalizations of well-known *orthogonal range queries*, with one of the dimensions being tree topology rather than a linear order. Since in our definitions d only represents the number of dimensions of the weight vector without taking the tree topology into account, a path query in a tree with d -dimensional weight vectors generalize the corresponding $(d + 1)$ -dimensional orthogonal range query. We solve *ancestor dominance reporting* problem as a direct generalization of dominance reporting problem, in time $\mathcal{O}(\lg^{d-1} n + k)$ and space of $\mathcal{O}(n \lg^{d-2} n)$ words, where k is the size of the output, for $d \geq 2$. We also achieve a tradeoff of $\mathcal{O}(n \lg^{d-2+\epsilon} n)$ words of space, with query time of $\mathcal{O}((\lg^{d-1} n)/(\lg \lg n)^{d-2} + k)$, for the same problem, when $d \geq 3$. We solve *path successor problem* in $\mathcal{O}(n \lg^{d-1} n)$ words of space and time $\mathcal{O}(\lg^{d-1+\epsilon} n)$ for $d \geq 1$ and an arbitrary constant $\epsilon > 0$. We propose a solution to *path counting problem*, with $\mathcal{O}(n(\lg n/\lg \lg n)^{d-1})$ words of space and $\mathcal{O}((\lg n/\lg \lg n)^d)$ query time, for $d \geq 1$. Finally, we solve *path reporting problem* in $\mathcal{O}(n \lg^{d-1+\epsilon} n)$ words of space and $\mathcal{O}((\lg^{d-1} n)/(\lg \lg n)^{d-2} + k)$ query time, for $d \geq 2$. These results match or nearly match the best tradeoffs of the respective range queries. We are also the first to solve path successor even for $d = 1$.

2012 ACM Subject Classification Information systems \rightarrow Data structures; Theory of computation \rightarrow Data structures design and analysis; Information systems \rightarrow Multidimensional range search

Keywords and phrases path queries, range queries, algorithms, data structures, theory

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.45

Related Version A full version of the paper is available at <https://arxiv.org/abs/1910.01147v1>.

Funding This work was supported by NSERC of Canada.

1 Introduction

The problem of preprocessing a weighted tree, i.e., a tree in which each node is associated with a weight value, to support various queries evaluating a certain function on the node weights of a given path, has been extensively studied [2, 6, 12, 17, 8, 15, 4]. For example, in *path counting* (resp. *path reporting*), the nodes of the given path with weights lying in the given query interval are counted (resp. reported). These queries address the needs of fast information retrieval from tree-structured data such as XML and tree network topology.

For many applications, meanwhile, a node in a tree is associated with not just a single weight, but rather with a vector of weights. Consider a simple scenario of an online forum thread, where users can rate responses and respond to posts. Induced is a tree-shaped structure with posts representing nodes, and replies to a post being its children. One can imagine enumerating all the ancestor posts of a given post that are not too short and have sufficiently high average ratings. Ancestor dominance query, which is among the problems we consider, provides an appropriate model in this case.



© Meng He and Serikzhan Kazi;

licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 45; pp. 45:1–45:17

Leibniz International Proceedings in Informatics



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

We define a d -dimensional weight vector $\mathbf{w} = (w_1, w_2, \dots, w_d)$ to be a vector with d components, each in rank space $[n]$,¹ i.e. $\mathbf{w} \in [n]^d$, with w_i being referred to as the i th weight of \mathbf{w} . We then consider an ordinal tree T on n nodes, each node x of which is assigned a d -dimensional weight vector $\mathbf{w}(x)$. The queries we will define all give a d -dimensional orthogonal range $Q = \prod_{i=1}^d [q_i, q'_i]$, and a weight vector \mathbf{w} is in Q iff for any $i \in [1, d]$, $q_i \leq w_i \leq q'_i$ holds. In our queries, then, we are given a pair of vertices $x, y \in T$, and an arbitrary orthogonal range Q . With $P_{x,y}$ being the path from x to y in the tree T , the goal is to preprocess the tree T for the following types of queries:

- *Path Counting*: return $|\{z \in P_{x,y} \mid \mathbf{w}(z) \in Q\}|$.
- *Path Reporting*: enumerate $\{z \in P_{x,y} \mid \mathbf{w}(z) \in Q\}$.
- *Path Successor*: return $\operatorname{argmin}\{w_1(z) \mid z \in P_{x,y} \text{ and } \mathbf{w}(z) \in Q\}$.²
- *Ancestor Dominance Reporting*: a special case of path reporting, in which y is the root of the tree and $q'_i = +\infty$ for all $i \in [d]$. That is, the query reports the ancestors of x whose weight vectors dominate the vector $\mathbf{q} = (q_1, q_2, \dots, q_d)$.

This indeed is a natural generalization of the traditional weighted tree, which we refer to as “scalarly-weighted”, to the case when the weights are multidimensional vectors. At the same time, when the tree degenerates into a single path, these queries become respectively $(d + 1)$ -dimensional orthogonal range counting, reporting, successor, as well as $(d + 1)$ -dimensional dominance reporting queries. Thus, the queries we study are generalizations of these fundamental geometric queries in high dimensions. We also go along with the state-of-art in orthogonal range search by considering weights in rank space, since the case in which weights are from a larger universe can be reduced to it [10].

1.1 Previous Work

Path Queries in Weighted Trees. For scalarly-weighted trees, Chazelle [6] gave an $\mathcal{O}(n)$ -word *emulation dag*-based data structure that answers path counting queries in $\mathcal{O}(\lg n)$ time;³ it works primarily with topology of the tree and is thus oblivious to the distribution of weights. Later, He et al. [15] proposed a solution with $nH(W_T) + o(n \lg \sigma)$ bits of space and $\mathcal{O}(\frac{\lg \sigma}{\lg \lg n} + 1)$ query time, when the weights are from $[\sigma]$; here, $H(W_T)$ is the entropy of the multiset of the weights in T . When $\sigma \ll n$, this matters.

He et al. [15] introduced and solved path reporting problem using linear space and $\mathcal{O}((1 + k) \lg \sigma)$ query time, and $\mathcal{O}(n \lg \lg \sigma)$ words of space but $\mathcal{O}(\lg \sigma + k \lg \lg \sigma)$ query time, in the word-RAM model; henceforth we reserve k for the size of the output. Patil et al. [21] presented a succinct data structure for path reporting with $n \lg \sigma + 6n + o(n \lg \sigma)$ bits of space and $\mathcal{O}((\lg n + k) \lg \sigma)$ query time. An optimal-space solution with $nH(W_T) + o(n \lg \sigma)$ bits of space and $\mathcal{O}((1 + k)(\frac{\lg \sigma}{\lg \log n} + 1))$ reporting time is due to He et al. [15]. One of the tradeoffs proposed by Chan et al. [4], requires $\mathcal{O}(n \lg^\epsilon n)$ words of space for the query time of $\mathcal{O}(\lg \lg n + k)$.

Orthogonal Range Queries. Dominance reporting in 3D was solved by Chazelle and Edelsbrunner [7] in linear space with either $\mathcal{O}((1 + k) \lg n)$ or $\mathcal{O}(\lg^2 n + k)$ time, in pointer-machine (PM) model, with the latter being improved to $\mathcal{O}(\lg n \lg \lg n + k)$ by Makris and Tsakalidis [18]. Same authors [18] developed, in the word-RAM, a linear-size, $\mathcal{O}(\log n + k)$

¹ Throughout this paper, $[n]$ stands for $\{1, 2, \dots, n\}$ for any positive integer n .

² For path successor, we assume that $q'_1 = \infty$; if not, we need only check whether the 1st weight of the returned node is at most q'_1 .

³ $\lg x$ denotes $\log_2 x$ in this paper.

and $\mathcal{O}((\lg \lg n \lg \lg n + k) \lg \lg n)$ query-time data structures for the unrestricted case and for points in rank space, respectively. Nekrich [19] presented a word-RAM data structure for points in rank space, supporting queries in $\mathcal{O}((\lg \lg n)^2 + k)$ time, and occupying $\mathcal{O}(n \lg n)$ words; this space was later reduced to linear by Afshani [1], retaining the same query time. Finally, in the same model, a linear-space solution with $\mathcal{O}(\log \log n + k)$ query time was designed for 3D dominance reporting in rank space [1, 3]. In the PM model, Afshani [1] also presented an $\mathcal{O}(\log n + k)$ query time, linear-space data structure for the points in \mathbb{R}^3 .

For the word-RAM model, JáJá et al. [16] generalized the range counting problem for $d \geq 2$ dimensions and proposed a data structure with $\mathcal{O}(n(\lg n / \lg \lg n)^{d-2})$ words of space and $\mathcal{O}((\lg n / \lg \lg n)^{d-1})$ query time. Chan et al. [5] solved orthogonal range reporting in 3D rank space in $\mathcal{O}(n \lg^{1+\epsilon} n)$ words of space and $\mathcal{O}(\lg \lg n + k)$ query time.

Nekrich and Navarro [20] proposed two tradeoffs for range successor, with either $\mathcal{O}(n)$ or $\mathcal{O}(n \lg \lg n)$ words of space, and respectively with $\mathcal{O}(\lg^\epsilon n)$ or $\mathcal{O}((\lg \lg n)^2)$ query time. Zhou [23] later improved upon the query time of the second tradeoff by a factor of $\lg \lg n$, within the same space. Both results are for points in rank space.

1.2 Our Results

As d -dimensional path queries generalize the corresponding $(d + 1)$ -dimensional orthogonal range queries, we compare results on them to show that our bounds match or nearly match the best results or some of the best tradeoffs on geometric queries in Euclidean space. We present solutions for the (we assume d is a positive integer constant):

- ancestor dominance reporting problem, in $\mathcal{O}(n \lg^{d-2} n)$ words of space and $\mathcal{O}(\lg^{d-1} n + k)$ query time for $d \geq 2$. When $d = 2$, this matches the space bound for 3D dominance reporting of [1, 3], while still providing efficient query support. When $d \geq 3$, we also achieve a tradeoff of $\mathcal{O}(n \lg^{d-2+\epsilon} n)$ words of space, with query time of $\mathcal{O}(\lg^{d-1} n / (\lg \lg n)^{d-2} + k)$;
- path successor problem, in $\mathcal{O}(n \lg^{d-1} n)$ words and $\mathcal{O}(\log^{d-1+\epsilon} n)$ query time, for an arbitrarily small positive constant ϵ , and $d \geq 2$. These bounds match the first tradeoff for range successor of Nekrich and Navarro [20].⁴ Previously this problem has not been studied even on scalarly-weighted trees;
- path counting problem, in $\mathcal{O}(n(\frac{\log n}{\log \log n})^{d-1})$ words of space and $\mathcal{O}((\frac{\log n}{\log \log n})^d)$ query time for $d \geq 1$. This matches the best bound for range counting in $d + 1$ dimensions [16];
- path reporting problem, in $\mathcal{O}(n \lg^{d-1+\epsilon} n)$ words of space and $\mathcal{O}((\lg^{d-1} n) / (\lg \lg n)^{d-2} + k)$ query time, for $d \geq 2$. When $d = 2$, the space matches that of the corresponding result of Chan et al. [5] on 3D range reporting, while the first term in the query complexity is slowed down by a sub-logarithmic factor.

To achieve our results, we introduce a framework for solving range sum queries in arbitrary semigroups and extend base-case data structures to higher dimensions using universe reduction. A careful design with results hailing from succinct data structures and tree representations has been necessary, both for building space- and time-efficient base data structures, and for porting, using *tree extractions*, the framework of range trees decompositions from general point-sets to tree topologies (Lemma 5). We employ a few novel techniques, such as extending the notion of *maximality* in Euclidean sense to tree topologies, and providing the means of efficient computation thereof (Section 5). Given a weighted tree T , we propose efficient means of zooming into the nodes of T with weights in the given range in the range tree (Lemma 13). Given the ubiquitousness of the concepts, these technical contributions are likely to be of independent interest.

⁴ which can be generalized to higher dimensions via standard techniques based on range trees

Described further are our solutions to ancestor dominance reporting and path successor problems, while our data structures for path counting and path reporting are deferred to the full version of this paper.

2 Preliminaries

Notation. Given a d -dimensional weight vector $\mathbf{w} = (w_1, w_2, \dots, w_d)$, we define vector $\mathbf{w}_{i,j}$ to be $(w_i, w_{i+1}, \dots, w_j)$. We extend the definition to a range $Q = \prod_{i=1}^d [q_i, q'_i]$ by setting $Q_{i,j} = \prod_{k=i}^j [q_k, q'_k]$. We use the symbol \succeq for domination: $\mathbf{p} \succeq \mathbf{q}$ iff \mathbf{p} dominates \mathbf{q} . With $d' \leq d$ and $0 < \epsilon < 1$ being constants, a weight vector \mathbf{w} is said to be (d', d, ϵ) -dimensional iff $\mathbf{w} \in [n]^{d'} \times \lceil \lceil \lg^\epsilon n \rceil \rceil^{d-d'}$; i.e., each of its first d' weights is drawn from $[n]$, while each of its last $d - d'$ weights is in $\lceil \lceil \lg^\epsilon n \rceil \rceil$. When stating theorems, we define $i/0 = \infty$ for $i > 0$.

During a preorder traversal of a given tree T , the i th node visited is said to have *preorder rank* i . Preorder ranks are commonly used to identify tree nodes in various succinct data structures which we use as building blocks. Thus, we also identify a node by its preorder rank, i.e., node i in T is the node with preorder rank i in T . The path between the nodes $x, y \in T$ is denoted as $P_{x,y}$, both ends inclusive. For a node $x \in T$, its set of ancestors, denoted as $\mathcal{A}(x)$, includes x itself; $\mathcal{A}(x) \setminus \{x\}$ is then the set of *proper ancestors* of x . Given two nodes $x, y \in T$, where $y \in \mathcal{A}(x)$, we set $A_{x,y} \triangleq P_{x,y} \setminus \{y\}$.

Succinct Representations of Ordinal Trees. Succinct representations of unlabeled and labeled ordinal trees is a widely researched area. In a labeled tree, each node is associated with a label over an alphabet. Such a label can serve as a scalar weight; in our solutions, however, they typically categorize tree nodes into different classes. Hence we call these assigned values labels instead of weights. We summarize the previous result used in our solutions, in which a node (resp. ancestor) with label α is called an α -node (resp. α -ancestor):

► **Lemma 1** ([15, 13]). *Let T be an ordinal tree on n nodes, each having a label drawn from $[\sigma]$, where $\sigma = \mathcal{O}(\lg^\epsilon n)$ for some constant $0 < \epsilon < 1$. Then, T can be represented in $n(\lg \sigma + 2) + \mathcal{o}(n)$ bits of space to support the following operations, for any node $x \in T$, in $\mathcal{O}(1)$ time: $\text{child}(T, x, i)$, the i -th child of x ; $\text{depth}(T, x)$, the number of ancestors of x ; $\text{level_anc}(T, x, i)$, the i -th lowest proper ancestor of x ; $\text{pre_rank}_\alpha(T, x)$, the number of α -nodes that precede x in preorder; $\text{pre_select}_\alpha(T, i)$, the i -th α -node in preorder; and $\text{level_anc}_\alpha(T, x, i)$, the i -th lowest α -ancestor of x .*

Lemma 1 also includes a result on representing an unlabeled ordinal tree, which corresponds to $\sigma \equiv 1$, in $2n + \mathcal{o}(n)$ bits [13]. Another important special case is that of $\sigma = 2$; here, T is referred to as a $0/1$ -labeled tree, and the storage space becomes $3n + \mathcal{o}(n)$ bits.

Tree Extraction. *Tree extraction* [15] filters out a subset of nodes while preserving the underlying ancestor-descendant relationship among the nodes. Namely, given a subset X of tree nodes called *extracted nodes*, an *extracted tree* T_X can be obtained from the original tree T as follows. Consider each node $v \notin X$ in an arbitrary order; let p be v 's parent. We remove v and all its incident edges from T , and plug all its children v_1, v_2, \dots, v_k (preserving their left-to-right order) into the slot now freed from v in p 's list of children. After removing all the non-extracted nodes, if the resulting forest F_X is a tree, then $T_X \equiv F_X$. Otherwise, we create a dummy root r and insert the roots of the trees in F_X as the children of r , in the original left-to-right order. The preorder ranks and depths of r are both 0, so that those of non-dummy nodes still start at 1. An original node $x \in X$ of T and its copy, x' , in T_X

are said to *correspond* to each other; x' is also said to be the T_X -view of x , and x is the T -source of x' . The T_X -view of a node $y \in T$ (y is not required to be in X) is more generally defined to be the node $y' \in T_X$ corresponding to the lowest extracted ancestor of y , i.e. to the lowest node in $\mathcal{A}(y) \cap X$.

Representation of a Range Tree on Node Weights by Hierarchical Tree Extraction.

Range trees are widely used in solutions to query problems in Euclidean space. He et al. [15] further applied the idea of range trees to scalarly-weighted trees. They defined a conceptual range tree on node weights and represented it by a hierarchy of tree extractions. We summarize its workings when the weights are in rank space.

We first define a conceptual range tree on $[n]$ with branching factor f , where $f = \mathcal{O}(\lg^\epsilon n)$ for some constant $0 < \epsilon < 1$. Its root represents the entire range $[n]$. Starting from the root level, we keep partitioning each range, $[a, b]$, at the current lowest level into f child ranges $[a_1, b_1], \dots, [a_f, b_f]$, where $a_i = \lceil (i-1)(b-a+1)/f \rceil + a$ and $b_i = \lceil i(b-a+1)/f \rceil + a - 1$. This ensures that, if weight $j \in [a, b]$, then j is contained in the child range with subscript $\lceil f(j-a+1)/(b-a+1) \rceil$, which can be determined in $\mathcal{O}(1)$ time. We stop partitioning a range when its size is 1. This range tree has $h = \lceil \log_f n \rceil + 1$ levels. The root is at level 1 and the bottom level is level h .

For $1 \leq l < h$, we construct an auxiliary tree T_l for level l of this range tree as follows: Let $[a_1, b_1], \dots, [a_m, b_m]$ be the ranges at level l . For a range $[a, b]$, let $F_{a,b}$ stand for the extracted forest of the nodes of T with weights in $[a, b]$. Then, for each range $[a_i, b_i]$, we extract F_{a_i, b_i} and plug its roots as children of a dummy root r_l , retaining the original left-to-right order of the roots within the forest. Between forests, the roots in $F_{a_{i+1}, b_{i+1}}$ are the right siblings of the roots in F_{a_i, b_i} , for any $i \in [m-1]$. We then label the nodes of T_l using the reduced alphabet $[f]$, as follows. Note that barring the dummy root r_l , there is a bijection between the nodes of T and those of T_l . Let $x_l \in T_l$ be the node corresponding to $x \in T$. In the range tree, let $[a, b]$ be the level- l range containing the weight of x . Then, at level $l+1$, if the weight of x is contained in the j th child range of $[a, b]$, then $x_l \in T_l$ is labeled j . Each T_l is represented by Lemma 1 in $n(\lg f + 2) + \mathcal{O}(n)$ bits, so the total space cost of all the T_l 's is $n \lg n + (2n + \mathcal{O}(n)) \log_f n$ bits. When $f = \omega(1)$, this space cost is $n + \mathcal{O}(n)$ words. This completes the outline of hierarchical tree extraction. Henceforth, we shorthand as \mathcal{T}_v the extraction from T of the nodes with weights in v 's range, for a node v of the range tree. The following lemma maps x_l to x_{l+1} :

► **Lemma 2** ([15]). *Given a node $x_l \in T_l$ and the range $[a, b]$ of level l containing the weight of x , node $x_{l+1} \in T_{l+1}$ can be located in $\mathcal{O}(1)$ time, for any $l \in [h-2]$.*

Later, Chan et al. [4] augmented this representation with *ball-inheritance data structure* to map an arbitrary x_l back to x :

► **Lemma 3** ([4]). *Given a node $x_l \in T_l$, where $1 \leq l < h$, the node $x \in T$ that corresponds to x_l can be found using $\mathcal{O}(n \lg n \cdot \mathfrak{s}(n))$ bits of additional space and $\mathcal{O}(\mathfrak{t}(n))$ time, where (a) $\mathfrak{s}(n) = \mathcal{O}(1)$ and $\mathfrak{t}(n) = \mathcal{O}(\lg^\epsilon n)$; or (b) $\mathfrak{s}(n) = \mathcal{O}(\lg \lg n)$ and $\mathfrak{t}(n) = \mathcal{O}(\lg \lg n)$; or (c) $\mathfrak{s}(n) = \mathcal{O}(\lg^\epsilon n)$ and $\mathfrak{t}(n) = \mathcal{O}(1)$.*

Path Minimum in (Scalarly-)Weighted Trees. In a weighted tree, path minimum query asks for the node with the smallest weight in the given path. We summarize the best result on path minimum; in it, $\alpha(m, n)$ and $\alpha(n)$ are the inverse-Ackermann functions:

► **Lemma 4** ([4]). *An ordinal tree T on n weighted nodes can be indexed (a) using $\mathcal{O}(m)$ bits of space to support path minimum queries in $\mathcal{O}(\alpha(m, n))$ time and $\mathcal{O}(\alpha(m, n))$ accesses to the weights of nodes, for any integer $m \geq n$; or (b) using $2n + \mathcal{o}(n)$ bits of space to support path minimum queries in $\mathcal{O}(\alpha(n))$ time and $\mathcal{O}(\alpha(n))$ accesses to the weights of nodes. In particular, when $m = \Theta(n \lg^{**} n)$,⁵ one has $\alpha(m, n) = \mathcal{O}(1)$, and therefore (a) includes the result that T can be indexed in $\mathcal{O}(n \lg^{**} n)$ bits of space to support path minimum queries in $\mathcal{O}(1)$ time and $\mathcal{O}(1)$ accesses to the weights of nodes.*

3 Reducing to Lower Dimensions

This section presents a general framework for reducing the problem of answering a d -dimensional query to the same query problem in $(d - 1)$ dimensions, by generalizing the standard technique of range tree decomposition for the case of tree topologies weighted with multidimensional vectors. To describe this framework, we introduce a *d -dimensional semigroup path sum query problem* which is a generalization of all the query problems we consider in this paper. Let (G, \oplus) be a semigroup and T a tree on n nodes, in which each node x is assigned a d -dimensional weight vector $\mathbf{w}(x)$ and a semigroup element $g(x)$, with the semigroup sum operator denoted as \oplus . Then, in a d -dimensional semigroup path sum query, we are given a path $P_{x,y}$ in T , an orthogonal query range Q in d -dimensional space, and we are asked to compute $\sum_{z \in P_{x,y} \text{ and } \mathbf{w}(z) \in Q} g(z)$. When the weight vectors of the nodes and the query range are both from a (d', d, ϵ) -dimensional space, the *(d', d, ϵ) -dimensional semigroup path sum query problem* is defined analogously.

The following lemma presents our framework for solving a *d -dimensional semigroup path sum query problem*; its counterpart in (d', d, ϵ) -dimensional space is given in Section 4.

► **Lemma 5.** *Let d be a positive integer constant. Let $G^{(d-1)}$ be an $\mathfrak{s}(n)$ -word data structure for a $(d-1)$ -dimensional semigroup path sum problem of size n . Then, there is an $\mathcal{O}(\mathfrak{s}(n) \lg n + n)$ -word data structure $G^{(d)}$ for a d -dimensional semigroup path sum problem of size n , whose components include $\mathcal{O}(\lg n)$ structures of type $G^{(d-1)}$, each of which is constructed over a tree on $n + 1$ nodes. Furthermore, $G^{(d)}$ can answer a d -dimensional semigroup path sum query by performing $\mathcal{O}(\lg n)$ $(d - 1)$ -dimensional queries using these components and returning the semigroup sum of the answers. Determining which queries to perform on structures of type $G^{(d-1)}$ requires $\mathcal{O}(1)$ time per query.⁶*

Proof. We define a conceptual range tree R with branching factor 2 over the d th weights of the nodes of T and represent it using hierarchical tree extraction as in Section 2. For each level l of the range tree, we define a tree T_l^* with the same topology as T_l . We assign $(d - 1)$ -dimensional weight vectors and semigroup elements to each node, x' , in T_l^* as follows. If x' is not the dummy root, then $\mathbf{w}(x')$ is set to be $(w_1(x), \dots, w_{d-1}(x))$, where x is the node of T corresponding to x' . We also set $g(x') = g(x)$. If x' is the dummy root, then its first $(d - 1)$ weights are $-\infty$, while $g(x')$ is set to an arbitrary element of the semigroup. We then construct a data structure, G_l , of type $G^{(d-1)}$, over T_l^* . The data structure $G^{(d)}$ thus

⁵ $\lg^{**} n$ stands for the number of times an iterated logarithm function \lg^* needs to be applied to n in order for the result to become at most 1.

⁶ It may be tempting to simplify the statement of the lemma by defining $\mathfrak{t}(n)$ as the query time of $G^{(d-1)}$ and claiming that $G^{(d)}$ can answer a query in $\mathcal{O}(\mathfrak{t}(n) \lg n)$ time. However, this bound is too loose when applying this lemma to reporting queries.

comprises the structures T_l and G_l , over all l . The range tree has $\mathcal{O}(\lg n)$ levels, each T_l^* has $n+1$ nodes, and the G_l s are the $\mathcal{O}(\lg n)$ structures of type $G^{(d-1)}$ referred to in the statement. As all the structures T_l occupy $n + \mathcal{O}(n)$ words, $G^{(d)}$ occupies $\mathcal{O}(s(n) \lg n + n)$ words.

Next we show how to use $G^{(d)}$ to answer queries. Let $P_{x,y}$ be the query path and $Q = \prod_{j=1}^d [q_j, q'_j]$ be the query range. To answer the query, we first decompose $P_{x,y}$ into $A_{x,z}$, $\{z\}$, and $A_{y,z}$, where z is the lowest common ancestor of x and y , found in $\mathcal{O}(1)$ time via LCA in T_1 . It suffices to answer three path semigroup sum queries using each subpath and Q as query parameters, as the semigroup sum of the answers to these queries is the answer to the original query. Since the query on subpath $\{z\}$ reduces to checking whether $\mathbf{w}(z) \in Q$, we show how to answer the query on $A_{x,z}$; the query on $A_{y,z}$ is then handled similarly. To answer the query on $A_{x,z}$, we perform a standard top-down traversal in the range tree to identify up to two nodes at each level representing ranges that contain exactly one of q_d or q'_d . Let, thus, v be the node that we are visiting, in the range tree R . We maintain *current nodes*, x_v and z_v (initialized as respectively x and z) local to the current level l ; they are the nodes in T_l that correspond to \mathcal{T}_v -view of the original query nodes x and z . Nodes x_v and z_v are kept up-to-date in $\mathcal{O}(1)$ time as we descend the levels of the range tree. Namely, when descending to the j th ($j \in \{0, 1\}$) child of the node v , we identify, via Lemma 2, the corresponding nodes in T_{l+1} , for nodes $\text{level_anc}_j(T_l, x_v, 1)$ and $\text{level_anc}_j(T_l, z_v, 1)$.

For each node v identified at each level l , such that v 's range contains q_d but not q'_d , we check if it is its left child-range that contains q_d . If so, we perform a $(d-1)$ -dimensional semigroup range sum query with the following parameters: (i) the query range $[q_1, q'_1] \times [q_2, q'_2] \times \dots \times [q_{d-1}, q'_{d-1}]$ (i.e. we drop the last range); and (ii) the query path is A_{x_u, z_u} , where x_u and z_u are the nodes in T_{l+1} corresponding to the \mathcal{T}_u -views of x and z , with u being the right child of v ; this is analogous to updating x_v and z_v , i.e. applying Lemma 2 to nodes $\text{level_anc}_1(T_l, x_v, 1)$, $\text{level_anc}_1(T_l, z_v, 1)$. For each node whose range contains q'_d but not q_d , a symmetrical procedure is performed by considering its left child.

The semigroup sum of the answers to these $\mathcal{O}(\lg n)$ queries is the answer to the original query. \blacktriangleleft

4 Space Reduction Lemma for Non-Constant Branching Factor

This section presents a general framework for reducing the problem of answering a (d', d, ϵ) -dimensional query to the same query problem in $(d' - 1, d, \epsilon)$ dimensions, by generalizing the approach of [16] for the case of trees weighted with multidimensional vectors.

► **Lemma 6.** *Let d and d' be positive integer constants such that $d' \leq d$, and ϵ be a constant in $(0, 1)$. Let $G^{(d'-1)}$ be an $s(n)$ -word data structure for a $(d' - 1, d, \epsilon)$ -dimensional semigroup path sum problem of size n . Then, there is an $\mathcal{O}(s(n) \lg n / \lg \lg n + n)$ -word data structure $G^{(d')}$ for a (d', d, ϵ) -dimensional semigroup path sum problem of size n , whose components include $\mathcal{O}(\lg n / \lg \lg n)$ structures of type $G^{(d'-1)}$, each of which is constructed over a tree on $n+1$ nodes. Furthermore, $G^{(d')}$ can answer a (d', d, ϵ) -dimensional semigroup path sum query by performing $\mathcal{O}(\lg n / \lg \lg n)$ $(d' - 1, d, \epsilon)$ -dimensional queries using these components and returning the semigroup sum of the answers. Determining which queries to perform on structures of type $G^{(d'-1)}$ requires $\mathcal{O}(1)$ time per query.*

Proof. We define a conceptual range tree over the d' th weights of the nodes of T and represent it using hierarchical tree extraction as in Section 2. For each level l of the range tree, we define a tree T_l^* with the same topology as T_l . We assign $(d' - 1, d, \epsilon)$ -dimensional weight vectors and semigroup elements to each node, x' , in T_l^* , as follows. If x' is not the dummy

root, then $\mathbf{w}(x')$ is set to be $(w_1(x), \dots, w_{d'-1}(x), \lambda(T_l, x'), w_{d'+1}(x), \dots, w_d(x))$, where x is the corresponding node of x' in T , and $\lambda(T_l, x')$ is the label assigned to x' in T_l . We also set $g(x') = g(x)$. If x' is the dummy root, then its first $d' - 1$ weights are $-\infty$ and last $d - d' + 1$ weights are $-\lceil \lg^\epsilon \rceil$, while $g(x')$ is set to an arbitrary element of the semigroup. We further construct a data structure, G_l , of type $G^{(d'-1)}$, over T_l^* . The data structure $G^{(d')}$ then comprises the structures T_l and G_l , over all l . The range tree has $\mathcal{O}(\lg n / \lg \lg n)$ levels and each T_l^* has $n + 1$ nodes, and the structures G_l are the $\mathcal{O}(\lg n / \lg \lg n)$ structures of type $G^{(d'-1)}$ referred to in the statement. As all the T_l s occupy $n + \mathcal{O}(n)$ words, $G^{(d')}$ occupies $\mathcal{O}(s(n) \lg n / \lg \lg n + n)$ words.

Next we show how to use $G^{(d')}$ to answer queries. Let $P_{x,y}$ be the query path and $Q = \prod_{j=1}^d [q_j, q'_j]$ be the query range. As discussed in the proof of Lemma 5, it suffices to describe the handling of the path $A_{x,z}$, where z is the lowest common ancestor of x and y .

To answer the query on $A_{x,z}$, we perform a top-down traversal in the range tree to identify the up to two nodes at each level representing ranges that contain at least one of $q_{d'}$ and $q'_{d'}$. For each node v identified at each level l , we perform a $(d' - 1, d, \epsilon)$ -dimensional semigroup range sum query with parameters computed as follows: (i) the query path is P_{x_v, z_v} , where x_v and z_v are the nodes in T_l corresponding to the \mathcal{S}_v -views of x and z ; and (ii) the query range is $Q_v = [q_1, q'_1] \times [q_2, q'_2] \times \dots \times [q_{d'-1}, q'_{d'-1}] \times [i_v..j_v] \times [q_{d'+1}, q'_{d'+1}] \times \dots \times [q_d, q'_d]$, such that the children of v representing ranges that are entirely within $[q_{d'}, q'_{d'}]$ are children $i_v, i_v + 1, \dots, j_v$ (child i refers to the i th child); no queries are performed if such children do not exist. The semigroup sum of these $\mathcal{O}(\lg n / \lg \lg n)$ queries is the answer to the original query. It remains to show that the parameters of each query are computed in $\mathcal{O}(1)$ time per query. By Section 2, i_v and j_v are computed in $\mathcal{O}(1)$ time via simple arithmetic, which is sufficient to determine Q_v . Nodes x_v and z_v are computed in $\mathcal{O}(1)$ time each time we descend down a level in the range tree: Initially, when v is the root of the range tree, x_v and z_v are nodes x and z in T_1 . When we visit a child, v_j , of v whose range contains at least one of $q_{d'}$ and $q'_{d'}$, we compute (via Lemma 2) x_{v_j} as the node in T_{l+1} corresponding to the node $\text{level_anc}_j(T_l, x_v, 1)$ in T_l , which uses constant time. Node z_{v_j} is located similarly. ◀

5 Ancestor Dominance Reporting

In Lemma 7 we solve the $(1, d, \epsilon)$ -dimensional *path dominance reporting problem*, which asks one to enumerate the nodes in the query path whose weight vectors dominate the query vector. The strategy employed in Lemma 7 is that of zooming into the extraction dominating the query point in the last $(d - 1)$ weights, and therein reporting the relevant nodes based on the 1st weight and tree topology only.

► **Lemma 7.** *Let $d \geq 1$ be a constant integer and $0 < \epsilon < \frac{1}{d-1}$ be a constant number. A tree T on $m \leq n$ nodes, in which each node is assigned a $(1, d, \epsilon)$ -dimensional weight vector, can be represented in $m + \mathcal{O}(m)$ words, so that a path dominance reporting query can be answered in $\mathcal{O}(1 + k)$ time, where k is the number of the nodes reported.*

Proof. We represent T using Lemma 1. For any $(0, d - 1, \epsilon)$ -dimensional vector $\mathbf{g} = (g_1, g_2, \dots, g_{d-1})$, we consider a conceptual scalarly-weighted tree $E_{\mathbf{g}}$ by first extracting the node set $G = \{x \mid x \in T \text{ and } \mathbf{w}_{2,d}(x) \succeq \mathbf{g}\}$ from T . The weight of a non-dummy node in $E_{\mathbf{g}}$ is the 1st weight of its T -source. If $E_{\mathbf{g}}$ has a dummy root, then its weight is $-\infty$.

Instead of storing $E_{\mathbf{g}}$ explicitly, we create the following structures, the first two of which are built for any possible $(0, d - 1, \epsilon)$ -dimensional vector \mathbf{g} :

- A 0/1-labeled tree $T_{\mathbf{g}}$ (using Lemma 1) with the topology of T , in which a node u has label 1 iff u is extracted when constructing $E_{\mathbf{g}}$;
- A succinct index $I_{\mathbf{g}}$ for path maximum queries in $E_{\mathbf{g}}$ (using Lemma 4(a));
- An array W_1 where $W_1[x]$ stores the 1st weight of the node x in T ;
- A table C which stores pointers to $T_{\mathbf{g}}$ and $I_{\mathbf{g}}$ for each possible \mathbf{g} .

For any node x' in $E_{\mathbf{g}}$, its T -source x can be computed using $x = \text{pre_select}_1(T_{\mathbf{g}}, x')$. Then, the weight of x' is $W_1[x]$. With this $\mathcal{O}(1)$ -time access to node weights in $E_{\mathbf{g}}$, by Lemma 4 we can use $I_{\mathbf{g}}$ to answer path maximum queries in $E_{\mathbf{g}}$ in $\mathcal{O}(1)$ time.

We now show how to answer a path dominance reporting query in T . Let $P_{x,y}$ and $\mathbf{q} = (q_1, q_2, \dots, q_d)$ be respectively the path and weight vector given as query parameters. First, we use C to locate $T_{\mathbf{q}'}$ and $I_{\mathbf{q}'}$, where $\mathbf{q}' = \mathbf{q}_{2,d}$. As discussed in the proof of Lemma 5, it suffices to show how to answer the query with $A_{x,z}$ as the query path, where $z = \text{LCA}(T, x, y)$. To that end, we fetch the $T_{\mathbf{q}'}$ -view, x' , of x , as $x' = \text{pre_rank}_1(T_{\mathbf{q}'}, \text{level_anc}_1(T_{\mathbf{q}'}, x, 1))$, and analogously the view, z' , of z . Next, $I_{\mathbf{q}'}$ locates a node $t' \in A_{x',z'}$ with the maximum weight. If the weight of t' is less than q_1 , then no node in $A_{x,y}$ can possibly have a weight vector dominating \mathbf{q} , and our algorithm is terminated without reporting any nodes. Otherwise, the T -source t of t' is located as $t = \text{pre_select}_1(T_{\mathbf{q}'}, t')$. The node $t \in T$ then claims the following two properties: (i) as $T_{\mathbf{q}'}$ contains a node corresponding to t , one has $\mathbf{w}_{2,d}(t) \succeq \mathbf{q}'$; and (ii) as $w_1(t)$ equals the weight of t' , it is at least q_1 . We therefore have that $\mathbf{w}(t) \succeq \mathbf{q}$ and hence report t . Afterwards, we perform the same procedure recursively on paths $A_{x',t'}$ and $A_{s',z'}$ in $E_{\mathbf{q}'}$, where $s' = \text{pre_rank}_1(T_{\mathbf{q}'}, \text{level_anc}_1(T_{\mathbf{q}'}, t, 1))$.

To analyze the running time, the key observation is that we perform path maximum queries using $I_{\mathbf{q}'}$ at most $2k + 1$ times. Since both each query itself and the operations performed to identify the query path use $\mathcal{O}(1)$ time, our algorithm runs in $\mathcal{O}(1 + k)$ time.

To analyze the space cost, we observe that W_1 occupies m words. The total number of possible $(0, d-1, \epsilon)$ -dimensional vectors is $\mathcal{O}(\lg^{(d-1)\epsilon} n)$. Since each $T_{\mathbf{g}}$ uses $\mathcal{O}(m)$ bits and each $I_{\mathbf{g}}$ uses $\mathcal{O}(m \lg^{**} m)$ bits, the total space cost of storing $T_{\mathbf{g}}$'s and $I_{\mathbf{g}}$'s for all possible \mathbf{g} 's is $\mathcal{O}((m + m \lg^{**} m) \lg^{(d-1)\epsilon} n) = \mathcal{O}(m \lg^{**} m \lg^{(d-1)\epsilon} n) \leq \mathcal{O}(m \lg^{**} n \lg^{(d-1)\epsilon} n) = \mathcal{O}(m \lg n)$ bits for any constant $0 < \epsilon < 1/(d-1)$, which is $\mathcal{O}(m)$ words. Furthermore, C stores $\mathcal{O}(\lg^{(d-1)\epsilon} n)$ pointers. To save the space cost of each pointer, we concatenate the encodings of all the $T_{\mathbf{g}}$ s and $I_{\mathbf{g}}$ s and store them in a memory block of $\mathcal{O}(m \lg n)$ bits. Thus, each pointer stored in C can be encoded in $\mathcal{O}(\lg(m \lg n))$ bits, and the table C thus uses $\mathcal{O}((\lg m + \lg \lg n) \lg^{(d-1)\epsilon} n) = \mathcal{O}(\lg m \lg^{(d-1)\epsilon} n) + \mathcal{O}(\lg \lg n \lg^{(d-1)\epsilon} n) = \mathcal{O}(\lg m \lg n) + \mathcal{O}(\lg n) = \mathcal{O}(\lg m \lg n)$ bits for any constant $0 < \epsilon < 1/(d-1)$, which is $\mathcal{O}(\lg m)$ words. Finally, the encoding of T using Lemma 1 is $2m + \mathcal{O}(m)$ bits. Therefore, the total space cost is $m + \mathcal{O}(m)$ words. ◀

We next design a solution to the 2-dimensional ancestor dominance reporting problem, by first generalizing the notion of 2-dominance in Euclidean space to weighted trees. More precisely, in a tree T in which each node is assigned a d -dimensional weight vector, we say that a node x 2-dominates another node y iff $x \in \mathcal{A}(y)$ and $w_1(x) \geq w_1(y)$. Then a node x is defined to be 2-maximal iff no other node in T 2-dominates x .

The following property is then immediate: Given a set, X , of 2-maximal nodes, let T_X be the corresponding extraction from T . Let the weight of a node $x' \in T_X$ be the 1st weight of its T -source x . Then, in any upward path of T_X , the node weights are strictly decreasing. In such a tree as T_X , the *weighted ancestor problem* [9] is defined. In this problem, one is given a weighted tree with monotonically decreasing node weights along any upward path. We preprocess such a tree to answer *weighted ancestor queries*, which, for any given node

x and value κ , ask for the highest ancestor of x whose weight is at least κ . Farach and Muthukrishnan [9] presented an $\mathcal{O}(n)$ -word solution that answers this query in $\mathcal{O}(\lg \lg n)$ time, for an n -node tree weighted over $[n]$. With an easy reduction we can further achieve the following result:

► **Lemma 8.** *Let T be a tree on $m \leq n$ nodes, in which each node is assigned a weight from $[n]$. If the node weights along any upward path are strictly decreasing, then T can be represented using $\mathcal{O}(m)$ words to support weighted ancestor queries in $\mathcal{O}(\lg \lg n)$ time.*

Proof. Let W be the set of weights actually assigned to the nodes of T . We replace the weight, h , of any node x in T by the rank of h in W , which is in $[m]$. We then represent the resulting tree T' in $\mathcal{O}(m)$ words to support a weighted ancestor query in T' in $\mathcal{O}(\lg \lg m)$ time [9]. We also construct a y -fast trie [22], Y , on the elements of W ; the rank of each element is also stored with this element in Y . Y uses $\mathcal{O}(m)$ space. Given a weighted ancestor query over T , we first find the rank, κ , of the query weight in W in $\mathcal{O}(\lg \lg n)$ time by performing a predecessor query in Y , and κ is further used to perform a query in T' to compute the answer. ◀

To design our data structures, we define a conceptual range tree with branching factor $f = \lceil \lg^\epsilon n \rceil$ over the 2nd weights of the nodes in T and represent it using hierarchical tree extraction as in Section 2. Let v be a node in this range tree. In \mathcal{T}_v , we assign to each node the weight vector of the T -source and call the resulting weighted tree $T(v)$. We then define $M(v)$ as follows: If v is the root of the range tree, then $M(v)$ is the set of all the 2-maximal nodes in T . Otherwise, let u be the parent of v . Then a node, t , of $T(v)$ is in $M(v)$ iff t is 2-maximal in $T(v)$ and its corresponding node in $T(u)$ is not 2-maximal in $T(u)$. Thus, for any node x in T , there exists a unique node v in the range tree such that there is a node in $M(v)$ corresponding to x .

We further conceptually extract two trees from T_l : (i) M_l is an extraction from T_l of the node set $\{x \mid x \in T_l \text{ and there exists a node } u \text{ at level } l \text{ of the range tree, s.t. } x \text{ has a corresponding node in } M(u)\}$; while (ii) N_l is an extraction from T_l of the node set $\{x \mid x \in T_l \text{ and } \exists \text{ a node } v \text{ at level } l+1 \text{ of the range tree, s.t. } x \text{ has a corresponding node } \in M(v)\}$. $T_M(v)$ is the tree formed by extracting $M(v)$ from $T(v)$, and $T_G(v)$ is the tree formed by extracting from $T(v)$ the node set $\{x \mid x \in T(v) \text{ and there exists a child, } t \text{ of } v \text{ s.t. there is a node in } M(t) \text{ corresponding to } x\}$. Then, for each level l , we also create the following data structures (when defining these structures, we assume that the root, r_l , of T_l corresponds to a dummy node η in T with weight vector $(-\infty, -\infty)$; the node η is omitted when determining the rank space, preorder ranks, and depths in T):

- D_l , a 1-dimensional path dominance reporting structure (using Lemma 7) over the tree obtained by assigning weight vectors to the nodes of M_l as follows: each node x' of M_l is assigned a scalar weight $w_2(x)$, where x is the node of T corresponding to x' ;
- E_l , a 1-dimensional path dominance reporting structure (using Lemma 7) over the tree obtained by assigning weight vectors to the nodes of M_l as follows: each node x' of M_l is assigned a scalar weight $w_1(x)$, where x is the node of T corresponding to x' ;
- F_l , a $(1, 2, \epsilon)$ -dimensional path dominance reporting structure (using Lemma 7) over the tree obtained by assigning weight vectors to the nodes of N_l as follows: each node x' of N_l is assigned $(w_1(x), \kappa)$, where x is the node of T corresponding to x' , and κ is the label assigned to the node in T_l corresponding to x' ;
- A_l , a weighted ancestor query structure over M_l (using Lemma 8), when its nodes are assigned the 1st weights of the corresponding nodes in T ;

- T'_l , a 0/1-labeled tree (using Lemma 1) with the topology of T_l , and a node is assigned 1 *iff* it is extracted when constructing M_l ;
- T''_l , a 0/1-labeled tree (using Lemma 1) with the topology of T_l , and a node is assigned 1 *iff* it is extracted when constructing N_l ;
- P_l , an array where $P_l[x]$ stores the preorder number of the node in T corresponding to a node x in M_l .

We now describe the algorithm for answering queries, and analyze its running time and space cost:

► **Lemma 9.** *A tree T on n nodes, in which each node is assigned a 2-dimensional weight vector, can be represented in $\mathcal{O}(n)$ words, so that an ancestor dominance reporting query can be answered in $\mathcal{O}(\lg n + k)$ time, where k is the number of the nodes reported.*

Proof. Let x and $\mathbf{q} = (q_1, q_2)$ be the node and weight vector given as query parameters, respectively. We define Π as the path in the range tree between and including the root and the leaf storing q_2 . Let π_l denote the node at level l in this path. Then the root of the range tree is π_1 . To answer the query, we perform a traversal of a subset of the nodes of the range tree, starting from π_1 . The invariant maintained during this traversal is that a node u of the range tree is visited *iff* one of the following two conditions holds: (i) $u = \pi_l$ for some l ; or (ii) $M(u)$ contains at least one node whose corresponding node in T must be reported. We now describe how the algorithm works when visiting a node, v , at level l of this range tree, during which we will show how the invariant is maintained. Let x_v denote the node in T_l that corresponds to the \mathcal{F}_v -view of x ; x_v can be located in constant time each time we descend down one level in the range tree, as described in the proof of Lemma 5. Our first step is to report all the nodes in the answer to the query that have corresponding nodes in $M(v)$. There are two cases depending on whether $v = \pi_l$; this condition can be checked in constant time by determining whether q_2 belongs to the range represented by v . In either of these cases, we first locate the M_l -view, x'_v , of x_v by computing $x'_v = \text{pre_rank}_1(T'_l, \text{level_anc}_1(T'_l, x_v, 1))$.

If (i) holds, then the non-dummy ancestors of x'_v in M_l correspond to all the ancestors of x in T that have corresponding nodes in $M(v)$. We then perform a weighted ancestor query using A_l to locate the highest ancestor, y , of x'_v in M_l whose 1st weight is at least q_1 . Since the 1st weights of the nodes along any upward path in M_l are decreasing, the 1st weights of the nodes in path $P_{x'_v, y}$ are greater than or equal to q_1 , while those of the proper ancestors of y are strictly less. Hence, by performing a 1-dimensional path dominance reporting query in D_l using $P_{x'_v, y}$ as the query path and $\mathbf{q}' = (q_2)$ as the query weight vector, we can find all the ancestors of x'_v whose corresponding nodes in T have weight vectors dominating \mathbf{q} . Then, for each of these nodes, we retrieve from P_l its corresponding node in T which is further reported.

If $v \neq \pi_l$, the maintained invariant guarantees that the 2nd weights of the nodes in $M(v)$ are greater than q_2 . Therefore, by performing a 1-dimensional path dominance reporting query in $E_l(s)$ using the path between (inclusive) x'_v and the root of M_l as the query path and $\mathbf{q}'' = (q_1)$ as the query weight vector, we can find all the ancestors of x'_v in M_l whose corresponding nodes in T have weight vectors dominating \mathbf{q} . By mapping these nodes to nodes in T via P_l , we have reported all the nodes in the answer to the query that have corresponding nodes in $M(v)$.

After we handle both cases, the next task is to decide which children of v we should visit. Let v_i denote the i th child of v . We always visit π_{l+1} if it happens to be a child of v . To maintain the invariant, for any other child v_i , we visit it *iff* there exists at least one node in $M(v_i)$

whose corresponding node in T should be reported. To find the children that we will visit, we locate the N_l -view, x'_v , of x_v by computing $x'_v = \text{pre_rank}_1(T'_l, \text{level_anc}_1(T'_l, x_v, 1))$. Then the non-dummy ancestors of x'_v correspond to all the ancestors of x in T that have corresponding nodes in $\cup_{i=1,2,\dots} M(v_i)$. We then perform a $(1, 2, \epsilon)$ -dimensional path dominance reporting query in F_l using the path between (inclusive) x'_v and the root of N_l as the query path and $(q_1, \kappa + 1)$ as the query weight vector if π_{l+1} is the κ th child of v , and we set $\kappa = 0$ if π_{l+1} is not a child of v . For each node, t , returned when answering this query, if its 2nd weight in F_l is j , then t corresponds to a node in $M(v_j)$. Since the node corresponding to t in T should be included in the answer to the original query, we iteratively visit v_j if we have not visited it before (checked e.g. using an f -bit word to flag the children of v).

The total query time is dominated by the time used to perform queries using A_l , D_l , E_l and F_l . We only perform one weighted ancestor query when visiting each π_l , and this query is not performed when visiting other nodes of the range tree. Given the $\mathcal{O}(\lg n / \lg \lg n)$ levels of the range tree, all the weighted ancestor queries collectively use $\mathcal{O}(\lg \lg n \times (\lg n / \lg \lg n)) = \mathcal{O}(\lg n)$ time. Similarly, we perform one query using D_l at each level of the range tree, and the query times summed over all levels is $\mathcal{O}(\lg n / \lg \lg n + k)$. Our algorithm guarantees that, each time we perform a query using E_l , we report a not-reported hitherto, non-empty subset of the nodes in the answer to the original query. Therefore, the queries performed over all E_l 's use $\mathcal{O}(k)$ time in total. Querying the F_l -structures incurs $\mathcal{O}(k)$ time cost when visiting nodes not in Π , and $\mathcal{O}(\lg n / \lg \lg n + k)$ time when visiting nodes in Π . Thus, the query times spent on all these structures throughout the execution of the algorithm sum up to $\mathcal{O}(\lg n + k)$.

We next analyze space cost of our data structures. As mentioned in Section 2, all the T_l s occupy $n + \mathcal{O}(n)$ words. By Lemma 1, each T'_l or T''_l uses $3n + \mathcal{O}(n)$ bits, so over all $\lg n / \lg \lg n$ levels, they occupy $\mathcal{O}(n \lg n / \lg \lg n)$ bits, which is $\mathcal{O}(n / \lg \lg n)$ words. As discussed earlier, we know that, for any node x in T , there exists one and only one node v in the range tree such that there is a node in $M(v)$ corresponding to x . Furthermore, $M(v)$ s only contain nodes that have corresponding nodes in T . Therefore, the sum of the sizes of all $M(v)$ s is exactly n . Hence all the P_l 's have n entries in total and thus uses n words. By Lemma 7, the size of each D_l in words is linear in the number of nodes in M_l . The sum of the numbers of nodes in M_l s over all levels of the range tree is equal to the sum of the sizes of all $M(v)$ s plus the number of dummy roots, which is $n + \mathcal{O}(\lg n / \lg \lg n)$. Therefore, all the D_l s occupy $\mathcal{O}(n)$ words. By similar reasoning, all the E_l s and A_l s occupy $\mathcal{O}(n)$ words in total. Finally, it is also true that, for any node x in T , there exists a unique node v in the range tree such that there is a node in $N(v)$ corresponding to x . Thus, we can upper-bound the total space cost of all the F_l s by $\mathcal{O}(n)$ words in a similar way. All our data structures, therefore, use $\mathcal{O}(n)$ words. \blacktriangleleft

Further, we describe the data structure for $(2, d, \epsilon)$ -dimensional ancestor dominance reporting, and analyze its time- and space-bounds:

► Lemma 10. *Let $d \geq 2$ be a constant integer and $0 < \epsilon < \frac{1}{d-2}$ be a constant number. A tree T on n nodes, in which each node is assigned a $(2, d, \epsilon)$ -dimensional weight vector, can be represented in $\mathcal{O}(n \lg^{(d-2)\epsilon} n)$ words, so that an ancestor dominance reporting query can be answered in $\mathcal{O}(\lg n + k)$ time, where k is the number of the nodes reported.*

Proof. In our design, for any $(0, d - 2, \epsilon)$ -dimensional vector \mathbf{g} , we consider a conceptual scalarly-weighted tree $E_{\mathbf{g}}$ as the tree extraction from T of the node set $\{x \mid x \in T \text{ and } \mathbf{w}_{3,d}(x) \succeq \mathbf{g}\}$. The weight of a node x' in $E_{\mathbf{g}}$ is the 2-dimensional weight vector $\mathbf{w}_{1,2}(x)$, where x the T -source of x' . If $E_{\mathbf{g}}$ has a dummy root, then its weight is $(-\infty, -\infty)$. Rather than storing $E_{\mathbf{g}}$ explicitly, we follow the strategy in the proof of Lemma 7 and store

a 0/1-labeled tree $T_{\mathbf{g}}$ for each possible \mathbf{g} . $T_{\mathbf{g}}$ is obtained from T by assigning 1-labels to the nodes of T extracted when constructing $E_{\mathbf{g}}$. We also maintain arrays W_1 and W_2 storing respectively the 1st and 2nd weights of all nodes of T , in preorder, which enables accessing the weight of an arbitrary node of $E_{\mathbf{g}}$ in $\mathcal{O}(1)$ time. Let $n_{\mathbf{g}}$ be the number of nodes in $E_{\mathbf{g}}$. We convert the node weights of each $E_{\mathbf{g}}$ to rank space $[n_{\mathbf{g}}]$. For each such $E_{\mathbf{g}}$, we build the 2-dimensional ancestor dominance reporting data structure, $V_{\mathbf{g}}$, from Lemma 9. Thus, the space usage of the resulting data structure is upper-bounded by $\mathcal{O}(n \lg^{(d-2)\epsilon} n)$ words.

Let x and $\mathbf{q} = (q_1, q_2, \dots, q_d)$ be the node and weight vector given as query parameters, respectively. In $\mathcal{O}(1)$ time, we fetch the data structures pertaining to the range $\mathbf{q}' = \mathbf{q}_{3,d}$; this way, all the weights 3 through d of the query vector have been taken care of, and all we need to consider is the tree topology and the first two weights, q_1 and q_2 , of the original query vector. We localize the query node x to $E_{\mathbf{q}'}$ via $x' = \text{pre_rank}_1(T_{\mathbf{q}'}, \text{level_anc}_1(T_{\mathbf{q}'}, x, 1))$, and launch the query in $V_{\mathbf{q}'}$ with x' as a query node, having reduced the components of the query vector (q_1, q_2) to the rank space of $E_{\mathbf{q}'}$ (the time- and space-bounds for the reductions are absorbed in the final bounds). ◀

Instantiating Section 3 with $g(x) = \{x\}$ and the semigroup sum operator \oplus as the set-theoretic union operator \cup , Lemma 5 iteratively applied to Lemma 9 yields

► **Theorem 11.** *Let $d \geq 2$ be a constant integer. A tree T on n nodes, in which each node is assigned a d -dimensional weight vector, can be represented in $\mathcal{O}(n \lg^{d-2} n)$ words, so that an ancestor dominance reporting query can be answered in $\mathcal{O}(\lg^{d-1} n + k)$ time, where k is the number of the nodes reported.*

Analogously, Lemma 6 (Section 4) that is the counterpart of Lemma 5 when the range tree has a non-constant branching factor $f = \mathcal{O}(\lg^{\epsilon} n)$, with Lemma 10 which addresses $(2, d, \epsilon)$ -dimensional ancestor reporting, together yield a different tradeoff:

► **Theorem 12.** *Let $d \geq 3$ be a constant integer. A tree T on n nodes, in which each node is assigned a d -dimensional weight vector, can be represented in $\mathcal{O}(n \lg^{d-2+\epsilon} n)$ words of space, so that an ancestor dominance reporting query can be answered in $\mathcal{O}((\lg^{d-1} n)/(\lg \lg n)^{d-2} + k)$ time, where k is the number of the nodes reported. Here, $\epsilon \in (0, 1)$ is a constant.*

6 Path Successor

We first solve the path successor problem when $d = 1$, and extend the result to $d > 1$ via Lemma 5.

The topology of T is stored using Lemma 1. We define a binary range tree R over $[n]$, and build the associated hierarchical tree extraction as in Section 2; T_l denotes the auxiliary tree built for each level l of R , and \mathcal{T}_v denotes the tree extraction from T associated with the range of node $v \in R$. We represent R using Lemma 1, and augment it with the ball-inheritance data structure \mathcal{B} from Lemma 3(a), as well as with the data structure from the following

► **Lemma 13.** *Let R be a binary range tree with topology encoded using Lemma 1, and augmented with ball-inheritance data structure \mathcal{B} from Lemma 3(a). With additional space of $\mathcal{O}(n)$ words, the node $x_{u,l}$ in T_l corresponding to the \mathcal{T}_u -view of x can be found in $\mathcal{O}(\log^{\epsilon'} n)$ time, where ϵ' is an arbitrary constant in $(0, 1)$, for an arbitrary node $x \in T$ and an arbitrary node $u \in R$ residing on a level l .*

Proof. For a node $u \in R$ at a level l , and a node $x \in T$, the query can be thought of as a chain of transformations $T \rightarrow \mathcal{T}_u \rightarrow T_l$. In the first transition, $T \rightarrow \mathcal{T}_u$, given an original node $x \in T$, we are looking for its \mathcal{T}_u -view, x_u . That is, although \mathcal{T}_u is obtained from T through a *series* of extractions, the wish is to “jump” many successive extractions at once, as if \mathcal{T}_u were extracted from T *directly*. This would be trivial to achieve through storing a 0/1-labeled tree per range u , if it were not for prohibitive space-cost – number of bits more than quadratic in the number of nodes. One can avoid extra space cost altogether and use Lemma 2 directly to explicitly descend the hierarchy of extractions. In this case, the time cost is proportional to the height of the range tree, and hence becomes the bottleneck.

In turn, in the $\mathcal{T}_u \rightarrow T_l$ -transition, we are looking for the identity of x_u in T_l . For this second transformation, we recall (from Section 2) that \mathcal{T}_u is embedded within T_l . Moreover, the nodes of \mathcal{T}_u must lie contiguously in the preorder sequence of T_l .

We overcome these difficulties with the following data structures.

For R , we maintain an annotation array I , such that $I[u]$ stores a quadruple $\langle a_u, b_u, s_u, t_u \rangle$ for an arbitrary node $u \in R$, such that (i) the weight range associated with u is $[a_u, b_u]$; and (ii) all the nodes of T with weights in $[a_u, b_u]$ occupy precisely the preorder ranks s_u through t_u in T_l . The space occupied by the annotation array I , which is $\mathcal{O}(\lg n)$ bits summed over all the $\mathcal{O}(n)$ nodes of R , is $\mathcal{O}(n)$ words.

For each level $L \equiv 0 \pmod{\lceil \lg \lg n \rceil}$, which we call *marked*, we maintain a data structure enabling the direct $T \rightarrow \mathcal{T}_u \rightarrow T_L$ -conversion. Namely, for each individual node u on marked level L of R , we define a conceptual array A_u , which stores, in increasing order, the (original) preorder ranks of all the nodes of T whose weights are in the range represented by u . Rather than maintaining A_u explicitly, we store a succinct index, S_u , for predecessor/successor search [11] in A_u . Assuming the availability of a $\mathcal{O}(n^\delta)$ -bit universal table, where δ is a constant in $(0, 1)$, given an arbitrary value in $[n]$, this index can return the position of its predecessor/successor in A_u in $\mathcal{O}(\lg \lg n)$ time plus accesses to $\mathcal{O}(1)$ entries of A_u . The size of the index in bits is $\mathcal{O}(\lg \lg n)$ times the number of entries in A_u . For a fixed marked level L , therefore, all the S_u -structures sum up to $\mathcal{O}(n \lg \lg n)$ bits. There being $\mathcal{O}(\lg n / \lg \lg n)$ marked levels, the total space cost for the S_u -structures over all the entire tree R is $\mathcal{O}(n)$ words.

We now turn to answering the query using the data structures built. Resolving the query falls into two distinct cases, depending on whether the level l , at which the query node u resides, is marked or not.

When the level l is marked, we use the structures S_u stored therein, directly. We adopt the strategy in [14] to find $x_{u,l}$. First, for an arbitrary index i to A_u , we observe that node $A_u[i] \in T$ corresponds to node $(s_u + i - 1)$ in T_l . We thus fetch $\langle a_u, b_u, s_u, t_u \rangle$ from $I[u]$. Then the predecessor $p \in A_u$ of x is obtained through S_u via an $\mathcal{O}(\lg \lg n)$ query and $\mathcal{O}(1)$ calls to the \mathcal{B} -structure, which totals $\mathcal{O}(\lg^{\epsilon'} n)$ time. We then determine the lowest common ancestor $\chi \in T$ of x and p , in $\mathcal{O}(1)$ time. If the weight of χ is in $[a_u, b_u]$, then it must be present in A_u by the latter’s very definition. By another predecessor query, therefore, we can find the position, j , of χ in A_u , and $(s_u + j - 1)$ is the sought $x_{u,l}$. Otherwise, a final query to S_u returns the successor $\chi' \in A_u$ of χ . Let κ be the position of χ' in A_u . Then the parent of the node $(s_u + \kappa - 1)$ in T_l is $x_{u,l}$. We perform a constant number of predecessor/successor queries, and correspondingly a constant number of calls to the ball-inheritance problem. The time complexity is thus $\mathcal{O}(\lg^{\epsilon'} n)$.

When the level l is not marked, we ascend to the lowest ancestor u' of u residing on a marked level l' , and reduce the problem to the previous case. More precisely, via the navigation operations (`level_anc()`) to move to a parent, and `depth()` to determine the

status of a level) available through R 's encoding, we climb up at most $\lceil \lg \lg n \rceil$ levels to the closest marked level l' . Let u' be therefore the ancestor of u found on that marked level l' . We find the answer to the original query, as if the query node were u' ; that is, we find the node $x'_{u',l'}$ in $T_{l'}$ that corresponds to the $\mathcal{T}_{u'}$ -view of the original query node x in T . Let us initialize a variable χ to be $x_{u',l'}$. We descend down to the original level l , back to the original query node u , all the while adjusting the node χ as we move down a level, analogously to the proof of Lemma 5. As we arrive, in time $\mathcal{O}(\lg \lg n)$, at node u , the variable χ stores the answer, $x_{u,l}$.

In both cases, the term $\mathcal{O}(\lg^{\epsilon'} n)$ dominates the time complexity, as climbing to/from a marked level is an additive term of $\mathcal{O}(\lg \lg n)$. Therefore, a query is answered in $\mathcal{O}(\lg^{\epsilon'} n)$ time. ◀

Finally, each T_l is augmented with succinct indices m_l (resp. M_l) from Lemma 4(b), for path minimum (resp. path maximum) queries. As weights of the nodes of T_l , the weights of their corresponding nodes in T are used.

We now describe the algorithm for answering queries and analyze its running time, as well as give the space cost of the built data structures:

► **Lemma 14.** *A scalarly-weighted tree T on n nodes can be represented in $\mathcal{O}(n)$ words, so that a path successor query is answered in $\mathcal{O}(\lg^{\epsilon} n)$ time, where $\epsilon \in (0, 1)$ is a constant.*

Proof. Let x, y and $Q = [q_1, q'_1]$ be respectively the nodes and the orthogonal range given as query's parameters. Appealing to the proof of Lemma 5, we focus only on the path $A_{x,z}$, where z is $\text{LCA}(T, x, y)$. We locate in $\mathcal{O}(1)$ time the leaf L_{q_1} of R that corresponds to the singleton range $[q_1, q_1]$. Let Π be the root-to-leaf path to L_{q_1} in R ; let π_l be the node at level l of Π . We binary search in Π for the deepest node $\pi_f \in \Pi$ whose associated extraction \mathcal{T}_{π_f} contains the node corresponding to the answer to the given query.

We initialize two variables: *high* as 1 so that π_{high} is the root of R , and *low* as the height of R so that π_{low} is the leaf L_{q_1} . We first check if $\mathcal{T}_{\pi_{\text{low}}}$ already contains the answer, by fetching the node x' in T_{low} corresponding to the $\mathcal{T}_{\pi_{\text{low}}}$ -view of x , using Lemma 13. If x' exists, we examine its corresponding node x'' in T (fetched via \mathcal{B}) to see whether x'' is on $A_{x,z}$, by performing `depth` and `level_anc` operations in R ; if it is, x'' is the final answer. If not, this establishes the invariant of the ensuing search: $\mathcal{T}_{\pi_{\text{high}}}$ contains a node corresponding to the answer, whereas $\mathcal{T}_{\pi_{\text{low}}}$ does not.

At each iteration, therefore, we set (via `level_anc` in R) π_{mid} to be the node mid-way from π_{low} to π_{high} . We then fetch the nodes x', z' in T_{mid} corresponding to the $\mathcal{T}_{\pi_{\text{mid}}}$ -views of respectively x and z , using Lemma 13. The non-existence of x' or the emptiness of $A_{x',z'}$ sets *low* to *mid*, and the next iteration of the search ensues. If z' does not exist, z' is set to the root of T_{mid} . A query to the M_{mid} -structure then locates a node in $A_{x',z'}$ for which the 1st weight, μ , of its corresponding node in T is maximized. Accounting for the mapping of a node in T_{mid} to its corresponding node in T via \mathcal{B} , this query uses $\mathcal{O}((\lg^{\epsilon'} n)\alpha(n))$ time. The variables are then updated as *high* \leftarrow *mid* if $\mu \geq q_1$, and *low* \leftarrow *mid*, otherwise.

Once π_f is located, it must hold for π_f that (i) it is its left child that is on Π [20]; and (ii) its right child, v , contains the query result, even though v represents a range of values all larger than q_1 . When locating π_f , we also found the nodes in T_f corresponding to the \mathcal{T}_{π_f} -views of x and z ; they can be further used to find the nodes in T_{f+1} , x^* and z^* , corresponding to the \mathcal{T}_v -views of x and z . We then use m_{f+1} to find the node in A_{x^*,z^*} with minimum 1st weight, whose corresponding node in T is the answer.

The total query time is dominated by that needed for binary search. Each iteration of the search is in turn dominated by the path maximum query in T_{mid} , which is $\mathcal{O}((\lg^{\epsilon'} n)\alpha(n))$. Given the $\mathcal{O}(\lg n)$ levels of R , the binary search has $\mathcal{O}(\lg \lg n)$ iterations. Therefore, the total running time is $\mathcal{O}(\lg \lg n \cdot \lg^{\epsilon'} n \cdot \alpha(n)) = \mathcal{O}(\lg^{\epsilon} n)$ if we choose $\epsilon' < \epsilon$.

To analyze the space cost, we observe that the topology of T , represented using Lemma 1, uses only $2n + \mathcal{O}(n)$ bits. As mentioned in Section 2, all the structures T_l occupy $\mathcal{O}(n)$ words. The space cost of the structure from Lemma 13 built for R is $\mathcal{O}(n)$ words. The \mathcal{B} -structure occupies another $\mathcal{O}(n)$ words. The m_l - and M_l -structure occupy $\mathcal{O}(n)$ bits each, or $\mathcal{O}(n)$ words in total over all levels of R . Thus, the final space cost is $\mathcal{O}(n)$ words. ◀

Lemmas 5 and 14 yield the following

► **Theorem 15.** *Let $d \geq 1$ be a constant integer. A tree T on n nodes, in which each node is assigned a d -dimensional weight vector can be represented in $\mathcal{O}(n \lg^{d-1} n)$ words, so that a path successor query can be answered in $\mathcal{O}(\lg^{d-1+\epsilon} n)$ time, for an arbitrarily small positive constant ϵ .*

Proof. We instantiate Section 3 with $g(x) = x$ and the semigroup sum operator \oplus as $x \oplus y = \operatorname{argmin}_{\zeta=x,y} \{w_1(\zeta)\}$. Lemma 5 applied to Lemma 14 yields the space bound of $\mathcal{O}(n \lg^{d-1} n)$ words and query time complexity of $\mathcal{O}(\lg^{d-1+\epsilon} n)$. ◀

References

- 1 Peyman Afshani. On Dominance Reporting in 3D. In *ESA*, pages 41–51, 2008.
- 2 Noga Alon and Baruch Schieber. Optimal preprocessing for answering on-line product queries. Technical report, Tel-Aviv University, 1987.
- 3 Timothy M. Chan. Persistent Predecessor Search and Orthogonal Point Location on the Word RAM. In *SODA*, pages 1131–1145, 2011.
- 4 Timothy M. Chan, Meng He, J. Ian Munro, and Gelin Zhou. Succinct Indices for Path Minimum, with Applications. *Algorithmica*, 78(2):453–491, 2017.
- 5 Timothy M. Chan, Kasper Green Larsen, and Mihai Patrascu. Orthogonal range searching on the RAM, revisited. In *SoCG*, pages 1–10, 2011.
- 6 Bernard Chazelle. Computing on a free tree via complexity-preserving mappings. *Algorithmica*, 2(1):337–361, 1987.
- 7 Bernard Chazelle and Herbert Edelsbrunner. Linear Space Data Structures for Two Types of Range Search. *Discrete & Computational Geometry*, 2:113–126, 1987.
- 8 Erik D. Demaine, Gad M. Landau, and Oren Weimann. On Cartesian Trees and Range Minimum Queries. *Algorithmica*, 68(3):610–625, 2014.
- 9 Martin Farach and S. Muthukrishnan. Perfect Hashing for Strings: Formalization and Algorithms. In *CPM*, pages 130–140, 1996.
- 10 Harold N. Gabow, Jon Louis Bentley, and Robert E. Tarjan. Scaling and Related Techniques for Geometry Problems. In *STOC*, pages 135–143, 1984.
- 11 Roberto Grossi, Alessio Orlandi, Rajeev Raman, and S. Srinivasa Rao. More Haste, Less Waste: Lowering the Redundancy in Fully Indexable Dictionaries. In *STACS*, pages 517–528, 2009.
- 12 Torben Hagerup. Parallel Preprocessing for Path Queries Without Concurrent Reading. *Inf. Comput.*, 158(1):18–28, 2000.
- 13 Meng He, J. Ian Munro, and Srinivasa Rao Satti. Succinct ordinal trees based on tree covering. *ACM Trans. Algorithms*, 8(4):42:1–42:32, 2012.
- 14 Meng He, J. Ian Munro, and Gelin Zhou. A Framework for Succinct Labeled Ordinal Trees over Large Alphabets. *Algorithmica*, 70(4):696–717, 2014.

- 15 Meng He, J. Ian Munro, and Gelin Zhou. Data Structures for Path Queries. *ACM Trans. Algorithms*, 12(4):53:1–53:32, 2016.
- 16 Joseph JáJá, Christian Worm Mortensen, and Qingmin Shi. Space-Efficient and Fast Algorithms for Multidimensional Dominance Reporting and Counting. In *ISAAC*, pages 558–568, 2004.
- 17 Danny Krizanc, Pat Morin, and Michiel H. M. Smid. Range Mode and Range Median Queries on Lists and Trees. *Nord. J. Comput.*, 12(1):1–17, 2005.
- 18 Christos Makris and Athanasios K. Tsakalidis. Algorithms for Three-Dimensional Dominance Searching in Linear Space. *Inf. Process. Lett.*, 66(6):277–283, 1998.
- 19 Yakov Nekrich. A data structure for multi-dim. range reporting. In *SoCG*, pages 344–353, 2007.
- 20 Yakov Nekrich and Gonzalo Navarro. Sorted Range Reporting. In *SWAT*, pages 271–282, 2012.
- 21 Manish Patil, Rahul Shah, and Sharma V. Thankachan. Succinct representations of weighted trees supporting path queries. *J. Discrete Algorithms*, 17:103–108, 2012.
- 22 Dan E. Willard. Log-Logarithmic Worst-Case Range Queries are Possible in Space $\Theta(N)$. *Inf. Process. Lett.*, 17(2):81–84, 1983.
- 23 Gelin Zhou. Two-dimensional range successor in optimal time and almost linear space. *Inf. Process. Lett.*, 116(2):171–174, 2016.

A $\frac{21}{16}$ -Approximation for the Minimum 3-Path Partition Problem

Yong Chen


Department of Mathematics, Hangzhou Dianzi University, Hangzhou, Zhejiang, China
chenyong@hdu.edu.cn

Randy Goebel

Department of Computing Science, University of Alberta, Edmonton, Alberta T6G 2E8, Canada
rgoebel@ualberta.ca

Bing Su

School of Economics and Management, Xi'an Technological University, Xi'an, Shaanxi, China
subing684@sohu.com

Weitian Tong¹ 

Department of Computer Science, Eastern Michigan University, Ypsilanti, Michigan 48197, USA
wtong.research@gmail.com

Yao Xu

Department of Computer Science, Kettering University, Flint, Michigan 48504, USA
skyaoxu@outlook.com

An Zhang

Department of Mathematics, Hangzhou Dianzi University, Hangzhou, Zhejiang, China
anzhang@hdu.edu.cn

Abstract

The minimum k -path partition (Min- k -PP for short) problem targets to partition an input graph into the smallest number of paths, each of which has order at most k . We focus on the special case when $k = 3$. Existing literature mainly concentrates on the exact algorithms for special graphs, such as trees. Because of the challenge of NP-hardness on general graphs, the approximability of the Min-3-PP problem attracts researchers' attention. The first approximation algorithm dates back about 10 years and achieves an approximation ratio of $\frac{3}{2}$, which was recently improved to $\frac{13}{9}$ and further to $\frac{4}{3}$. We investigate the $\frac{3}{2}$ -approximation algorithm for the Min-3-PP problem and discover several interesting structural properties. Instead of studying the unweighted Min-3-PP problem directly, we design a novel weight schema for ℓ -paths, $\ell \in \{1, 2, 3\}$, and investigate the weighted version. A greedy local search algorithm is proposed to generate a heavy path partition. We show the achieved path partition has the least 1-paths, which is also the key ingredient for the algorithms with ratios $\frac{13}{9}$ and $\frac{4}{3}$. When switching back to the unweighted objective function, we prove the approximation ratio $\frac{21}{16}$ via amortized analysis.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Mathematics of computing → Approximation algorithms; Theory of computation → Graph algorithms analysis; Theory of computation → Approximation algorithms analysis

Keywords and phrases 3-path partition, exact set cover, approximation algorithm, local search, amortized analysis

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.46

Funding *Yong Chen*: NSFC Grants 11971139 and 11571252; CSC Grant 201508330054

Randy Goebel: NSERC Canada

An Zhang: NSFC Grants 11771114 and 11571252; CSC Grant 201908330090

¹ Corresponding author



1 Introduction

In the minimum k -path partition (abbreviated as Min- k -PP) problem, a simple graph $G = (V, E)$ is partitioned into the smallest number of paths such that each path has at most k vertices for a given positive integer k . We can observe that when $k = 2$ the Min- k -PP problem is closely related to the maximum matching problem in an unweighted graph and that when $k = n$ the Min- k -PP problem has an optimal value 1 if and only if the input graph contains a Hamiltonian path. Besides, the Min- k -PP problem can be treated as a special case of the minimum k -set exact cover problem by associating each path (of order at most k) with a set of size at most k . The decision version of the minimum exact cover problem is one of Karp's 21 NP-complete problems [5]. To the best of our knowledge, there are no non-trivial approximation algorithms for the minimum k -set exact cover problem. The Min- k -PP problem also has a close relation to the classic minimum k -set cover problem, which does not require the mutual disjointness for the resultant cover. Though the Min- k -PP and minimum k -set cover problems share some similarities, none contains the other as a special case. A detailed discussion of the relationship between them can be found in [1].

It is not hard to see the Min- k -PP problem is NP-hard on general graphs [4]. It remains intractable on cographs [8] and chordal bipartite graphs [9] when k is an input. Moreover, the Min- k -PP problem remains to be NP-hard in comparability graphs even for $k = 3$ [9]. Recently, Korpelainen [6] further investigated and depicted the NP-hardness of the Min- k -PP problem in some special graph classes.

On the positive side, the Min- k -PP problem is polynomial-time solvable in several special cases. Motivated by the application in network broadcasting, which finds the minimum number of message originators necessary to broadcast a message to all vertices in a tree network in one or two time units, Yan et al. [10] presented a linear-time algorithm for the Min- k -PP problem on trees. A polynomial-time algorithm on cographs when k is fixed was designed by Steiner [8], who later proposed a polynomial-time solution for the Min- k -PP problem, with any k , on bipartite permutation graphs [9].

Monnot and Toulouse [7] are the pioneer to investigate the approximability for the Min- k -PP problem. In particular, they studied the special case, Min-3-PP, and designed a neat $3/2$ -approximation algorithm with a running time $O(nm + n^2 \log n)$ for general graphs, where n and m are the numbers of vertices and edges in the graph. Recently, Chen et al. [2] presented an improved approximation algorithm with a ratio $13/9$ by first computing a k -path partition with the least 1-paths for any $k \geq 3$ and then greedily merging three 2-paths into two 3-paths whenever possible. Their greedy algorithm takes $O(nm)$ and $O(n^3)$ time respectively in these two steps. Based on the first step of the $13/9$ -approximation algorithm, Chen et al. [1] designed a novel local search scheme to improve the approximation ratio to $4/3$. Specifically, their local search algorithm repeatedly searches for an expected collection of 2- and 3-paths and replaces it by a strictly smaller replacement collection of new 2- and 3-paths. It is worth noting that the ratio $4/3$ matches the best approximation ratio for the minimum 3-set cover problem [3]. Due to the similarity between the Min-3-PP problem and the minimum 3-set cover problem, it seems difficult to improve this ratio a step further and Chen et al. [1] left an open question for a better approximation algorithm for the Min-3-PP problem.

Our paper addresses this open question proposed by Chen et al. [1]. Our main contributions are as follows. 1) We propose a novel weight function for ℓ -paths, $\ell \in \{1, 2, 3\}$, which forces any heavy path partition prefer specific combinations of ℓ -paths. In particular, the number of 1-paths in a heavy path partition cannot be too large. 2) We design a

greedy local search strategy (named as GREEDY) to generate a path partition that contains 2-paths “sparsely” compared with the 3-paths. Moreover, we are able to show there exists an optimal solution whose number of 1-paths is upper bounded by our greedy solution. 3) We design another well-designed wide-range tree-like search strategy (named as TREESearch) to further reduce the number of 1-paths. More specifically, the resultant path partition contains the least amount of 1-paths. 4) We conduct an delicate amortized analysis to show the “sparseness” of 2-paths quantitatively, leading to a better approximation ratio $\frac{21}{16}$ ($= 1.3125 < 1.3333 \approx \frac{4}{3}$) for the Min-3-PP problem.

In the following, Section 2 introduces basic concepts and notations; Section 3 restates the classic $\frac{3}{2}$ -algorithm by Monnot and Toulouse [7]; Section 4 shows our $\frac{21}{16}$ -approximation algorithm based on non-trivial discoveries of the structural properties of the Min-3-PP problem; Section 5 concludes the paper and proposes open questions.

2 Preliminaries

We begin with definitions and notations which hold throughout this paper. Let $G = (V, E)$ be a simple undirected graph, defined by a set of vertices $V = \{v_1, v_2, \dots, v_n\}$ and a set of undirected edges $E = \{e_1, e_2, \dots, e_m\}$, where each edge $e = \{u, v\}$ connects two vertices $u, v \in V$. Let $U \subset V$ be any subset of vertices of G . Then the (vertex) induced subgraph $G[U]$ is the subgraph whose vertex set is U and whose edge set consists of edges in E with both endpoints in U . For any subgraph S of G , let $V_G(S)$ and $E_G(S)$ denote the vertex set and edge set of S , respectively. The order of S is defined as the cardinality of $V_G(S)$. For each vertex $v \in V$, define its neighbor set as $N_G(v) = \{u \mid \{u, v\} \in E(G)\}$ and its degree as $d_G(v) = |N_G(v)|$. If the underlying graph G is clear, we may omit the subscript G in all notations for the sake of simplicity. In sequel, we use \cup and \uplus to denote the set union and multiset union respectively. For any graph (even multigraph) S with $V(S) \subseteq V_G$, we abuse the notation $G[S]$ to denote the induced graph $(V_G(S), E(S))$.

A path P in G is a sequence of distinct vertices $\langle v_1, v_2, \dots, v_\ell \rangle$, $\ell \geq 1$, such that $\{v_i, v_{i+1}\} \in E$, for $i = 1, 2, \dots, \ell - 1$. We say a path is an ℓ -path if its order is ℓ , i.e., $|V(P)| = \ell$. A path partition of G is a collection of vertex disjoint paths \mathcal{P} such that $V(G) = \bigcup_{P \in \mathcal{P}} V(P)$. A k -path partition is a path partition \mathcal{P} with each path having at most k vertices for a given positive integer k , and the minimum k -path partition problem aims at minimizing the cardinality of \mathcal{P} . Our paper considers the special case when $k = 3$. In the following context, a path partition is a 3-path partition by default.

Consider an optimal path partition $\mathcal{P}^* = \{\mathcal{P}_1^*, \mathcal{P}_2^*, \mathcal{P}_3^*\}$, where \mathcal{P}_ℓ^* , $\ell \in \{1, 2, 3\}$ denotes the set of ℓ -paths. Suppose $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3\}$ is any feasible path partition on G . Let $\text{OPT} = |\mathcal{P}^*|$ and $\text{SOL} = |\mathcal{P}|$. Denote the cardinality of \mathcal{P}_ℓ^* (\mathcal{P}_ℓ , respectively) as p_ℓ^* (p_ℓ , respectively), $\ell \in \{1, 2, 3\}$. Then we have

$$n = p_1^* + 2p_2^* + 3p_3^* = p_1 + 2p_2 + 3p_3, \quad (1)$$

$$\text{OPT} = p_1^* + p_2^* + p_3^*, \quad (2)$$

$$\text{SOL} = p_1 + p_2 + p_3, \quad (3)$$

which implies

$$\text{SOL} \geq \text{OPT} \geq \frac{n}{3}. \quad (4)$$

Let \mathcal{Q}_ℓ , $\ell \in \{1, 2, 3\}$ denote the collection of all possible ℓ -paths in the given graph G and define $\mathcal{Q} = \mathcal{Q}_1 \cup \mathcal{Q}_2 \cup \mathcal{Q}_3$. We introduce two important concept: *conflict graph* and *intersection graph*.

► **Definition 1** (Conflict graph). *In a conflict graph, denoted by CG , each vertex represents an element of \mathcal{Q} , i.e., some ℓ -path in G , $\ell \in \{1, 2, 3\}$; and each edge $\{P, Q\}$ with $P, Q \in \mathcal{Q}$ exists only if $V_G(P) \cap V_G(Q) \neq \emptyset$. In addition, we add i parallel edges between two vertices in CG if and only if two corresponding paths in G have i vertices in common.*

► **Definition 2** (Intersection graph). *Considering the optimal partition \mathcal{P}^* and any feasible partition \mathcal{P} , the intersection graph induced \mathcal{P} and \mathcal{P}^* , denoted by IG , is a bipartite graph, where each vertex represents an element of \mathcal{P} and \mathcal{P}^* , and two vertices in IG are adjacent if and only if the vertex sets of the corresponding paths in G intersect. Similar to the conflict graph, parallel edges are allowed in IG .*

According to the above definitions, parallel edges are allowed in both CG and IG . Next, we partition the edges of IG into sets $E_{ij}, i, j \in \{1, 2, 3\}$ such that $E_{ij} = \{\{P, P^*\} \mid P \cap P^* \neq \emptyset, P \in \mathcal{P}_i, P^* \in \mathcal{P}_j^*\}$. Let m_{ij} denote the cardinality of E_{ij} . We have

$$p_i = \frac{1}{i} \sum_{j=1}^3 m_{ij}, \quad p_j^* = \frac{1}{j} \sum_{i=1}^3 m_{ij}, \quad n = \sum_{i,j} m_{ij}. \quad (5)$$

Then the relation between any feasible solution and the optimal solution can be represented as follows.

$$\begin{aligned} \text{SOL} &= p_1 + p_2 + p_3 = \sum_{i=1}^3 \left(\frac{1}{i} \sum_{j=1}^3 m_{ij} \right) \\ &= (m_{11}/1 + m_{12}/1 + m_{13}/1) + (m_{21}/2 + m_{22}/2 + m_{23}/2) + (m_{31}/3 + m_{32}/3 + m_{33}/3) \\ &= (m_{31}/3 + m_{21}/2 + m_{11}/1) + (m_{32}/3 + m_{22}/2 + m_{12}/1) + (m_{33}/3 + m_{23}/2 + m_{13}/1) \\ &= \left(\sum_{i=1}^3 m_{i1} - \frac{2}{3}m_{31} - \frac{1}{2}m_{21} \right) + \left(\frac{1}{2} \sum_{i=1}^3 m_{i2} - \frac{1}{6}m_{32} + \frac{1}{2}m_{12} \right) + \left(\frac{1}{3} \sum_{i=1}^3 m_{i3} + \frac{1}{6}m_{23} + \frac{2}{3}m_{13} \right) \\ &= p_1^* + p_2^* + p_3^* + \left(\frac{2}{3}m_{13} + \frac{1}{2}m_{12} + \frac{1}{6}m_{23} - \frac{2}{3}m_{31} - \frac{1}{2}m_{21} - \frac{1}{6}m_{32} \right) \\ &= \text{OPT} + \left(\frac{2}{3}m_{13} + \frac{1}{2}m_{12} + \frac{1}{6}m_{23} - \frac{2}{3}m_{31} - \frac{1}{2}m_{21} - \frac{1}{6}m_{32} \right). \end{aligned} \quad (6)$$

3 A brief review of the classic 3/2-approximation algorithm

For the algorithm designed by Monnot and Toulouse [7], the main idea is to first compute a maximum matching M_1 in the input graph G and then find another maximum matching M_2 to connect M_1 with the vertices left from the calculation of M_1 . Since each time an edge is added to the solution, the number of connected components decreases by 1 and therefore $\text{SOL} = n - |M_1| - |M_2|$. Let's consider the vertex set left over after the first maximum matching M_1^* is found, i.e., $V \setminus V(M_1^*)$. For any vertex $v \in (V \setminus V(M_1^*)) \setminus \{\cup_{P \in \mathcal{P}_1^*} V(P)\}$, v must be contained in some ℓ -path with $\ell \geq 2$ in the optimal partition, which implies v is adjacent to some edge in M_1^* . Each vertex in $V(M_1^*)$ can be adjacent to at most two such vertices and from the maximality each edge $\{u, v\}$ in M_1^* can be adjacent to at most two such vertices. Also from the maximality, we have $|M_2| \geq \frac{1}{2}(n - 2|M_1| - p_1^*)$, which implies $\text{SOL} \leq \frac{1}{2}(n + p_1^*)$. Therefore, $\text{OPT} = p_3^* + p_2^* + p_1^* = \frac{1}{3}(n + p_2^* + 2p_1^*) \geq \frac{1}{3}(n + p_1^*) \geq \frac{2}{3}\text{SOL}$, which shows an approximation ratio of 3/2.

4 The 21/16-Approximation Algorithm

Based on the observation and analysis of the equation (6), our main idea is to find a partition \mathcal{P} to minimize m_{13} , m_{12} and m_{23} . Note that the analysis of our algorithm only depends on the existence of an optimal partition and we do not need to find an optimal partition.

Due to the definitions of the conflict graph and intersection graph, we may abuse an ℓ -path to denote a vertex in CG and IG, vice versa. Recall that \mathcal{Q} is the collection of all ℓ -paths, $\ell \in \{1, 2, 3\}$, in G . We define a weight function $w(\cdot)$ mapping each path to a real value. Specifically, let

$$w(P) = \begin{cases} 1, & \text{if } P \in \mathcal{Q}_1; \\ 5, & \text{if } P \in \mathcal{Q}_2; \\ 8, & \text{if } P \in \mathcal{Q}_3. \end{cases} \tag{7}$$

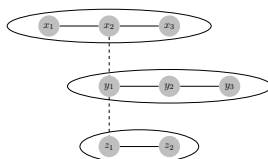
For any subset $\mathcal{Q}' \subset \mathcal{Q}$, let $w(\mathcal{Q}')$ denote the total weight of paths in \mathcal{Q}' .

► **Definition 3 (Improving set).** For any $\mathcal{I} \subset \mathcal{Q} \setminus \mathcal{P}$, $N_{CG}(\mathcal{I}, \mathcal{P}) = N_{CG}(\mathcal{I}) \cap \mathcal{P}$ is the set of neighbors of \mathcal{I} restricted in \mathcal{P} . \mathcal{I} is an improving set if $w(\mathcal{I}) > w(N_{CG}(\mathcal{I}, \mathcal{P}))$ and $(\mathcal{P} \setminus N_{CG}(\mathcal{I}, \mathcal{P})) \cup \mathcal{I}$ is a partition of G .

Our algorithm, named as GREEDY-TREESearch, is a local search algorithm, which invokes two local search strategies, GREEDY and TREESearch, as subroutines iteratively until the total weight of the partition cannot be increased.

The intuition behind the design of our weight function is that we prefer partitioning a 4-path into two 2-paths over partitioning into one 1-path and one 3-path and partitioning a 6-path into two 3-paths over three 2-paths. When analyzing the algorithm GREEDY-TREESearch, an optimal solution is compared. Without loss of generality, we choose the optimal path partition \mathcal{P}^* with the maximum weight with respect to our weight function.

Considering two vertex disjoint paths $X, Y \in \mathcal{P}$, we say X and Y are *friends* if there is an edge $\{u, v\}$ incident to both X and Y in the original graph G . Refer to Figure 1 for more details. If u (v , respectively) is the ending vertex of X (Y , respectively), connecting X and Y in G via $\{u, v\}$ forms a path in G and we say X and Y are *close friends* with respect to $\{u, v\}$. We also say X is a *close friend* to Y via $\{u, v\}$ or via u or via v . Otherwise, assuming u is the middle vertex of X , we say X is an *ordinary friend* to Y via $\{u, v\}$. Or we just say X and Y are *ordinary friends* via $\{u, v\}$. According to the definitions of different types of friends, we have the following observation.



■ **Figure 1** An illustration of friends. The solid and dashed edges denote the edges in $E(\mathcal{P})$ and $E(G) \setminus E(\mathcal{P})$, respectively. The paths in \mathcal{P} are indicated in ellipses. The $\{\langle z_1, z_2 \rangle, \langle y_1, y_2, y_3 \rangle\}$ and $\{\langle y_1, y_2, y_3 \rangle, \langle x_1, x_2, x_3 \rangle\}$ are two pairs of friends. The first pair are close friends while the second pair are ordinary friends.

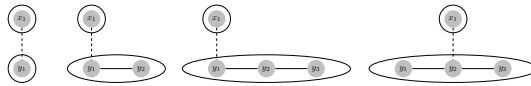
► **Observation 4.** For any two paths X and Y in a path partition \mathcal{P} , X and Y are ordinary friends only if at least one of them is a 3-path and the friendship is built through the middle vertex of the 3-path.

4.1 The Greedy Algorithm

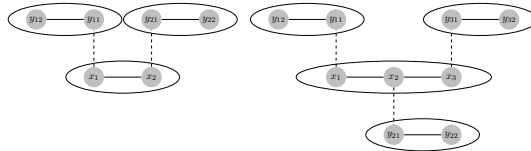
The algorithm GREEDY starts with an arbitrary path partition for G and then increases the partition's weight by iteratively updating the current partition with an improving set of size at most 6 until there are no improving sets. The value 6 is determined in the proof for Lemma 14, where at most 6 paths are involved for the detection of an improving set. We abuse the notation \mathcal{P} to denote the resultant partition when GREEDY terminates.

► **Lemma 5.** *For any two friends X and Y in \mathcal{P} , if X is a 1-path, Y can only be a 3-path and the friendship is built through the middle vertex on Y .*

Proof. Recall that the weight of a 1-, 2-, 3-path is 1, 5, 8, respectively. We prove the lemma by contradiction. If Y is an ℓ -path with $\ell \leq 2$, connecting X and Y forms an $(\ell + 1)$ -path, which is an improving set. Refer to the left two subfigures of Figure 2. If Y is a 3-path and Y is a close friend of X , connecting X and Y produces a 4-path, which can be partitioned into two 2-paths $\langle x_1, y_1 \rangle$ and $\langle y_2, y_3 \rangle$. It is easy to check these two paths form an improving set. Refer to the third subfigure of Figure 2. The GREEDY algorithm will not terminate if there is an improving set. This implies the correctness of the lemma. ◀



■ **Figure 2** Four subgraphs of G show the different cases of the friendship involving a 1-path. The solid and dashed edges denote the edges in $E(\mathcal{P})$ and $E(G) \setminus E(\mathcal{P})$, respectively. The paths in \mathcal{P} are indicated in ellipses.



■ **Figure 3** The left and right subgraphs show the friendship of a 2-path and 3-path, respectively. The solid and dashed edges denote the edges in $E(\mathcal{P})$ and $E(G) \setminus E(\mathcal{P})$, respectively. The paths in \mathcal{P} are indicated in ellipses.

► **Lemma 6.** *Consider an ℓ -path $X = \langle x_1, \dots, x_\ell \rangle$ in \mathcal{P} . Let \mathcal{F}_i denote the set of 2-path friends in \mathcal{P} via x_i , $i \in \{1, \dots, \ell\}$ in the multigraph induced by the union of paths in \mathcal{P} and \mathcal{P}^* , that is, $G[E(\mathcal{P}^*) \uplus E(\mathcal{P})]$.*

- In $G[E(\mathcal{P}^*) \uplus E(\mathcal{P})]$, X has at most 2ℓ distinct friends in \mathcal{P} .
- There are at most $\ell - 1$ distinct 2-paths Y_i such that $Y_i \in \mathcal{F}_i$.

Proof. By Lemma 5, we only need to consider $\ell \in \{2, 3\}$. Since each path in \mathcal{P}^* has an order at most 3, X can be adjacent to at most 2 other paths in \mathcal{P} via one vertex and therefore X has at most 2ℓ distinct friends in \mathcal{P} . Suppose there are ℓ distinct 2-paths Y_i such that $Y_i \in \mathcal{F}_i$. Assume $Y_i = \langle y_{i1}, y_{i2} \rangle$ are the 2-path friends via the ending vertex x_i . Refer to Figure 3. Then the ℓ 3-paths $\langle x_i, y_{i1}, y_{i2} \rangle$, $i \leq \ell$, form an improving set, which indicates a contradiction. ◀

Recall that the chosen optimal path partition \mathcal{P}^* has the maximum weight with respect to our weight function. In addition, we require that $|E_G(\mathcal{P}^*) \cap E_G(\mathcal{P})|$ is maximized over all the heaviest optimal path partitions. In other words, we consider the heaviest optimal path partition that overlaps our solution as many edges as possible. Then we have the following lemma.

► **Lemma 7.** *Let \mathcal{P}^* be the heaviest optimal path partition that maximizes $|E_G(\mathcal{P}^*) \cap E_G(\mathcal{P})|$. We have $\mathcal{P}_1^* \subseteq \mathcal{P}_1$ and thus $p_1^* \leq p_1$.*

Proof. Suppose $X = \langle x \rangle \in \mathcal{P}_1^* \setminus \mathcal{P}_1$. x must be on some ℓ -path P in $\mathcal{P}_2 \cup \mathcal{P}_3$. Since \mathcal{P}^* has the maximum weight, any 1-path in \mathcal{P}^* can only have an ordinary friend in \mathcal{P}^* by a similar argument in the proof for Lemma 5. Without loss of generality, we assume y is a neighbor of x on P . Then y has to be the middle vertex of a 3-path in \mathcal{P}^* , denoted by $Y = \langle y_1, y, y_3 \rangle$. Since $\{x, y\} \in E(\mathcal{P})$, at least one of $\{y, y_1\}$ and $\{y, y_3\}$ is not in $E(\mathcal{P})$. Assume $\{y, y_1\} \notin E(\mathcal{P})$, we modify the paths X and Y to $\langle x, y, y_3 \rangle$ and $\langle y_1 \rangle$. This modification does not change the weight of \mathcal{P}^* and $|E_G(\mathcal{P}^*) \cap E_G(\mathcal{P})|$ is increased by 1, which contradicts to the maximality. ◀

4.2 The TreeSearch Algorithm

The algorithm TREESEARCH aims at reducing the number of 1-paths in the partition \mathcal{P} returned by the GREEDY algorithm. Though the TREESEARCH algorithm also targets to find an improving set, the size of an improving set may be fairly large.

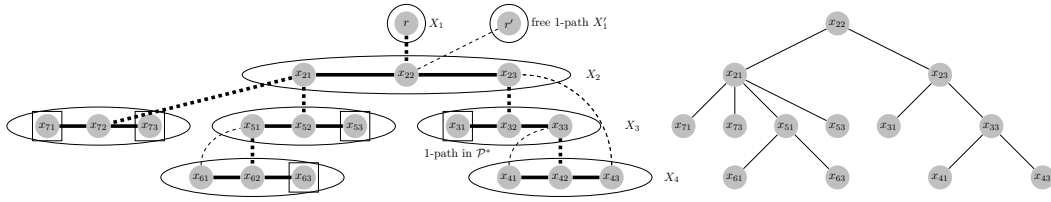
Fix any 1-path $\langle r \rangle$ in \mathcal{P} . We modify the depth first search (DFS for short) to explore G with the root r . Recall that the DFS grows a tree node by node as deep as possible to expand the whole connected graph. We modify the DFS as follows: if the currently exploring vertex u is connected with its parent via an edge in $E(G) \setminus E(\mathcal{P})$ in the current tree, instead of expanding the vertex as the normal DFS, we expand the current tree simply with the ℓ -path in \mathcal{P} containing u , and then the normal DFS is applied on the ending vertex (or vertices) of this ℓ -path. Abbreviate the modified DFS as MDFS. Note that the MDFS produces a forest instead of a spanning tree as the MDFS does not fully expand every node in the graph.

We say an ℓ -path is *involved* in a tree path P_T starting at r , if the intersection of P_T and this ℓ -path is not empty. In Figure 4, the tree path $\langle r, x_{22}, x_{23}, x_{32}, x_{42}, x_{43} \rangle$ involves paths X_1, X_2, X_3, X_4 . Once the MDFS finds an improving set for the ℓ -paths involved along a path starting at r , the TREESEARCH algorithm refines the current partition. Our TREESEARCH algorithm invokes the MDFS on the 1-paths iteratively until no more improving sets can be identified.

Let's consider the multigraph induced by the union of paths in \mathcal{P} and \mathcal{P}^* , i.e., $G[E(\mathcal{P}^*) \uplus E(\mathcal{P})]$. Let F be the resultant forest returned by the TREESEARCH algorithm on this multigraph. For any 3-path in \mathcal{P} involved in F , its endpoints are expanded by the MDFS but its middle vertex is ignored during the MDFS search. It is possible that this middle vertex is connected to at most two friends in \mathcal{P} . In particular, if the friends is a 1-path, we call it as a *free 1-path*, which forms a tree of size 1 in the forest F . Refer to Figure 4 for an example of a free 1-path.

► **Lemma 8.** *Consider any tree T in the forest F . For any path connecting the root r and a leaf in T , suppose the ℓ -paths in \mathcal{P} involved in the path are $\langle X_1, X_2, \dots, X_t \rangle$ in order with $X_1 = \{r\}$. We claim that*

- X_2, \dots, X_t are all 3-paths and moreover X_i and X_{i+1} are ordinary friends, $i \in \{1, \dots, t-1\}$;
- any two involved 3-paths say, X and Y , cannot be connected via the ending vertices in $G[E(\mathcal{P}^*) \uplus E(\mathcal{P})]$;
- the two ending vertices of X_i are not connected in $G[E(\mathcal{P}^*) \uplus E(\mathcal{P})]$.



■ **Figure 4** The left subfigure is an MDPS tree, which is highlighted in bold. The solid and dashed edges denote the edges in $E(\mathcal{P})$ and $E(\mathcal{P}^*)$, respectively. The paths in \mathcal{P} are indicated in ellipses. The 1-paths in \mathcal{P}^* are indicated in squares. The right subfigure is the contracted tree \tilde{T} .

Proof. We prove the first claim in the lemma by contradiction. Assume $X_j, j \geq 2$ is the first ℓ -path such that $\ell < 3$ or it is a close friend to X_{j-1} . Suppose $X_i = \langle x_{i,1}, x_{i,2}, x_{i,3} \rangle, i \in \{2, \dots, j-1\}$. If $j = 2$, the proof reduces to the proof of Lemma 5. We assume $j \geq 3$ without loss of generality.

- X_j is an ℓ -path with $\ell \leq 2$. The 3-paths $\langle r, x_{2,2}, x_{2,1} \rangle, \langle x_{2,3}, x_{3,2}, x_{3,1} \rangle, \dots, \langle x_{j-2,3}, x_{j-1,2}, x_{j-1,1} \rangle$ together with the $(\ell + 1)$ -path connecting $x_{j-1,3}$ and X_j form an improving set.
- X_j is a 3-path and X_j is a close friend of X_{j-1} . Connecting X_j and $X_{j-1,3}$ produces a 4-path, which can be partitioned into two 2-paths. Together with the 3-paths $\langle r, x_{2,2}, x_{2,1} \rangle, \langle x_{2,3}, x_{3,2}, x_{3,1} \rangle, \dots, \langle x_{j-2,3}, x_{j-1,2}, x_{j-1,1} \rangle$, we find an improving set.

The TREESEARCH algorithm will not terminate if there is an improving set, which implies the correctness of the claim.

By the first claim, each tree in the forest produced by the TREESEARCH algorithm on $G[E(\mathcal{P}^*) \uplus E(\mathcal{P})]$ can only involve 3-paths if its root is a 1-path in \mathcal{P} . Suppose there are two involved 3-paths X and Y such that they are connected via the ending vertices. Consider X if 1) X and Y are involved in the same root-to-leaf path and X is the ancestor; 2) X and Y are involved in different root-to-leaf paths. An improving set can be found similarly following the argument for the first claim.

The third claim states the edge $\{x_{i,3}, x_{i,1}\}$ does not exist in $G[E(\mathcal{P}^*) \uplus E(\mathcal{P})]$. Otherwise, we can find a 4-path $\langle x_{i-1,3}, x_{i,2}, x_{i,3}, x_{i,1} \rangle$ if $i \geq 3$ or $\langle r, x_{i,2}, x_{i,3}, x_{i,1} \rangle$ if $i = 2$. Then an improving set can be identified similar to the proof for the first claim. ◀

Let's consider any tree T in the forest F returned by the TREESEARCH algorithm. If we contract the edges in \mathcal{P}^* , T become a tree where each internal node has a degree 2 or 4. Denote the contracted tree as \tilde{T} . Refer to Figure 4 for details.

▶ **Lemma 9.** $m_{13} + m_{12} \leq m_{31}$.

Proof. Since the \mathcal{P}^* contains paths of order at most 3, each edge in $E_G(T) \cap E_G(\mathcal{P}^*)$ can be connected at most once to some leaf in T . In the left subfigure in Figure 4, the edge $\{x_{23}, x_{32}\}$ is connected with the leaf x_{43} in a 3-path in \mathcal{P}^* . Each leaf in T is also a vertex in G thus must be included in some ℓ -path in \mathcal{P}^* . According to Lemma 8, different leaves can only be connected to different edges in $E(T) \cap E(\mathcal{P}^*)$, i.e., different internal nodes in \tilde{T} . If a leaf is not connected to the other vertex, it must be a 1-path in \mathcal{P}^* , which contributes 1 to the value of m_{31} . Suppose the number of leaves is L . Also assume the number of leaves that are connected to edges in $E(T) \cap E(\mathcal{P}^*)$ is L_1 . Denote the number of remaining leaves as $L_2 = L - L_1$.

In the contracted tree \tilde{T} , each internal node representing edge(s) in $E(\mathcal{P}^*)$ has at least 2 children. Thus, the number of leaves in \tilde{T} is at least 1 plus the number of the internal vertices. Since the \mathcal{P}^* contains paths of order at most 3, each internal node in \tilde{T} can be

connected to at most one leaf or one free path. Assume without loss of generality that the number of these two types of internal nodes are I_1 and I_2 respectively. Each node of the second type contributes 1 to m_{13} if the free path is a 1-path. Denote the number of internal nodes to be I . Then $I_1 + I_2 \leq I$.

According to the definitions of I_1, I_2, L_1, L_2 , and L , we have

$$\begin{aligned} I_1 + I_2 + 1 \leq I + 1 \leq L = L_1 + L_2; L_1 = I_1; m_{13} + m_{12} = I_2 + 1; m_{31} \\ = L_2; m_{13} \leq I_2 + 1; m_{12} \leq 1, \end{aligned}$$

where the last inequality is because of the root contributing 1 to m_{12} or m_{13} . Using the above equations and summarizing over all trees in the forest F , we have

$$m_{13} + m_{12} = I_2 + 1 \leq L - L_1 = L_2 = m_{31}. \quad \blacktriangleleft$$

► **Corollary 10.** *Comparing the resultant partition \mathcal{P} and any optimal partition \mathcal{P}^* , \mathcal{P} has less amount of 1-paths than \mathcal{P}^* .*

Proof. $p_1 = m_{13} + m_{12} + m_{11} \leq m_{31} + m_{11} \leq m_{31} + m_{21} + m_{11} = p_1^*$. ◀

Corollary 10 is coincident with an intermediate result in [2, 1]. Combining with Lemma 7, we have the following theorem.

► **Theorem 11.** *For the path partition obtained by our GREEDY-TREESearch algorithm, there exists an optimal path partition \mathcal{P}^* such that $\mathcal{P}_1^* = \mathcal{P}_1$ and $m_{13} = m_{12} = m_{31} = m_{21} = 0$.*

4.3 Algorithm Analysis

Recall that

$$\text{SOL} = \text{OPT} + \left(\frac{2}{3}m_{13} + \frac{1}{2}m_{12} + \frac{1}{6}m_{23} - \frac{2}{3}m_{31} - \frac{1}{2}m_{21} - \frac{1}{6}m_{32} \right).$$

By Theorem 11, we have $m_{13} = m_{12} = m_{31} = m_{21} = 0$ and thus

$$\text{SOL} \leq \text{OPT} + \frac{1}{6}(m_{23} - m_{32}). \quad (8)$$

$m_{23} \leq n$ holds trivially as there are at most $\frac{n}{2}$ 2-paths in \mathcal{P} and each 2-path contributes at most 2 to m_{23} . Thus, we have

$$\text{SOL} \leq \text{OPT} + \frac{n}{6} \leq \text{OPT} + \frac{1}{2}\text{OPT} = \frac{3}{2}\text{OPT},$$

which matches the ratio obtained by Monnot and Toulouse [7]. Next, we present an amortized analysis to show the value of the second term in (8), $\frac{1}{6}m_{23}$ in particular, is actually much less than $\frac{n}{6}$. The idea behind is to lower bound the number of 3-paths in \mathcal{P} by the number of *effective* 2-paths in \mathcal{P} , where an *effective* 2-path means a 2-path that contributes to the value of the second term in (8). The relation between 2-paths and 3-paths is in turn used to upper bound the number of 2-paths. Assume $\alpha \cdot p_2 \leq p_3$ for some $\alpha > 0$. Since $2p_2 + 3p_3 \leq n$, we have $p_2 \leq \frac{n}{2+3\alpha}$. Each 2-path contributes at most 2 to the value of m_{23} . Then we have

$$\text{SOL} \leq \text{OPT} + \frac{1}{6} \cdot m_{23} \leq \text{OPT} + \frac{1}{6} \cdot \frac{2n}{2+3\alpha} \leq \left(\frac{1}{2+3\alpha} + 1 \right) \text{OPT}, \quad (9)$$

where the last inequality follows from the fact $\text{OPT} \geq \frac{n}{3}$.

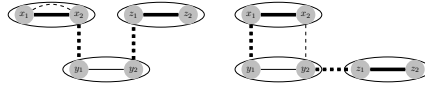
In the following, we consider a connect component in $G[E(\mathcal{P}^*) \uplus E(\mathcal{P})]$, which is induced by the 2-paths in \mathcal{P} and the neighbors in IG from \mathcal{P}^* and thus may contain parallel edges. Besides, we only consider the 2-path that shares at least one vertex with some 3-path in \mathcal{P}^* , as otherwise a 2-path contributes 0 to m_{23} and thus a non-positive value to the second term in (8).

► **Lemma 12.** *For any 2-path $X = \langle x_1, x_2 \rangle$ in \mathcal{P} , either X has at least one 3-path friend(s), or X has only one 2-path friend in \mathcal{P} which has at least one 3-path friend in \mathcal{P} .*

Proof. Let \mathcal{F}_i denote the set of 2-path friends via x_i , $i \in \{1, 2\}$. Define the cardinality of \mathcal{F}_i as f_i . Let $\vec{f} = (f_1, f_2)$. It is possible that $\mathcal{F}_1 \cap \mathcal{F}_2 \neq \emptyset$. We introduce $\mathcal{F} = \bigcup_i \mathcal{F}_i$ to denote the set of distinct 2-path friends of X . Let $f = |\mathcal{F}|$. Suppose X does not have a 3-path friend in \mathcal{P} .

According to Theorem 11, X does not share a vertex with a 1-path in \mathcal{P}^* . If X is contained in a 3-path in \mathcal{P}^* , X has only one friend via one ending vertex; otherwise, X has friend(s) via both ending vertices. By Lemma 6, $f \leq 2$. When $f = 2$, either $\vec{f} = (2, 0)$ or $\vec{f} = (1, 1)$, both of which are impossible due to Theorem 11 and Lemma 6, respectively. Thus, we have $f = 1$. Let $\mathcal{F} = \{Y\}$ and $Y = \langle y_1, y_2 \rangle$.

1. $\vec{f} = (1, 0)$: The symmetric case $\vec{f} = (0, 1)$ can be discussed similarly. X must be contained in a 3-path in \mathcal{P}^* , say $\langle x_1, x_2, y_1 \rangle$. y_2 cannot form a 1-path in \mathcal{P}^* by Theorem 11, which implies Y has another friend via y_2 , denoted by Z . If Z is a 2-path as shown in the first subfigure of Figure 5, an improving set $\langle x_1, x_2, y_1 \rangle$ and $\langle y_2, z_1, z_2 \rangle$ can be identified. This is a contradiction
2. $\vec{f} = (1, 1)$: We have $\mathcal{F}_1 = \mathcal{F}_2 = \{Y\}$. At least one of the $\langle x_1, y_1 \rangle$ and $\langle x_2, y_2 \rangle$ is a part of a 3-path in \mathcal{P}^* as otherwise X contributes 0 to m_{23} and thus can be ignored. Without loss of generality, assume Y has another friend via y_2 , denoted by Z . If Z is a 2-path as shown in the second subfigure of Figure 5, an improving set $\langle x_2, x_1, y_1 \rangle$ and $\langle y_2, z_1, z_2 \rangle$ can be identified. This is a contradiction.



■ **Figure 5** The solid and dashed edges denote the edges in $E(\mathcal{P})$ and $E(G) \setminus E(\mathcal{P})$, respectively. The paths in \mathcal{P} are indicated in ellipses. The thick paths form an improving set. ◀

► **Definition 13.** *We say a 2-path X in \mathcal{P} is the special 1-hop-away friend of a 3-path Z in \mathcal{P} , if they satisfy the relation in Figure 5.*

In the resultant partition \mathcal{P} , suppose each 3-path owns one token. We distribute each token to 2-paths in \mathcal{P} evenly if these 2-paths are the friends or special 1-hop-away friends of this 3-path in the induced graph $G[E(\mathcal{P}^*) \uplus E(\mathcal{P})]$. We say such 2-paths are *associated with* this 3-path. Assume each 2-path can receive at least γ token in average and the value of γ will be estimated in Lemma 14.

► **Lemma 14.** $\gamma = \min \left\{ \frac{2}{5}, \frac{2+\gamma}{6}, \frac{2+3\gamma}{7} \right\}$.

Proof. Consider any 3-path $X = \langle x_1, x_2, x_3 \rangle$ in \mathcal{P} . Let \mathcal{F}_i denote the set of 2-path friends via x_i , $i \in \{1, 2, 3\}$. Define the cardinality of \mathcal{F}_i as f_i . Let $\vec{f} = (f_1, f_2, f_3)$. Suppose $\mathcal{F} = \bigcup_i \mathcal{F}_i$ and $f = |\mathcal{F}|$. Assume $\mathcal{F} = \{Y_1, Y_2, \dots, Y_f\}$ and $Y_i = \{y_{i1}, y_{i2}\}$.

We claim $f \leq 4$. Otherwise, there are five different 2-path friends and we are able to find three distinct 2-paths Y_i such that $Y_i \in \mathcal{F}_i, i \in \{1, 2, 3\}$, which is contradictory to Lemma 6. Next, we discuss how to distribute the token case by case with respect to the value of $f \leq 4$ and \vec{f} . Due the page limit, we only discuss *Case 1: $f \leq 1$* and *Case 2: $f = 2$* . The discussions for *Case 3: $f = 3$* and *Case 4: $f = 4$* will be delayed to Appendix A.

Case 1: $f \leq 1$. There is at most one 2-path friend and possibly one special 1-hop-away 2-path friend. Thus, each 2-path associated with X receives at least $1/2$ token.

Case 2: $f = 2$. There are two distinct 2-path friends. If $f_1 + f_2 + f_3 \geq 4$, there exist two pairs of i and j with $i \neq j \in \{1, 2, 3\}$ such that $\mathcal{F}_i \cap \mathcal{F}_j \neq \emptyset$, which indicates that X has no special 1-hop-away 2-path friends. If X has no special 1-hop-away 2-path friends, only two 2-paths are associated with X and each receives $\frac{1}{2}$ token. In the following discussion under the Case 2, we assume X has at least one special 1-hop-away 2-path friend(s) and thus we suppose $f_1 + f_2 + f_3 \leq 3$.

Case 2.1: $f = 2$ and $f_1 + f_2 + f_3 = 2$, i.e., $\mathcal{F}_i \cap \mathcal{F}_j = \emptyset, \forall i \neq j \in \{1, 2, 3\}$.

Case 2.1.1: $\vec{f} = (2, 0, 0)$. The symmetric case $\vec{f} = (0, 0, 2)$ can be discussed similarly. Suppose $\mathcal{F}_1 = \{Y_1, Y_2\}$. By Theorem 11, Y_i must have a friend via $y_{i2}, i \in \{1, 2\}$. We claim X has at most one special 1-hop 2-path friend either via Y_1 or Y_2 . Suppose Y_i has a 2-path friend Z_i via $y_{i2}, i \in \{1, 2\}$. As shown in Figure 6, there exists an improving set $\{\langle z_{12}, z_{11}, y_{12} \rangle, \langle z_{22}, z_{21}, y_{22} \rangle, \langle y_{11}, x_1, y_{21} \rangle, \langle x_2, x_3 \rangle\}$. Moreover, at least one of Y_1 and Y_2 has a 3-path friend in \mathcal{P} , which cannot be X as $\vec{f} = (2, 0, 0)$. There are at most three 2-paths associated with X and each receives at least $\frac{1+\gamma}{3}$ token in average.

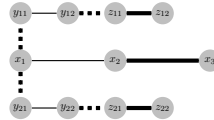
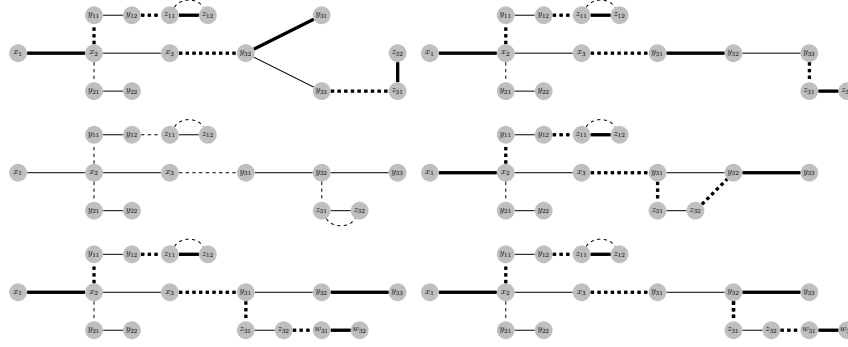


Figure 6 Case 2.1.1 $f = 2$ and $\vec{f} = (2, 0, 0)$. The solid and dashed edges denote the edges in $E(\mathcal{P})$ and $E(\mathcal{P}^*)$, respectively. The thick paths form an improving set.

Case 2.1.2: $\vec{f} = (0, 2, 0)$. Suppose $\mathcal{F}_2 = \{Y_1, Y_2\}$. By Theorem 11, Y_i has another friend via y_{i2} , denoted as $Z_i, i \in \{1, 2\}$. If at least one of the Z_1 and Z_2 is a 3-path, each 2-path (special 1-hop-away) friend receives at least $\frac{1+\gamma}{3}$ token in average. Suppose both Z_1 and Z_2 are the special 1-hop-away 2-path friends of X . In the following discussion, we focus on Z_1 . Without loss of generality.

By Theorem 11, X has friends in \mathcal{P} via both x_1 and x_3 . More specifically, these friends are 3-paths as we are discussing the case $\vec{f} = (0, 2, 0)$. Let's focus on the 3-path friend via x_3 , denoted as $Y_3 = \langle y_{31}, y_{32}, y_{33} \rangle$. For Y_3 , we define $\mathcal{F}'_i, f'_i, \mathcal{F}', f'$, and \vec{f}' similarly.

Case 2.1.2.1: Y_3 is connected with X via y_{32} . We claim $f'_1 = f'_3 = 0$ and $f'_2 \leq 1$. Suppose Y_3 has a 2-path friend via y_{33} , denoted as $Z_3 = \langle z_{31}, z_{32} \rangle$. It is possible that $Z_3 = Y_2$. As shown in the first subfigure in Figure 7, there is an improving set $\{\langle z_{12}, z_{11}, y_{12} \rangle, \langle y_{11}, x_2, x_1 \rangle, \langle x_3, y_{32}, y_{31} \rangle, \langle y_{33}, z_{31}, z_{32} \rangle\}$, which is a contradiction. $f'_2 \leq 1$ is because X is a friend of Y_3 via y_{32} and Y_3 has at most one more friend via y_{32} . Thus, there are at most five 2-paths associated with X and Y_3 and each receives at least $\frac{2}{5}$ token in average.



■ **Figure 7** Case 2.1.2 $f = 2$ and $\vec{f} = (0, 2, 0)$. The solid and dashed edges denote the edges in $E(\mathcal{P})$ and $E(\mathcal{P}^*)$, respectively. The thick paths form an improving set.

Case 2.1.2.2: Y_3 is connected with X via y_{31} . We claim $f'_3 = 0$ and $f'_1 \leq 1$. The proof is similar to the Case 2.1.2.1. An analogical example is shown in the second subfigure of Figure 7.

If $f'_1 = 1$, denote this friend via y_{31} as $Z_3 = \langle z_{31}, z_{32} \rangle$. We claim Z_3 has another 3-path in \mathcal{P} , except for X and Y_3 . It is possible that Z_3 is coincident with Y_2 . By Theorem 11, Z_3 has another friend via z_{32} . W_3 cannot be X as we are discussing under the case $\vec{f} = (0, 2, 0)$. If $W_3 = Y_3$, depending on whether the friendship is via y_{32} or y_{33} there exists an improving set $\{ \langle z_{12}, z_{11}, y_{12} \rangle, \langle y_{11}, x_2, x_1 \rangle, \langle x_3, y_{31}, z_{31} \rangle \langle z_{32}, y_{32}, y_{33} \rangle \}$ or $\{ \langle z_{12}, z_{11}, y_{12} \rangle, \langle y_{11}, x_2, x_1 \rangle, \langle x_3, y_{31}, z_{31} \rangle \langle z_{32}, y_{33}, y_{32} \rangle \}$, respectively, as shown in the fourth subfigure of Figure 7. This is a contradiction. We claim W_3 is a 3-path except for X and Y_3 . Otherwise, suppose W_3 is a 2-path. As shown in the fifth subfigure of Figure 7, there is an improving set $\{ \langle z_{12}, z_{11}, y_{12} \rangle, \langle y_{11}, x_2, x_1 \rangle, \langle x_3, y_{31}, z_{31} \rangle \langle y_{32}, y_{33} \rangle \langle z_{32}, w_{31}, w_{32} \rangle \}$, which is a contradiction.

If Y_3 has a 2-path friend Z_3 via y_{32} and Z_3 has another friend in \mathcal{P} , except for X and Y_3 , we claim Z_3 is a 3-path. Otherwise, suppose $W_3 = \langle z_{31}, z_{32} \rangle$ is a 2-path. It is possible that Z_3 is coincident with Y_2 . As shown in the sixth subfigure of Figure 7, there is an improving set $\{ \langle z_{12}, z_{11}, y_{12} \rangle, \langle y_{11}, x_2, x_1 \rangle, \langle x_3, y_{31} \rangle \langle z_{31}, y_{32}, y_{33} \rangle \langle z_{32}, w_{31}, w_{32} \rangle \}$, which is a contradiction.

Due to the previous discussion, we have $f' \leq 3$ and Y_3 has no special 1-hop-away 2-path friends.

1. $f' = 0$: there are at most four 2-paths associated with X and Y_3 , and each receives at least $\frac{1}{2}$ token in average.
2. $f' = 1$: There is only one 2-path friend associated with Y_3 . $\vec{f}' = (1, 1, 0)$ cannot happen as we discussed above. If $\vec{f}' = (1, 0, 0)$, Y_3 ' 2-path friend receives extra γ token from another 3-path. If $\vec{f}' = (0, 1, 0)$, Y_3 ' 2-path friend may not receive extra tokens from other 3-paths. There are at most five 2-paths associated with X and Y_3 . Thus each 2-path receives at least $\min\{\frac{2+\gamma}{5}, \frac{2}{5}\}$ token in average.
3. $f' = 2$: There are two 2-paths associated with Y_3 . $\vec{f}' = (1, 2, 0)$ cannot happen as we discussed above (also refer to the fourth subfigure of Figure 7). If $\vec{f}' = (1, 1, 0)$, Y_3 ' 2-path friend via y_{31} receives extra γ token from another 3-path, but Y_3 ' 2-path friend via y_{32} may not receive extra token from other 3-paths. If $\vec{f}' = (0, 2, 0)$, both Y_3 ' 2-path friends via y_{32} have another friends

except for Y_3 and X , and each receives extra γ token from other 3-paths. There are at most six 2-paths associated with X and Y_3 . Thus each 2-path receives at least $\min\{\frac{2+\gamma}{6}, \frac{2+2\gamma}{6}\}$ token in average.

4. $f' = 3$: There are three 2-paths associated with Y_3 . If $\vec{f}' = (1, 2, 0)$, every Y_3 2-path friend receives extra γ token from another 3-path except for Y_3 and X . There are at most seven 2-paths associated with X and Y_3 . Thus each 2-path receives at least $\frac{2+3\gamma}{7}$ token in average.

To summarize, each 2-path receives at least $\min\{\frac{2}{5}, \frac{2+\gamma}{6}, \frac{2+3\gamma}{7}\}$ token in average.

Case 2.1.3: $\vec{f} = (1, 1, 0)$. The symmetric case $\vec{f} = (0, 1, 1)$ can be discussed similarly. Suppose $\mathcal{F}_1 = \{Y_1\}$ and $\mathcal{F}_2 = \{Y_2\}$. We claim that X cannot have a special 1-hop-away 2-path friend via Y_2 . Otherwise, denote this friend as $Z_2 = \langle z_{21}, z_{22} \rangle$. As shown in the first subfigure of Figure 8, there is an improving set $\{ \langle x_1, y_{11}, y_{12} \rangle, \langle y_{21}, x_2, x_3 \rangle, \langle y_{22}, z_{21}, z_{22} \rangle \}$, which is a contradiction.

Suppose X has a special 1-hop-away 2-path friend via Y_1 , denoted as Z_1 .

1. If Y_2 has a friend via y_{22} , denoted as Z_2 , Z_2 is another 3-path except for X , following the previous argument. Each of three associated 2-paths receives $\frac{1+\gamma}{3}$ token from X .
2. If Y_2 has no other friends via y_{22} , it is contained in a 3-path in \mathcal{P}^* as shown in the second subfigure of Figure 8, X have a 3-path friend in \mathcal{P} via x_3 , denoted as $Y_3 = \langle y_{31}, y_{32}, y_{33} \rangle$. There are three (special 1-hop-away) 2-path friends associated with X . By a similar discussion in the Case 2.1.2 “ $f = 2$ and $\vec{f} = (0, 2, 0)$ ”, each 2-path receives at least $\min\{\frac{1}{2}, \frac{2+\gamma}{5}, \frac{2+3\gamma}{6}\}$ token from X and Y_3 token in average.

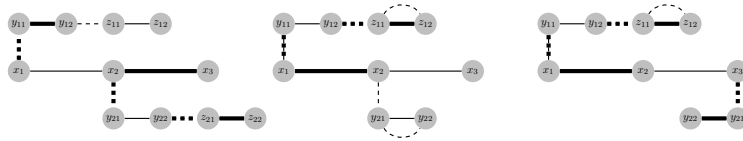


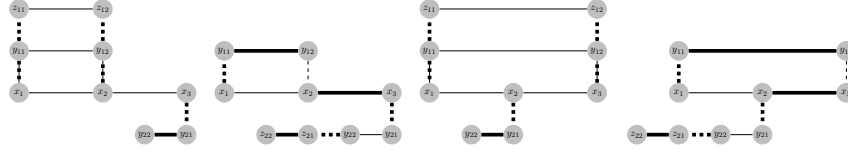
Figure 8 Case 2.1.3 $f = 2$ and $\vec{f} = (1, 1, 0)$ (left two subfigures) and Case 2.1.4 $f = 2$ and $\vec{f} = (1, 0, 1)$ (rightmost subfigure). The solid and dashed edges denote the edges in $E(\mathcal{P})$ and $E(\mathcal{P}^*)$, respectively. The thick paths form an improving set.

Case 2.1.4: $\vec{f} = (1, 0, 1)$. Suppose $\mathcal{F}_1 = \{Y_1\}$ and $\mathcal{F}_3 = \{Y_2\}$. Without loss of generality, assume X has a special 1-hop-away 2-path friend via Y_1 , denoted as Z_1 . By the definition of the special 1-hop-away 2-path friend, $Z_1 \neq Y_2$. As shown in the third subfigure of Figure 8, there is an improving set $\{ \langle y_{12}, z_{11}, z_{12} \rangle, \langle y_{11}, x_1, x_2 \rangle, \langle x_3, y_{21}, y_{22} \rangle \}$, which implies this case is impossible. That is, X has no a special 1-hop-away 2-path friends via $Y_i, i \in \{1, 2\}$.

Case 2.2: $f = 2$ and $f_1 + f_2 + f_3 = 3$. There exist some $i \neq j \in \{1, 2, 3\}$ such that $\mathcal{F}_i \cap \mathcal{F}_j \neq \emptyset$.

Case 2.2.1: $f = 2$ and $\vec{f} = (1, 1, 1)$.

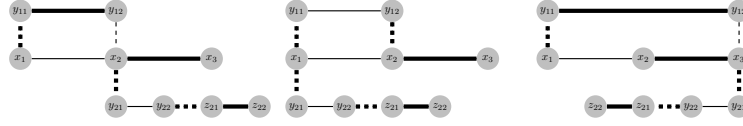
Suppose $\mathcal{F}_1 = \{Y_1\}$, $\mathcal{F}_2 = \{Y_1\}$, and $\mathcal{F}_3 = \{Y_2\}$. The symmetric case can be discussed similarly. We claim X has no a special 1-hop-away 2-path friends via $Y_i, i \in \{1, 2\}$. Otherwise, if X has a special 1-hop-away 2-path friend via Y_1 , this special 1-hop-away 2-path friend can only in the format shown in the first subfigure of Figure 9, where an improving set $\{ \langle z_{11}, y_{11}, x_1 \rangle, \langle z_{12}, y_{12}, x_2 \rangle, \langle x_3, y_{21}, y_{22} \rangle \}$ can be identified; if X has a special 1-hop-away 2-path friend via Y_2 , as shown in the



■ **Figure 9** Case 2.2.1 $f = 2$ and $\vec{f} = (1, 1, 1)$. The solid and dashed edges denote the edges in $E(\mathcal{P})$ and $E(\mathcal{P}^*)$, respectively. The thick paths form an improving set.

second subfigure of Figure 9, there is an improving set $\{\langle z_{22}, z_{21}, y_{22} \rangle, \langle y_{12}, y_{11}, x_1 \rangle, \langle x_2, x_3, y_{21} \rangle\}$, which is a contradiction.

The case where $\mathcal{F}_1 = \{Y_1\}$, $\mathcal{F}_2 = \{Y_2\}$, and $\mathcal{F}_3 = \{Y_1\}$ can be discussed similarly by following the third and fourth subfigures of Figure 9. To summarize, this case cannot happen. That is, X has no special 1-hop-away 2-path friends via Y_i , $i \in \{1, 2\}$.



■ **Figure 10** Cases 2.2.2 – 2.2.4 from left to right. ($\vec{f} = (1, 2, 0)$, $\vec{f} = (2, 1, 0)$, $\vec{f} = (1, 0, 2)$). The solid and dashed edges denote the edges in $E(\mathcal{P})$ and $E(\mathcal{P}^*)$, respectively. The thick paths form an improving set.

Case 2.2.2: $f = 2$ and $\vec{f} = (1, 2, 0)$. The symmetric case $\vec{f} = (0, 2, 1)$ can be discussed similarly. Suppose $\mathcal{F}_1 = \{Y_1\}$ and $\mathcal{F}_2 = \{Y_1, Y_2\}$. By the definition of the special 1-hop-away 2-path friend, X has no special 1-hop-away 2-path friends via Y_1 . We also claim X has no special 1-hop-away 2-path friends via Y_2 . Otherwise, denote this special 1-hop-away 2-path friend as $Z_2 = \langle z_{21}, z_{22} \rangle$. As shown in the first subfigure of Figure 10, there is an improving set $\{\langle z_{22}, z_{21}, y_{22} \rangle, \langle y_{12}, y_{11}, x_1 \rangle, \langle x_3, x_2, y_{21} \rangle\}$, which is a contradiction.

Case 2.2.3: $f = 2$ and $\vec{f} = (2, 1, 0)$. The symmetric case $\vec{f} = (0, 1, 2)$ can be discussed similarly. Suppose $\mathcal{F}_1 = \{Y_1, Y_2\}$ and $\mathcal{F}_2 = \{Y_1\}$. By the definition of the special 1-hop-away 2-path friend, X has no special 1-hop-away 2-path friends via Y_1 . We also claim X has no special 1-hop-away 2-path friends via Y_2 . Otherwise, denote this special 1-hop-away 2-path friend as $Z_2 = \langle z_{21}, z_{22} \rangle$. As shown in the second subfigure of Figure 10, there is an improving set $\{\langle z_{22}, z_{21}, y_{22} \rangle, \langle y_{11}, x_1, y_{21} \rangle, \langle x_3, x_2, y_{12} \rangle\}$, which is a contradiction.

Case 2.2.4: $f = 2$ and $\vec{f} = (1, 0, 2)$. The symmetric case $\vec{f} = (0, 1, 2)$ can be discussed similarly. Suppose $\mathcal{F}_1 = \{Y_1, Y_2\}$ and $\mathcal{F}_2 = \{Y_1\}$. By the definition of the special 1-hop-away 2-path friend, X has no special 1-hop-away 2-path friends via Y_1 . We also claim X has no special 1-hop-away 2-path friends via Y_2 . Otherwise, denote this special 1-hop-away 2-path friend as $Z_2 = \langle z_{21}, z_{22} \rangle$. As shown in the second subfigure of Figure 10, there is an improving set $\{\langle z_{22}, z_{21}, y_{22} \rangle, \langle y_{11}, x_1, y_{21} \rangle, \langle x_2, x_3, y_{12} \rangle\}$, which is a contradiction. ◀

The minimum average number of token a 2-path can receive is $\gamma = \frac{2}{5}$ by solving the equation in Lemma 14. The total number of token is p_3 and thus we have

$$\frac{2}{5} \cdot p_2 \leq p_3. \quad (10)$$

Combining inequalities (10) and (9), we have

$$\text{SOL} \leq \frac{21}{16} \cdot \text{OPT}. \quad (11)$$

► **Theorem 15.** *Our algorithm GREEDY-TREESEARCH is a $\frac{21}{16}$ -approximation algorithm.*

Proof. The approximation ratio is shown in (11). Next we argue the running time of the GREEDY-TREESEARCH algorithm is polynomial.

According to the definition of our weight function, the upper bound for the weight of a path partition is $5 \times \lceil \frac{n}{2} \rceil$. Each iteration of GREEDY-TREESEARCH identifies an improving set and the weight of the partition increases by at least 1. Therefore, our local search algorithm terminates within $O(n)$ iterations. In the worst case, finding an improving set needs to invoke both GREEDY and TREESEARCH. The subroutine GREEDY searches for the improving set of size at most 6 by exhausting all possible path set of size 6. Note that GREEDY does not need to recheck all examined path sets. Since the collection of all ℓ -path, $\ell \in \{1, 2, 3\}$, in G has a size $O(n^3)$, the total time of invoking GREEDY is $O(n^{18})$. The subroutine TREESEARCH applies the modified depth first search algorithm to G and has a time complexity $O(n + m)$. To summarize, the time complexity for GREEDY-TREESEARCH is $O(n^{18} + n \cdot (n + m)) = O(n^{18})$, which is a polynomial. ◀

5 Conclusion

We study the approximability of the minimum 3-path partition (Min-3-PP) problem, which has wide applications in the communication network. Several intrinsic structural properties on the feasible and optimal solutions are discovered. In particular, a quantitative relation between any feasible solution and the optimal solution to an arbitrary Min-3-PP instance is described. A further exploration of the optimal solution's structure distills the quantitative relation to estimate the number of a special type of 2-paths, named as *effective* 2-paths. Then we show that the number of effective 2-paths is upper bounded by a ratio of the 3-paths, which implies the number of effective 2-paths cannot be too large. Inspired by the discovered properties, a novel weighted local search algorithm is designed to obtain a better approximation ratio $\frac{21}{16}$ for the Min-3-PP problem.

As we discussed in the introduction section, the Min-3-PP problem is closely related to the minimum 3-set cover problem, for which it is widely believed difficult to break the approximation barrier of $4/3$. However, we break this barrier for the Min-3-PP problem. It will be interesting to further investigate the differences and similarities between these two problems. Since the inapproximability of the Min-3-PP problem is still open, it is interesting to investigate whether there exists a better approximation algorithm or there is an approximation barrier.

For the general minimum k -path partition problem, its approximability is open in the literature. We think it should be also interesting to design non-trivial approximation algorithms even for some fixed $k \geq 4$.

References

- 1 Y. Chen, R. Goebel, G. Lin, L. Liu, B. Su, W. Tong, Y. Xu, and A. Zhang. A local search $4/3$ -approximation algorithm for the minimum 3-path partition problem. In *FAW*, pages 14–25, 2019.
- 2 Y. Chen, R. Goebel, G. Lin, B. Su, Y. Xu, and A. Zhang. An improved approximation algorithm for the minimum 3-path partition problem. *Journal of Combinatorial Optimization*, 38(1):150–164, 2019.

- 3 R.-c. Duh and M. Fürer. Approximation of k-set cover by semi-local optimization. In *STOC*, pages 256–264, 1997.
- 4 M. R. Garey and D. S. Johnson. *Computers and intractability*, volume 29. wh freeman New York, 2002.
- 5 R. M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. 1972.
- 6 N. Korpelainen. A boundary class for the k-path partition problem. *Electronic Notes in Discrete Mathematics*, 2018.
- 7 J. Monnot and S. Toulouse. The P_k partition problem and related problems in bipartite graphs. In *SOFSEM*, pages 422–433, 2007.
- 8 G. Steiner. On the k-th path partition problem in cographs. *Congressus Numerantium*, pages 89–96, 2000.
- 9 G. Steiner. On the k-path partition of graphs. *Theoretical Computer Science*, 290(3):2147–2155, 2003.
- 10 J.-H. Yan, G. J. Chang, S. M. Hedetniemi, and S. T. Hedetniemi. k-Path partitions in trees. *Discrete Applied Mathematics*, 78(1-3):227–233, 1997.

A Further Proof for Lemma 14

Recall that Lemma 14 states each 2-path receives at least $\gamma = \min \left\{ \frac{2}{5}, \frac{2+\gamma}{6}, \frac{2+3\gamma}{7} \right\}$ token in average. We have already discussed *Case 1*: $f \leq 1$ and *Case 2*: $f = 2$. Now we continue to discuss *Case 3*: $f = 3$ and *Case 4*: $f = 4$.

Case 3: $f = 3$. There are three distinct 2-path friends. If $f_1 + f_2 + f_3 \geq 5$, we have $\vec{f} = (1, 2, 2)$ or $(2, 2, 1)$ or $(2, 1, 2)$ or $(2, 2, 2)$, which is impossible by Lemma 6. Thus, $f_1 + f_2 + f_3 \leq 4$.

Case 3.1: $f = 3$ and $f_1 + f_2 + f_3 = 3$, i.e., $\mathcal{F}_i \cap \mathcal{F}_j = \emptyset, \forall i \neq j \in \{1, 2, 3\}$.

Case 3.1.1: $f = 3$ and $\vec{f} = (0, 2, 1)$. The symmetric case $\vec{f} = (1, 2, 0)$ can be discussed similarly. Suppose $\mathcal{F}_2 = \{Y_1, Y_2\}$ and $\mathcal{F}_3 = \{Y_3\}$. Using a similar argument in the Case 2.1.3 “ $f = 2$ and $\vec{f} = (1, 1, 0)$ ”, X cannot have a special 1-hop-away 2-path friend via Y_1 and Y_2 , which can also be observed from the first subfigure of Figure 11. By Theorem 11, Y_i has friends via $y_{i2}, i \in \{1, 2\}$

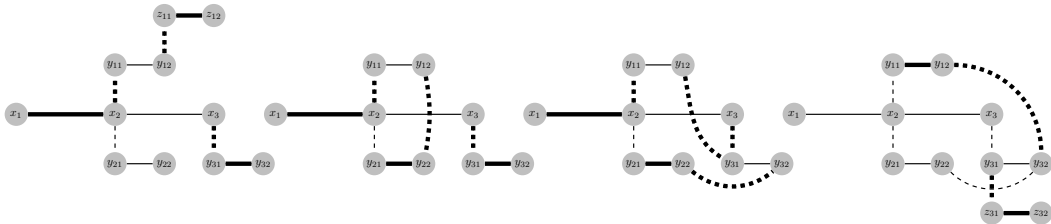


Figure 11 Case 3.1.1 $f = 3$ and $\vec{f} = (0, 2, 1)$. The solid and dashed edges denote the edges in $E(\mathcal{P})$ and $E(\mathcal{P}^*)$, respectively. The thick paths form an improving set.

We discuss the following cases.

Case 3.1.1.1: X has a special 1-hop-away 2-path friend via Y_3 . We claim both Y_1 and Y_2 have 3-path friends except for X , denoted as Z_1 and Z_2 . It is possible that $Z_1 = Z_2$. Assume without loss of generality, let’s focus on Z_1 . Suppose $Z_1 = \{z_{11}, z_{12}\}$. $Z_1 = Y_3$ is impossible in this case.

1. $Z_1 \notin \{Y_2, Y_3\}$. We can find an improving set $\{\langle x_1, x_2, y_{11} \rangle, \langle y_{12}, z_{11}, z_{12} \rangle, \langle x_3, y_{31}, y_{32} \rangle\}$ for \mathcal{P} . Refer to the first subfigure in Figure 11.
2. $Z_1 = Y_2$. The friendship between Y_1 and Y_2 can only be built via the edge $\{y_{12}, y_{22}\}$. We can find an improving set $\{\langle x_1, x_2, y_{11} \rangle, \langle y_{12}, y_{22}, y_{21} \rangle, \langle x_3, y_{31}, y_{32} \rangle\}$ for \mathcal{P} . Refer to the second subfigure of Figure 11.

There are at most four 2-paths associated with X . Y_1 and Y_2 both receive γ token from other 3-paths in \mathcal{P} . Thus each 2-path receives at least $\frac{1+2\gamma}{4}$ token in average.

Case 3.1.1.2: X does not a special 1-hop-away 2-path friend via Y_3 . We claim at least one of $Y_i, i \in \{1, 2, 3\}$ have 3-path friends except for X . Suppose $Z_1 = Y_3$ and $Z_2 = Y_3$.

1. The friendship between Y_1 and Z is built via the edge $\{y_{12}, y_{31}\}$. The friendship between Y_2 and Z can only be built via the edge $\{y_{22}, y_{32}\}$. We can find an improving set $\{\langle x_1, x_2, y_{11} \rangle, \langle y_{21}, y_{22}, y_{32} \rangle, \langle y_{12}, y_{31}, x_3 \rangle\}$ for \mathcal{P} . Refer to the third subfigure of Figure 11.
2. The friendship between Y_1 and Z is built via the edge $\{y_{12}, y_{32}\}$. The friendship between Y_2 and Z can only be built via the edge $\{y_{22}, y_{31}\}$. It is symmetric to the previous case.
3. The friendship between Y_1 and Z is built via the edge $\{y_{12}, y_{32}\}$. The friendship between Y_2 and Z is built via the edge $\{y_{22}, y_{32}\}$. If the edge $\{x_3, y_{31}\}$ is a 2-path in \mathcal{P}^* , it contributes 1 to m_{32} . Contracting the vertices x_3, y_{31}, y_{32} does not affect the value of the second term in (8). It reduces to Case 2 “ $f = 2$ ”, where X has two distinct 2-path friends. If the edge $\{x_3, y_{31}\}$ is a part of 3-path in \mathcal{P}^* and x_3 is the middle vertex, then $f_3 = 2$, which is a contradiction. If the edge $\{x_3, y_{31}\}$ is a part of 3-path in \mathcal{P}^* and y_{31} is the middle vertex, Y_3 has another friend in \mathcal{P} , denoted as $Z_3 = \langle z_{31}, z_{32} \rangle$. We claim Z' is a 3-path. Otherwise, We can find an improving set $\{\langle y_{11}, y_{12}, y_{32} \rangle, \langle y_{31}, z_{31}, z_{32} \rangle\}$ for \mathcal{P} . Refer to the fourth subfigure in Figure 11.

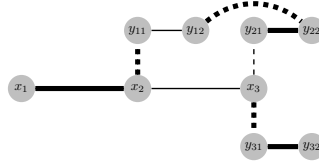
There are three 2-path associated with X . At least one Y_i receive γ token from other 3-path friends in \mathcal{P} . Thus each 2-path receives at least $\frac{1+\gamma}{3}$ token in average.

To summarize, each 2-path receives at least $\min\{\frac{1+2\gamma}{4}, \frac{1+\gamma}{3}, \frac{1}{2}\}$ token in average.

Case 3.1.2: $f = 3$ and $\vec{f} = (0, 1, 2)$. The symmetric case $\vec{f} = (2, 1, 0)$. can be discussed similarly. Suppose $\mathcal{F}_2 = \{Y_1\}$ and $\mathcal{F}_3 = \{Y_2, Y_3\}$. By Theorem 11, Y_1 has another friend via y_{12} , denoted as Z_1 . We claim Z_1 is a 3-path in \mathcal{P} , which implies X has no special 1-hop-away 2-path friend via Y_1 . Otherwise, let $Z = \langle z_1, z_2 \rangle$. If $Z \notin \{Y_2, Y_3\}$, Z must be a 3-path with a similar argument in Case 3.1.1.1; if $Z \in \{Y_2, Y_3\}$, say $Z = Y_2$, the friendship between Y_1 and Y_2 can only be built via the edge $\{y_{12}, y_{22}\}$ and we can find an improving set $\{\langle x_1, x_2, y_{11} \rangle, \langle y_{12}, y_{22}, y_{21} \rangle, \langle x_3, y_{31}, y_{32} \rangle\}$ for \mathcal{P} . Refer to Figure 12. On the other hand, following from a similar argument in Case 2.1.1 “ $f = 2$ and $\vec{f} = (2, 0, 0)$ ”, X has at most one special 1-hop-away 2-path friend either via Y_2 or Y_3 , say Y_2 without loss of generality, and Y_3 has a another 3-path friend in \mathcal{P} .

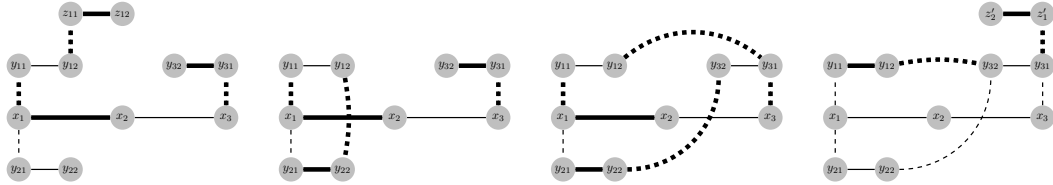
To summarize, there are at most one special 1-hop away 2-path friend associated with X and each 2-path receives at least $\min\{\frac{1+\gamma}{3}, \frac{1+2\gamma}{4}\}$ token from X .

Case 3.1.3: $f = 3$ and $\vec{f} = (2, 0, 1)$. The symmetric case $\vec{f} = (1, 0, 2)$ can be discussed similarly. Suppose $\mathcal{F}_1 = \{Y_1, Y_2\}$ and $\mathcal{F}_3 = \{Y_3\}$. Using a similar argument in the Case 2.1.4 “ $f = 2$ and $\vec{f} = (1, 0, 1)$ ”, X cannot have special 1-hop-away 2-path



■ **Figure 12** Case 3.1.2 $f = 3$ and $\vec{f} = (0, 1, 2)$. The solid and dashed edges denote the edges in $E(\mathcal{P})$ and $E(\mathcal{P}^*)$, respectively. The thick paths form an improving set.

friends via Y_1 and Y_2 , which can also be observed from the first subfigure of Figure 13. The Case 3.1.1 “ $f = 3$ and $\vec{f} = (0, 2, 1)$ ” is a “semi-symmetric” to this case. Since the edge $\{x_2, x_3\}$ is not used to construct an improving set during the discussion for the Case 3.1.1, the same argument still holds correctly. Subfigures for corresponding different subcases are shown in Figure 13. To summarize, each 2-path receives at least $\min\{\frac{1+2\gamma}{4}, \frac{1+\gamma}{3}\}$ token in average.



■ **Figure 13** Case 3.1.3 $f = 3$ and $\vec{f} = (2, 0, 1)$. The solid and dashed edges denote the edges in $E(\mathcal{P})$ and $E(\mathcal{P}^*)$, respectively. The thick paths form an improving set.

Case 3.1.4: $f = 3$ and $\vec{f} = (1, 1, 1)$. That is, $\mathcal{F}_1 = Y_1$, $\mathcal{F}_2 = Y_2$, $\mathcal{F}_3 = Y_3$, which is impossible by Lemma 6.

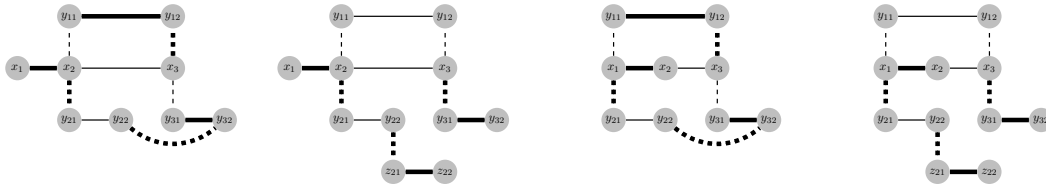
Case 3.2: $f = 3$ and $f_1 + f_2 + f_3 = 4$. There exist some $i \neq j \in \{1, 2, 3\}$ such that $\mathcal{F}_i \cap \mathcal{F}_j \neq \emptyset$.

Case 3.2.1: $\vec{f} = (0, 2, 2)$. The symmetric case $\vec{f} = (2, 2, 0)$ can be discussed similarly. Without loss of generality, suppose $\mathcal{F}_2 = \{Y_1, Y_2\}$ and $\mathcal{F}_3 = \{Y_1, Y_3\}$. By Theorem 11, Y_i has another friend via y_{i2} , denoted as Z_i , $i \in \{2, 3\}$. By a similar argument in Case 2.2.2 “ $f = 2$ and $\vec{f} = (1, 2, 0)$ ” and Case 2.2.3 “ $f = 2$ and $\vec{f} = (2, 1, 0)$ ”, we claim X has no special 1-hop-away 2-path friends via Y_i , $i \in \{1, 2, 3\}$.

We claim Z_2 is a 3-path. Otherwise, let $Z_2 = \{z_{21}, z_{22}\}$. If $Z_2 = Y_3$, the friendship between Y_2 and Y_3 can only be built via the edge $\{y_{22}, y_{32}\}$, we can find an improving set $\{\langle x_1, x_2, y_{21} \rangle, \langle y_{22}, y_{32}, y_{31} \rangle, \langle y_{11}, y_{12}, x_3 \rangle\}$ for \mathcal{P} , as shown in the first subfigure of Figure 14. If $Z_2 \neq Y_3$, there is an improving set $\{\langle x_1, x_2, y_{21} \rangle, \langle y_{22}, z_1, z_2 \rangle, \langle x_3, y_{31}, y_{32} \rangle\}$ for \mathcal{P} , as shown in the second subfigure of Figure 14.

Similarly, we can also prove Z_3 is a 3-path. To summarize, there are three 2-paths associated with X . Each of Y_2 and Y_3 receives γ token from other 3-path friends in \mathcal{P} . Thus each 2-path receives at least $\frac{1+2\gamma}{3}$ token in average.

Case 3.2.2: $\vec{f} = (2, 0, 2)$. Without loss of generality, suppose $\mathcal{F}_1 = \{Y_1, Y_2\}$ and $\mathcal{F}_3 = \{Y_1, Y_3\}$. By a similar argument in Case 2.2.4 “ $f = 2$ and $\vec{f} = (1, 0, 2)$ ”, we claim X has no special 1-hop-away 2-path friends via Y_i , $i \in \{1, 2, 3\}$. The Case 3.2.1 “ $f = 3$ and $\vec{f} = (2, 0, 2)$ ” is a “semi-symmetric” to this case. Since the edge $\{x_2, x_3\}$ is not used to construct an improving set during the discussion for the Case 3.2.1, the same argument still holds correctly. A set of subfigures for different subcases is shown in Figure 14. Each associated 2-path receives at least $\frac{1+2\gamma}{3}$ tokens in average.

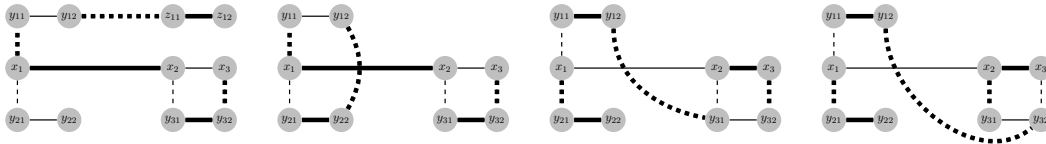


■ **Figure 14** Case 3.2.1 $f = 3, \vec{f} = (0, 2, 2)$ (left two subfigures) and Case 3.2.2 $f = 3, \vec{f} = (2, 0, 2)$ (right two subfigures). The solid and dashed edges denote the edges in $E(\mathcal{P})$ and $E(\mathcal{P}^*)$, respectively. The thick paths form an improving set.

Case 3.2.3: $\vec{f} = (2, 1, 1)$. The symmetric case $\vec{f} = (1, 1, 2)$ can be discussed similarly. Suppose $\mathcal{F}_1 = \{Y_1, Y_2\}, \mathcal{F}_2 = \mathcal{F}_3 = \{Y_3\}$. We claim X cannot have a special 1-hop-away 2-path friend via $Y_i, i \in \{1, 2, 3\}$. The correctness proof follows from a similar argument in Case 2.2.1 “ $f = 2$ and $\vec{f} = (1, 1, 1)$ ”. By Theorem 11, Y_i has another friend via y_{i2} , denoted as $Z_i, i \in \{1, 2\}$. We claim Z_i is a 3-path in $\mathcal{P}, i \in \{1, 2\}$. Otherwise, let $Z_1 = \{z_{11}, z_{12}\}$.

- If $Z \notin \{Y_2, Y_3\}$, we can find an improving set $\{\langle y_{11}, x_1, x_2 \rangle, \langle y_{12}, z_{11}, z_{12} \rangle, \langle x_3, y_{32}, y_{31} \rangle\}$ for \mathcal{P} . Refer to the first subfigure in Figure 15.
- If $Z = Y_2$, the friendship between Y_1 and Z can only be built via the edge $\{y_{12}, y_{22}\}$ and we can find an improving set $\{\langle y_{11}, x_1, x_2 \rangle, \langle y_{12}, y_{22}, y_{21} \rangle, \langle x_3, y_{32}, y_{31} \rangle\}$ for \mathcal{P} . Refer to the second subfigure in Figure 15.
- If $Z = Y_3$ and the friendship between Y_1 and Z is built via the edge $\{y_{12}, y_{31}\}$, we can find an improving set $\{\langle x_1, y_{21}, y_{22} \rangle, \langle x_2, x_3, y_{32} \rangle, \langle y_{11}, y_{12}, y_{31} \rangle\}$ for \mathcal{P} . Refer to the third subfigure in Figure 15.
- If $Z = Y_3$ and the friendship between Y_1 and Z is built via the edge $\{y_{12}, y_{32}\}$, we can find an improving set $\{\langle x_1, y_{21}, y_{22} \rangle, \langle x_3, x_2, y_{31} \rangle, \langle y_{11}, y_{12}, y_{32} \rangle\}$ for \mathcal{P} . Refer to the fourth subfigure in Figure 15.

To summarize, each 2-path receives at least $\frac{1+2\gamma}{3}$ token in average.



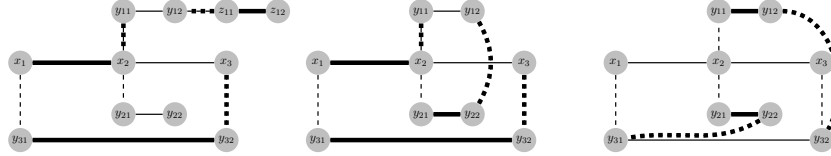
■ **Figure 15** Case 3.2.3 $f = 3$ and $\vec{f} = (2, 1, 1)$. The solid and dashed edges denote the edges in $E(\mathcal{P})$ and $E(\mathcal{P}^*)$, respectively. The thick paths form an improving set.

Case 3.2.4: $\vec{f} = (1, 2, 1)$. Suppose $\mathcal{F}_1 = \{Y_1\}, \mathcal{F}_2 = \{Y_2\}, \mathcal{F}_3 = \{Y_3\}$. We claim X cannot have a special 1-hop-away 2-path friend via $Y_i, i \in \{1, 2, 3\}$. The correctness proof follows from a similar argument in Case 2.2.1 “ $f = 2$ and $\vec{f} = (1, 1, 1)$ ”. By Theorem 11, Y_i has another friend via y_{i2} , denoted as $Z_i, i \in \{1, 2\}$. We claim Z_i is a 3-path in $\mathcal{P}, i \in \{1, 2\}$. Otherwise, let $Z_1 = \{z_{11}, z_{12}\}$.

- If $Z \notin \{Y_2, Y_3\}$, we can find an improving set $\{\langle x_1, x_2, y_{11} \rangle, \langle y_{12}, z_{11}, z_{12} \rangle, \langle x_3, y_{32}, y_{31} \rangle\}$ for \mathcal{P} . Refer to the first subfigure in Figure 16.
- If $Z = Y_2$, the friendship between Y_1 and Z can only be built via the edge $\{y_{12}, y_{22}\}$ and we can find an improving set $\{\langle x_1, x_2, y_{11} \rangle, \langle y_{12}, y_{22}, y_{21} \rangle, \langle x_3, y_{32}, y_{31} \rangle\}$ for \mathcal{P} . Refer to the second subfigure in Figure 16.

- If $Z = Y_3$, the friendship between Y_1 and Z is built via the edge $\{y_{12}, y_{32}\}$ or $\{y_{12}, y_{31}\}$. We consider the first case without loss of generality. By Theorem 11, Y_2 has another friend via y_{22} , denoted as Z' . We claim $Z' \neq Z$. Otherwise, the friendship between Y_2 and Z can only be built via the edge $\{y_{22}, y_{31}\}$. Refer to the third subfigure in Figure 16. We can find an improving set $\{\langle y_{21}, y_{22}, y_{31} \rangle, \langle y_{11}, y_{12}, y_{32} \rangle\}$ for \mathcal{P} .

To summarize, each 2-path receives at least $\frac{1+2\gamma}{3}$ token in average.



■ **Figure 16** Case 3.2.4 $f = 3$ and $\vec{f} = (1, 2, 1)$. The solid and dashed edges denote the edges in $E(\mathcal{P})$ and $E(\mathcal{P}^*)$, respectively. The thick paths form an improving set.

Case 4: $f = 4$.

By Lemma 6, we cannot find three distinct 2-paths Y_i such that $Y_i \in \mathcal{F}_i$, $i \in \{1, 2, 3\}$, which implies $f_1 + f_2 + f_3 < 5$, that is $f_1 + f_2 + f_3 = 4$. Moreover, there exists one $i \in \{1, 2, 3\}$ such that $f_i = 0$. Otherwise, each \mathcal{F}_i contains distinct 2-path friends, which is a contradiction.

Case 4.1: $f = 4$ and $\vec{f} = (0, 2, 2)$. The symmetric case $\vec{f} = (2, 2, 0)$ can be discussed similarly. Suppose $\mathcal{F}_2 = \{Y_1, Y_2\}$ and $\mathcal{F}_3 = \{Y_3, Y_4\}$. Using a similar argument in the Case 2.1.3 “ $f = 2$ and $\vec{f} = (1, 1, 0)$ ”, X cannot have a special 1-hop-away 2-path friend via Y_1 and Y_2 . Besides, using a similar argument in the Case 2.1.1 “ $f = 2$ and $\vec{f} = (2, 0, 0)$ ”, X has at most one special 1-hop-away 2-path friend either via Y_3 or Y_4 , and at least one of Y_3 and Y_4 has a another 3-path friend in \mathcal{P} . We discuss the following cases.

Case 4.1.1: X has a special 1-hop-away 2-path friend via Y_4 without loss of generality. Following in the argument in Case 3.1.1.1, both Y_1 and Y_2 have 3-path friends except for X . There are five 2-paths associated with X . Y_i , $i \in \{1, 2, 3\}$, receives γ token from other 2-paths in \mathcal{P} . Therefore, each 2-path receives at least $\frac{1+3\gamma}{5}$ token in average.

Case 4.1.2: X has no special 1-hop-away 2-path friends. There are four 2-paths associated with X . For at least two paths in $\{Y_i, i \in \{1, 2, 3, 4\}\}$, each receives γ token from other 2-paths in \mathcal{P} . Therefore, each 2-path receives at least $\frac{1+2\gamma}{4}$ token in average.

To summarize, each 2-path associated with X receives at least $\min\{\frac{1+3\gamma}{5}, \frac{1+2\gamma}{4}\}$ token in average.

Case 4.2: $f = 4$ and $\vec{f} = (2, 0, 2)$. Suppose $\mathcal{F}_1 = \{Y_1, Y_2\}$ and $\mathcal{F}_3 = \{Y_3, Y_4\}$. Using a similar argument in the Case 2.1.4 “ $f = 2$ and $\vec{f} = (1, 0, 1)$ ”, X cannot have a special 1-hop-away 2-path friend via Y_i , $i \in \{1, 2, 3, 4\}$. Besides, using a similar argument in the Case 2.1.1 “ $f = 2$ and $\vec{f} = (2, 0, 0)$ ”, at least one of Y_1 and Y_2 (Y_3 and Y_4) has a another 3-path friend in \mathcal{P} . There are four 2-paths associated with X . For at least two paths in $\{Y_i, i \in \{1, 2, 3, 4\}\}$, each receives γ token from other 2-paths in \mathcal{P} . Therefore, each 2-path associated with X receives at least $\frac{1+2\gamma}{4}$ token in average.

Efficiently Realizing Interval Sequences*

Amotz Bar-Noy

City University of New York (CUNY), USA
amotz@sci.brooklyn.cuny.edu

Keerti Choudhary

Weizmann Institute of Science, Rehovot, Israel
keerti.choudhary@weizmann.ac.il

David Peleg

Weizmann Institute of Science, Rehovot, Israel
david.peleg@weizmann.ac.il

Dror Rawitz

Bar Ilan University, Ramat-Gan, Israel
dror.rawitz@biu.ac.il

Abstract

We consider the problem of realizable interval-sequences. An interval sequence comprises of n integer intervals $[a_i, b_i]$ such that $0 \leq a_i \leq b_i \leq n - 1$, and is said to be *graphic/realizable* if there exists a graph with degree sequence, say, $D = (d_1, \dots, d_n)$ satisfying the condition $a_i \leq d_i \leq b_i$, for each $i \in [1, n]$. There is a characterisation (also implying an $O(n)$ verifying algorithm) known for realizability of interval-sequences, which is a generalization of the Erdős-Gallai characterisation for graphic sequences. However, given any realizable interval-sequence, there is no known algorithm for computing a corresponding graphic certificate in $o(n^2)$ time.

In this paper, we provide an $O(n \log n)$ time algorithm for computing a graphic sequence for any realizable interval sequence. In addition, when the interval sequence is non-realizable, we show how to find a graphic sequence having minimum deviation with respect to the given interval sequence, in the same time. Finally, we consider variants of the problem such as computing the most regular graphic sequence, and computing a minimum extension of a length p non-graphic sequence to a graphic one.

2012 ACM Subject Classification Mathematics of computing \rightarrow Graph algorithms; Mathematics of computing \rightarrow Enumeration

Keywords and phrases Graph realization, graphic sequence, interval sequence

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.47

Funding US-Israel BSF grant 2018043; Army Research Laboratory Cooperative Grant, ARL Network Science CTA, W911NF-09- 2-0053.

1 Introduction

The *Graph Realization problem* for a property P deals with the following existential question: Does there exist a graph that satisfies the property P ? Its fundamental importance is apparent, ranging from better theoretical understanding, to network design questions (such as constructing networks with certain desirable connectivity properties). Some very basic, yet challenging, properties that have been considered in past are degree sequences [7, 16, 18], eccentricities [4, 22], connectivity and flow [14, 10, 8, 9].

One of the earliest classical problems studied in this domain is that of *graphic sequences*. A sequence of n positive integers, $D = (d_1, \dots, d_n)$, is said to be *graphic* if there exists an n vertex graph G such that D is identical to the sequence of vertex degrees of G . The problem

* Extended abstract



of realizing graphic sequences and counting the number of non-isomorphic realizations of a given graphic-sequence, is particularly of interest due to many practical applications, see [25] and reference therein. In 1960, Erdős and Gallai [7] gave a characterization (also implying an $O(n)$ verifying algorithm) for graphic sequences. Havel and Hakimi [16, 18] gave a recursive algorithm that given a sequence D of integers computes a realizing graph, or proves that the sequence is non-graphic, in optimal time $O(\sum_i d_i)$. Recently, Tripathi et al. [26] provided a constructive proof of Erdős and Gallai’s [7] characterization.

We consider a generalization of the graphic sequence problem where instead of specifying precise degrees, we are given a *range* (or interval) of possible degree values for each vertex. Formally, an *interval-sequence* is a sequence of n intervals $\mathcal{S} = ([a_1, b_1], \dots, [a_n, b_n])$, also represented as $\mathcal{S} = (A, B)$, where $A = (a_1, \dots, a_n)$ and $B = (b_1, \dots, b_n)$, and $0 \leq a_i \leq b_i \leq n - 1$ for every i . It is said to be *realizable* if there exists a sequence $D = (d_1, \dots, d_n)$ that is graphic and satisfies the condition $a_i \leq d_i \leq b_i$, for $1 \leq i \leq n$. Two questions that are natural to ask here are:

► **Question 1 (Verification).** *Find an efficient algorithm for verifying the realizability of any given interval-sequence \mathcal{S} ?*

► **Question 2 (Graphic Certificate).** *Given a realizable interval-sequence \mathcal{S} , compute a certificate (that is, a graphic sequence D) realizing it.*

Cai et al. [5] extended Erdős and Gallai’s work by providing an easy to verify characterization for realizable interval-sequences, thereby resolving Question 1. Their result crucially uses the (g, f) -Factor Theorem of Lovász [23]. Garg et al. [13] provided a constructive proof of the characterisation of Cai et al. [5] for realizable interval sequences. In [20], Hell and Kirkpatrick provided an algorithm based on Havel and Hakimi’s work for computing a graph that realizes an interval sequence (if exists). For non-realizable interval sequences \mathcal{S} , their algorithm computes a graph whose deviation $\delta(D, \mathcal{S})$ (see Section 2 for definition) with respect to L1-norm is minimum. The time complexity of their algorithm is $O(\sum_{i=1}^n b_i)$ (which can be as high as $\Theta(n^2)$).

Our Contributions. In this paper we introduce a new approach for representing and analyzing the interval sequence realization problem. Our algorithms are based on a novel divide and conquer methodology, wherein we show that partitioning a realizable interval sequence along any levelled sequence (a new class of sequences introduced herein) guarantees that at least one of the new child interval sequences is also realizable. This enables us to present an $O(n \log n)$ time algorithm for computing a graphic certificate (if exists) for a given interval sequence. While the problem was well studied, to the best of our knowledge there was no known $o(n^2)$ time algorithm for computing graphic certificate. Also, there was no sub-quadratic time algorithm known for computing even the deviation $\delta(D, \mathcal{S})$. Specifically, we obtain the following result.

► **Theorem 1.** *There exists an algorithm that for any integer $n \geq 1$ and any length n interval sequence \mathcal{S} , computes a graphic sequence D realizing \mathcal{S} , if exists, in $O(n \log n)$ time.*

Moreover, when \mathcal{S} is non-realizable, the algorithm outputs in same time a graphic sequence D minimizing the deviation $\delta(D, \mathcal{S})$.

Our new approach enables us to tackle also an optimization version of the problem in which it is required to compute the “most regular” sequence realizing the given interval sequence \mathcal{S} , using the natural measure of the minimum sum of pairwise degree differences, $\sum_{i,j} |d_i - d_j|$, as our regularity measure. To the best of our knowledge, this problem was not studied before and is not dealt with directly by the existing approaches to the interval sequence problem. Specifically, we obtain the following.

► **Theorem 2.** *There exists an algorithm that for any integer $n \geq 1$ and any length n realizable interval sequence \mathcal{S} , computes the most regular graphic sequence realizing interval sequence \mathcal{S} (i.e., the one minimizing the sum of pairwise degree difference), in time $O(n^2)$.*

The tools developed in this paper allows us to study other interesting applications, such as computing a minimum extension of non-graphic sequences to graphic ones (see Section 6).

Related work. Kleitman and Wang [21], and Fulkerson-Chen-Anstee [2, 6, 12] solved the problem of degree realization for directed graphs, wherein, for each vertex both the in-degree and out-degree is specified. In [17], Nichterlein and Hartung proved the NP-completeness of the problem when the additional constraint of acyclicity is imposed. Over the years, various extensions of the degree realization problems were studied as well, cf. [1, 28]. The *Subgraph Realization problem* considers the restriction that the realizing graph must be a subgraph (*factor*) of some fixed input graph. For an interesting line of work on graph factors, refer to [27, 3, 19, 15]. The subgraph realization problems are generally harder. For instance, it is very easy to compute an n -vertex connected graph whose degree sequence consists of all values 2, however, the same problem for subgraph-realization is NP-hard (since it reduces to Hamiltonian-cycle problem).

Lesniak [22] provided a characterization for the sequence of eccentricities of an n -vertex graph. Behzad et al. [4] studied the problem of characterizing the set comprising of vertex-eccentricity values of general graphs (the sequence problem remains open). Fujishige et al. [11] considered the problem of realizing graphs and hypergraphs with given cut specifications.

Organization of the Paper. In Section 2, we present the notation and definitions. In Section 3, we discuss the main ideas and tools that help us to construct graph certificates for interval sequence problem. Section 4 presents our $O(n \log n)$ time algorithm for computing graphic certificate with minimum deviation. Section 5 provides a quadratic-time algorithm for computing the most regular certificate. We discuss the applications in Section 6.

2 Preliminaries

A *sequence* is defined to be an n -element vector whose entries are non-negative integers. For any sequence $D = (d_1, \dots, d_n)$, define $\min(D) = \min_{i=1}^n \{d_i\}$, $\max(D) = \max_{i=1}^n \{d_i\}$, $\text{sum}(D) = \sum_{i=1}^n d_i$, and $\text{parity}(D) = \text{sum}(D) \bmod 2$. Given any two sequences $X = (x_1, \dots, x_n)$ and $Y = (y_1, \dots, y_n)$, we say that $X \leq Y$ if $x_i \leq y_i$ for $1 \leq i \leq n$. Any two sequences X and Y are said to be *similar* if they are identical up to permutation of the elements (i.e., their sorted versions are identical). A sequence D is said to *lie* in an interval-sequence (A, B) , denoted by $D \in (A, B)$, if $A \leq D \leq B$. We define $\min(X, Y) = (\min\{x_1, y_1\}, \dots, \min\{x_n, y_n\})$, and $\max(X, Y) = (\max\{x_1, y_1\}, \dots, \max\{x_n, y_n\})$. The \mathbf{L}_1 -distance of the pair (X, Y) is defined as $\mathbf{L}_1(X, Y) = \sum_{i=1}^n |y_i - x_i|$.

Denote by \top and \perp the n -length sequences all whose entries are respectively $n - 1$ and 0 . Given a sequence $D = (d_1, \dots, d_n)$ and an integer $k \in [1, n]$, define the vectors $X(D)$ and $Y(D)$ by setting for $1 \leq k \leq n$:

$$X_k(D) \triangleq \sum_{i=1}^k d_i, \quad \text{and} \quad Y_k(D) \triangleq k(k-1) + \sum_{i=k+1}^n \min(d_i, k).$$

For any sequence $D = (d_1, \dots, d_n)$, the *spread* of D is defined as $\phi(D) = \sum_{1 \leq r < s \leq n} |d_r - d_s|$, and it always lies in the range $[0, n^3]$. A sequence D is said to be more *regular* than another sequence D' if $\phi(D) < \phi(D')$. For any two integers $x \leq y$, $[x, y] = \{x, x + 1, \dots, y\}$. For any

47:4 Efficiently Realizing Interval Sequences

$I \subseteq [1, n]$, define $D[I]$ to be the subsequence of D consisting of elements d_i , for $i \in I$; and define E_I to be the *characteristic vector* of I , namely, the sequence (e_1, e_2, \dots, e_n) such that $e_i = 1$ if $i \in I$, and $e_i = 0$ otherwise. For any sequence $D = (d_1, \dots, d_n)$ and an interval-sequence $\mathcal{S} = ([a_1, b_1], \dots, [a_n, b_n])$, the upper and lower deviation of D , is respectively defined as

$$\delta_U(D, \mathcal{S}) = \sum_{i=1}^n \max\{0, (d_i - b_i)\}, \quad \text{and} \quad \delta_L(D, \mathcal{S}) = \sum_{i=1}^n \max\{0, (a_i - d_i)\}.$$

The *deviation* of D is defined as $\delta(D, \mathcal{S}) = \delta_U(D, \mathcal{S}) + \delta_L(D, \mathcal{S})$. For any vertex x in an undirected simple graph H , define $\deg_H(x)$ to be the degree of x in H , and define $N_H(x) = \{y \mid (x, y) \in E(H)\}$ to be the neighbourhood of x in H .

We next state the Erdős and Gallai [7] characterisation for realizable(graphic) sequences, and Cai et al. [5] characterisation for realizable interval sequences. An $O(n)$ -time implementation of the both theorems is provided in the full version.

► **Theorem 3** (Erdős and Gallai [7]). *A non-increasing sequence $D = (d_1, \dots, d_n)$ is graphic if and only if*

- (i) $X_n(D)$ is even, and
- (ii) $X(D) \leq Y(D)$.

► **Theorem 4** (Cai et al. [5]). *Let $\mathcal{S} = ([a_1, b_1], \dots, [a_n, b_n]) = (A, B)$ be an interval-sequence such that A is non-increasing and for any index $1 \leq i < n$, $b_{i+1} \leq b_i$ whenever $a_i = a_{i+1}$. For each $k \in [1, n]$, define $W_k(\mathcal{S}) = \{i \in [k+1, n] \mid b_i \geq k+1\}$. Then \mathcal{S} is realizable if and only if $X(A) \leq Y(B) - \varepsilon(\mathcal{S})$, where, $\varepsilon(\mathcal{S})$ is defined by setting*

$$\varepsilon_k(\mathcal{S}) = \begin{cases} 1 & \text{if } a_i = b_i \text{ for } i \in W_k(\mathcal{S}) \text{ and } \sum_{i \in W_k(\mathcal{S})} (b_i + k |W_k(\mathcal{S})|) \text{ is odd,} \\ 0 & \text{otherwise.} \end{cases}$$

3 Main Tools

In this section, we develop some crucial tools that help us in efficient computation of certificate for a realizable interval-sequence. These tools will help us to search a graphic sequence in $O(n \log n)$ time using a clever divide and conquer methodology. Also they aid in searching for the maximally regular sequence in just quadratic time.

Levelling operation. Given a sequence $D = (d_1, \dots, d_n)$ and a pair of indices $\alpha \neq \beta$ satisfying $d_\alpha > d_\beta$, we define $\pi(D, \alpha, \beta) = D^* = (d_1^*, \dots, d_n^*)$ to be a sequence obtained from D by decrementing d_α by 1 and incrementing d_β by 1 (i.e., $d_\alpha^* = d_\alpha - 1$, $d_\beta^* = d_\beta + 1$, and $d_k^* = d_k$ for $k \neq \alpha, \beta$). This operation is called the *levelling operation* on D for the indices α and β . The operation essentially “levels” (or “flattens”) the sequence D , making it more uniform.

We now discuss some properties of levelling operations.

► **Lemma 5.** *Any levelling operation on a sequence D that results in a non-similar sequence, reduces its spread $\phi(D)$ by a value at least two.*

Proof. Let $D = (d_1, d_2, \dots, d_n)$ and $Z = (z_1, \dots, z_n) = \pi(D, \alpha, \beta)$, be a sequence obtained from D by performing a levelling operation on a pair of indices α, β such that $d_\alpha > d_\beta$. If $d_\alpha = d_\beta + 1$, then it is easy to verify that D and Z are similar. If $d_\alpha \geq d_\beta + 2$, then

$$\begin{aligned} \phi(Z) &= |z_\alpha - z_\beta| + \sum_{s \neq \alpha, \beta} (|z_\alpha - z_s| + |z_\beta - z_s|) + \sum_{\substack{1 \leq r < s \leq n, \\ r, s \notin \{\alpha, \beta\}}} |z_r - z_s| \\ &= |d_\alpha - d_\beta| - 2 + \sum_{\substack{s \neq \alpha, \beta \text{ s.t.} \\ d_s \notin \{d_\beta, d_\alpha\}}} (|d_\alpha - d_s| + |d_\beta - d_s|) \\ &\quad + \sum_{\substack{s \neq \alpha, \\ \beta \text{ s.t. } d_s \in \{d_\beta, d_\alpha\}}} (|d_\alpha - d_s| + |d_\beta - d_s| - 2) + \sum_{\substack{1 \leq r < s \leq n, \\ r, s \notin \{\alpha, \beta\}}} |d_r - d_s| \\ &\leq \left(\sum_{1 \leq r < s \leq n} |d_r - d_s| \right) - 2 = \phi(D) - 2. \end{aligned}$$

Thus, the claim follows. ◀

► **Lemma 6** (Corollary 3.1.4, [24]). *The levelling operations preserves graphicity, that is, if we perform a levelling operation on a graphic sequence, then the resulting sequence is also graphic.*

Proof. Let $D = (d_1, \dots, d_n)$ be a graphic sequence, and $\pi(D, \alpha, \beta) = D^* = (d_1^*, \dots, d_n^*)$ for some indices α, β satisfying $d_\alpha > d_\beta$. If $d_\alpha = 1 + d_\beta$, then D^* is similar to D , and thus also graphic. So for the rest the proof let us focus on the case $d_\alpha \geq 2 + d_\beta$. Let $G = (V, E)$ be a graph realising the sequence D , and let x_α and x_β be two vertices in G having degrees respectively d_α and d_β . Since $|N_G(x_\alpha)| \geq 2 + |N_G(x_\beta)|$, there must exist at least one neighbour, say w , of vertex x_α that does not lie in set $\{x_\beta\} \cup N_G(x_\beta)$. Let $G^* = (V, E^*)$ be a graph obtained from G by deleting the edge (w, x_α) , and adding a new edge (w, x_β) . Observe that the degree of all vertices other than x_α and x_β are identical in graphs G and G^* , also $\deg_{G^*}(x_\alpha) = \deg_G(x_\alpha) - 1$, and $\deg_{G^*}(x_\beta) = \deg_G(x_\beta) + 1$. Therefore G^* is a graph realising the profile D^* , and thus the claim follows. ◀

Levelled sequences. A sequence D is said to be *levelled with respect to the integer-sequence* $\mathcal{S} = (A, B)$ if

- (i) $A \leq D \leq B$, and
- (ii) the spread of D cannot be decreased by a levelling operation, i.e., for any two indices $\alpha \neq \beta$ satisfying $d_\alpha > d_\beta$ and $A \leq \pi(D, \alpha, \beta) \leq B$, we have $\phi(\pi(D, \alpha, \beta)) = \phi(D)$.

See Figure 1.

The *volume* of a sequence D lying between A and B with respect to $\mathcal{S} = (A, B)$ is defined as

$$\text{VOL}(D, \mathcal{S}) \triangleq \mathbf{L}_1(D, A),$$

and is invariant of levelling operations applied to D . In other words, applying a levelling operation to a sequence D may reduce its spread but preserves its volume. Note that the volume lies in the range $[0, \mathbf{L}_1(A, B)]$.

► **Lemma 7.** *For any $\mathcal{S} = (A, B)$, a sequence D satisfying $A \leq D \leq B$ can be transformed into a levelled sequence D^* having the same volume $\text{VOL}(D, \mathcal{S})$ by a repeated application of (at most $O(n^3)$) levelling operations.¹*

Proof. By Lemma 5, every levelling operation that results in a new (non-similar) sequence decreases the spread by at least two. Since the spread of any sequence D is always non-negative and finite (specifically, $O(n^3)$), it is possible to perform ($O(n^3)$) levelling operations on D so that the resultant sequence D^* is levelled. Since the levelling operation preserves the volume, $\text{VOL}(D^*, \mathcal{S})$ must be same as $\text{VOL}(D, \mathcal{S})$. ◀

Any graphic sequence D realizing the interval sequence $\mathcal{S} = (A, B)$ by Lemma 7 can be altered by $O(n^3)$ levelling operations to obtain a levelled sequence lying between A and B . The resultant sequence by Lemma 6 remains graphic, thus the following theorem is immediate.

► **Theorem 8.** *For any realizable interval sequence $\mathcal{S} = (A, B)$ there exists a graphic sequence realizing \mathcal{S} which is a levelled sequence.*

Characterizing and Computing Levelled sequences. Given any interval sequence $\mathcal{S} = (A, B)$ and a real number $\ell \in [\min(A), \max(B)]$, let²

$$F(\ell, \mathcal{S}) \triangleq \sum_{i \in [1, n]} (\min\{\ell, b_i\} - \min\{\ell, a_i\}) .$$

Observe that $F(\cdot, \mathcal{S})$ is a non-decreasing function in the range $(\min(A), \max(B))$. Hence we may define the corresponding inverse function as $F^{-1}(L, \mathcal{S}) = \min\{\ell \mid F(\ell, \mathcal{S}) = L\}$.

Given any interval sequence $\mathcal{S} = (A, B)$, we define $I(\ell, \mathcal{S}) \triangleq \{i \in [1, n] \mid a_i < \ell < b_i\}$.

We conclude this section by providing the following theorems for characterising and computing levelled sequences (proofs are deferred to the full version).

► **Theorem 9.** *Consider an interval sequence $\mathcal{S} = (A, B)$. Let L be an integer in $[0, \mathbf{L}_1(A, B)]$ and $\ell \geq 0$ be such that $\ell = F^{-1}(L, \mathcal{S})$. Then the collection of levelled sequences that have volume L with respect to \mathcal{S} is equal to the collection of sequences $D = (d_1, \dots, d_n)$ satisfying the following three conditions:*

- (a) $d_i = b_i$ for any i satisfying $b_i \leq \ell$;
- (b) $d_i = a_i$ for any i satisfying $a_i \geq \ell$; and
- (c) Among all indices lying in set $I(\ell, \mathcal{S})$, exactly $F(\ell, \mathcal{S}) - F(\lfloor \ell \rfloor, \mathcal{S})$ indices i satisfy $d_i = \lfloor \ell \rfloor$, and the remaining indices i satisfy $d_i = \lceil \ell \rceil$.

► **Theorem 10.** *Given an interval sequence $\mathcal{S} = (A, B)$ consisting of n -pairs, and an integer $L \in [0, \mathbf{L}_1(A, B)]$, a levelled sequence D having volume L with respect to \mathcal{S} can be computed in $O(n)$ time.*

¹ We remark that the algorithms presented later on generate a desired levelled sequence using more efficient methods than the one implicit in the proof, and are therefore faster.

² One can think of \mathcal{S} as representing a collection of n connected vessels, each in the shape of a unit column closed at both ends, then $F(\ell, \mathcal{S})$ is the amount of fluid that will fill this connected vessel system to level ℓ .

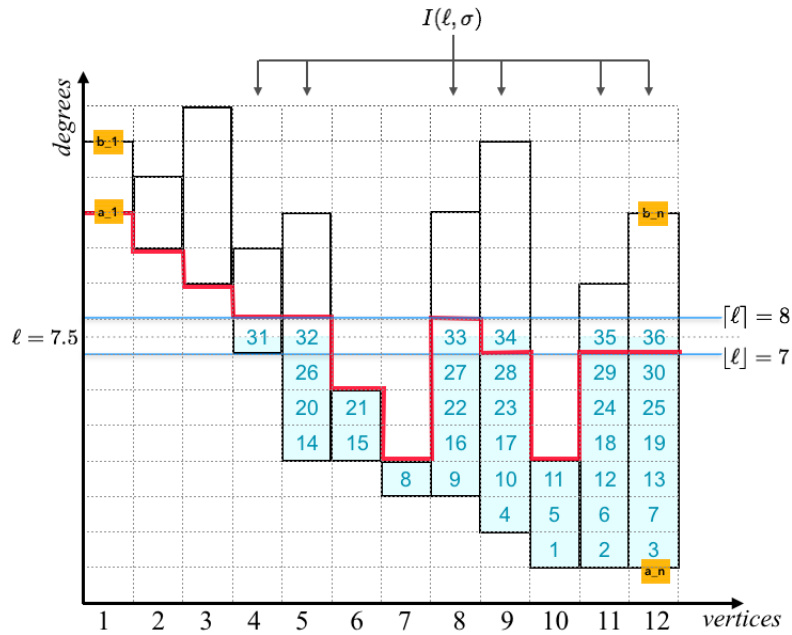


Figure 1 Illustration of a levelled sequence D (in red) satisfying $L = \text{VOL}(D, \mathcal{S}) = 33$. For $\ell = 7.5$, $F(\ell = 7.5, \mathcal{S}) = 33$, $F(\lfloor \ell \rfloor = 7, \mathcal{S}) = 30$, and $F(\lceil \ell \rceil = 8, \mathcal{S}) = 36$. The segments contributing to $F(\ell = 7.5, \mathcal{S})$, i.e., the parts of the connected vessel system filled with fluid, are shown in blue. The values in D at all indices in set $I(\ell, \mathcal{S})$ differ by at most one as they lie in the set $\{\lfloor \ell \rfloor, \lceil \ell \rceil\}$.

4 An $O(n \log n)$ time algorithm for Graphic Certificate

In this section, we present an algorithm for computing a certificate for interval sequence that takes just $O(n \log n)$ time. If the input interval $\mathcal{S} = (A, B)$ is realizable, our algorithm computes a graphic sequence $D \in \mathcal{S}$, otherwise it computes a sequence minimizing the deviation value $\delta(D, \mathcal{S})$. We begin by considering the case where the sequence \mathcal{S} is realizable (since it is simpler to understand given Theorems 9 and 10), and then we move to the case where \mathcal{S} is non-realizable. Then characterization of [5] implies an $O(n)$ time verification algorithm for realizability of interval sequence. (For details refer to the full version).

4.1 Realizable Interval Sequences

First we show that any two levelled sequences after an appropriate reordering of their elements are coordinate-wise comparable.

► **Lemma 11.** *For any interval sequence $\mathcal{S} = (A, B)$, and any two levelled sequences $C, D \in \mathcal{S}$ satisfying $\text{VOL}(D, \mathcal{S}) \leq \text{VOL}(C, \mathcal{S})$, the following holds.*

1. $D' \leq C$, for some sequence $D' \in \mathcal{S}$ similar to D .
2. $D \leq C''$, for some sequence $C'' \in \mathcal{S}$ similar to C .

Proof. We show how to transform $D = (d_1, \dots, d_n)$ into sequence $D' = (d'_1, \dots, d'_n) \in \mathcal{S}$ such that $D' \leq C$. Let $\ell_D = F^{-1}(\text{VOL}(D, \mathcal{S}), \mathcal{S})$ and $\ell_C = F^{-1}(\text{VOL}(C, \mathcal{S}), \mathcal{S})$. Since $F(\cdot, \mathcal{S})$ is a non-decreasing function, we have that $\ell_D \leq \ell_C$.

Let us first consider the case where ℓ_C and ℓ_D are both non-integral, and $\lfloor \ell_C \rfloor = \lfloor \ell_D \rfloor =$ (say ℓ_1) and $\lceil \ell_C \rceil = \lceil \ell_D \rceil =$ (say ℓ_2). By Theorem 9, for any index $i \in [1, n]$,

- (i) $a_i \geq \ell_D$ (or $a_i \geq \ell_C$) implies $d_i = a_i = c_i$;
- (ii) $b_i \leq \ell_D$ (or $b_i \leq \ell_C$) implies $d_i = b_i = c_i$.

Also, among indices in set $I_0 = I(\ell_D, \mathcal{S}) = I(\ell_C, \mathcal{S})$,

- (i) exactly $L_D - F(\lfloor \ell_D \rfloor, \mathcal{S})$ indices i satisfy $d_i = \ell_2$ (let I_D denote the set of these indices) and the remaining indices i satisfy $d_i = \ell_1$;
- (ii) exactly $L_C - F(\lfloor \ell_C \rfloor, \mathcal{S})$ indices i satisfy $c_i = \ell_2$ (let I_C denote the set of these indices) and the remaining indices i satisfy $c_i = \ell_1$.

Since $L_D \leq L_C$, it follows that $|I_D| \leq |I_C|$, however, observe that I_D need not be a subset of I_C . We set D' to be the sequence that satisfy the condition that

- (i) $d'_i = d_i$, for each $i \notin I_0$, and
- (ii) for indices in I_0 , at any arbitrary $|I_D|$ indices lying in I_C , d'_i take the value ℓ_2 , and at remaining $|I_0| - |I_D|$ indices d'_i take the value ℓ_1 .

It is easy to verify that D and D' are similar, and $D' \leq C$.

The remaining case is when $\lceil \ell_D \rceil \leq \lfloor \ell_C \rfloor$. For any index $i \in [1, n]$, $d_i \leq \lceil \ell_D \rceil$ and $c_i \geq \lfloor \ell_C \rfloor$, implies $d_i \leq c_i$. Observe that by Theorem 9,

- (i) for an index i , $d_i > \lceil \ell_D \rceil$ implies $d_i = a_i (\leq c_i)$; and
- (ii) for an index i , $c_i < \lfloor \ell_C \rfloor$ implies $c_i = b_i (\geq d_i)$.

Therefore, for each index i , $d_i \leq c_i$. So in this case, we set D' to be D . The construction of sequence C'' follows similarly. \blacktriangleleft

Next lemma shows significance of partitioning an interval-sequence using a levelled sequence.

► Lemma 12. *Let C and D be any two levelled sequences lying in an interval sequence $\mathcal{S} = (A, B)$, and having volume L_C and L_D , respectively. Also assume D is a graphic sequence. Then,*

- (a) $L_D \leq L_C$ implies (A, C) is a realizable interval sequence.
- (b) $L_D \geq L_C$ implies (C, B) is a realizable interval sequence.

Proof. We provide proof of the case $L_D \leq L_C$ (the proof of part (b) will follow in a similar fashion). By Lemma 11, we can transform $D = (d_1, \dots, d_n)$ into another levelled sequence $D' = (d'_1, \dots, d'_n) \in \mathcal{S}$ such that D' is similar to D and $D' \leq C$. Since $D' \leq C$, and D' is a graphic sequence, it follows that (A, C) is realizable interval sequence. \blacktriangleleft

From Lemma 12, and the fact that each realizable interval-sequence contains a levelled graphic sequence (see Theorem 8), we obtain following.

► Theorem 13. *For any realizable interval sequence $\mathcal{S} = (A, B)$, and any levelled sequence $C \in \mathcal{S}$, at least one of the interval-sequences (A, C) and (C, B) is realizable.*

The above theorem provides a divide-and-conquer strategy to search for a levelled graphic sequence for realizable interval-sequences as shown in Algorithm 1. Let (A_0, B_0) be initialized to (A, B) . We compute a levelled sequence C_0 having volume $\lfloor \mathbf{L}_1(A_0, B_0)/2 \rfloor$ using Theorem 9. It follows from Theorem 13, either (A_0, C_0) or (C_0, B_0) must be a realizable interval-sequence. If (A_0, C_0) is realizable then we replace B_0 by C_0 ; otherwise (C_0, B_0) must be realizable, so we replace A_0 by C_0 . We continue this process (of replacements) until $\mathbf{L}_1(A_0, B_0)$ decreases to a value smaller than 2. In the end, the interval sequence (A_0, B_0) contains at most two sequences, namely A_0 and B_0 . If A_0 is graphic then we return A_0 , otherwise we return B_0 . The correctness of the algorithm is immediate from the description.

Algorithm 1 CERTIFICATE-REALIZABLE(A, B).

```

1 Initialize interval sequence  $(A_0, B_0)$  to  $(A, B)$ ;
2 while  $\mathbf{L}_1(A_0, B_0) \geq 2$  do
3    $C_0 \leftarrow$  a levelled sequence of volume  $\lfloor \mathbf{L}_1(A_0, B_0)/2 \rfloor$ ;
4   if (Interval-sequence  $(A_0, C_0)$  is realizable) then  $B_0 \leftarrow C_0$ ;
5   else  $A_0 \leftarrow C_0$ ;
6 if  $A_0$  is graphic then Return  $A_0$ ;
7 else Return  $B_0$ ;

```

To analyze the running time, observe that the \mathbf{L}_1 -distance between A_0 and B_0 decreases by (roughly) a factor of 2 in each call of the while loop, so it follows that number of iterations is $O(\log n)$. Verifying if an interval sequence is realizable, or a sequence D is graphic can be performed in $O(n)$ time. Also in $O(n)$ time we can generate a levelled sequence of any given volume L by Theorem 10. Thus, the total time complexity of the algorithm is $O(n \log n)$.

We obtain the following result:

► **Theorem 14.** *There exists an algorithm that for any integer $n \geq 1$ and any n -length interval sequence $\mathcal{S} = (A, B)$, computes a graphic sequence $D \in (A, B)$, if it exists, in $O(n \log n)$ time.*

4.2 Non-Realizable Sequences

In this subsection we consider the scenario where \mathcal{S} is non-realizable, our goal is to compute a graphic sequence D minimizing the deviation $\delta(D, \mathcal{S})$ with respect to the given interval sequence \mathcal{S} .

As a first step, we show that in order to search a sequence D minimizing $\delta(D, \mathcal{S})$, it suffices to search a sequence $D \geq A$ that minimizes the value $\delta_U(D, \mathcal{S})$.

► **Lemma 15.** $\min\{\delta(D, \mathcal{S}) \mid D \text{ is graphic}\} = \min\{\delta_U(D, \mathcal{S}) \mid D \text{ is graphic, } D \geq A\}$, for any interval sequence $\mathcal{S} = (A, B)$.

Proof. Let $D = (d_1, \dots, d_n)$ be a graphic sequence minimizing the value $\delta(D, \mathcal{S})$, and in case of ties take that D for which $\delta_L(D, \mathcal{S})$ is the lowest. Let us suppose there exists an index $i \in [1, n]$ such that $d_i < a_i$. Consider the graph G realizing the sequence D , and let v_i denote the i th vertex of G , so that, $\deg(v_i) = d_i$. Observe that $|N_G(v_i)| \neq n - 1$, since $d_i < a_i \leq n - 1$. For any vertex $v_j \notin N_G(v_i)$, $d_j = \deg(v_j)$ must be at least b_j , because otherwise adding (v_i, v_j) to G reduces $\delta(D, \mathcal{S})$. Thus for any vertex $v_j \notin N_G(v_i)$, adding (v_i, v_j) to G , decreases $\delta_L(D, \mathcal{S})$ and increases $\delta_U(D, \mathcal{S})$ by a value exactly 1. However, by our choice D was a sequence minimizing $\delta_L(D, \mathcal{S})$, thus $\delta_L(D, \mathcal{S})$ must be zero. The claim follows from the fact that $D \geq A$ and $\delta(D, \mathcal{S}) = \delta_U(D, \mathcal{S})$. ◀

By the previous lemma, our goal is to find a graphic sequence D in the interval sequence (A, \top) minimizing $\delta(D, \mathcal{S})$. Notice that if D is graphic, then the interval sequence (A, R) , where $R = \max(D, B)$, is realizable. Also, $\delta(D, \mathcal{S}) = \text{sum}(R - B)$. Hence, in order to compute a graphic sequence with minimum deviation, we define \mathcal{R} to be the set of all sequence $R \in [B, \top]$ such that

- (i) the interval sequence (A, R) is realizable, and
- (ii) $\text{sum}(R - B)$ is minimized.

47:10 Efficiently Realizing Interval Sequences

The following lemma shows the significance of the set \mathcal{R} in computing a certificate with minimum deviation.

► **Lemma 16.** *For any $R \in \mathcal{R}$, and any graphic sequence D_0 lying in the interval sequence (A, R) , we have $\delta(D_0, \mathcal{S}) = \min\{\delta(D, \mathcal{S}) \mid D \text{ is graphic}\} = \text{sum}(R - B)$.*

Proof. Let D^* be a graphic sequence minimizing the value $\delta(D, \mathcal{S})$. By Lemma 15, we may assume that D^* belongs to (A, \top) . Observe that $\delta(D^*, \mathcal{S}) = \text{sum}(R^* - B)$, where $R^* = \max\{B, D^*\}$. By the choice of D^* we have that $\text{sum}(R^* - B) = \delta(D^*, \mathcal{S}) \leq \delta(D_0, \mathcal{S}) = \delta_U(D_0, \mathcal{S}) \leq \text{sum}(R - B)$, where the last inequality follows from the fact that $D_0 \in (A, R)$. By definition of \mathcal{R} , we have that $\text{sum}(R^* - B) \geq \text{sum}(R - B)$, and therefore $\delta(D^*, \mathcal{S}) = \delta(D_0, \mathcal{S}) = \text{sum}(R - B) = \text{sum}(R^* - B)$. Thus R^* also lies in the set \mathcal{R} . The lemma follows from the fact that $\delta(D^*, \mathcal{S}) = \min\{\delta(D, \mathcal{S}) \mid D \text{ is graphic}\}$. ◀

Next, let \mathcal{R}_L be the set of all levelled sequences in \mathcal{R} with respect to interval sequence (B, \top) .

► **Lemma 17.** $\mathcal{R}_L \neq \emptyset$.

Proof. Clearly, $\mathcal{R} \neq \emptyset$. Consider any sequence $R = (r_1, \dots, r_n) \in \mathcal{R}$. Suppose there exists $\alpha, \beta \in [1, n]$ such that $r_\alpha - r_\beta \geq 1$ and $R' = \pi(R, \alpha, \beta) \in [B, \top]$. Observe that $\text{sum}(R' - B) = \text{sum}(R - B)$. It remains to show that (A, R') is realizable. Indeed, if $D \in (A, R)$ is a graphic sequence, then either

- (i) $D = (d_1, \dots, d_n)$ lies in (A, R') , or
- (ii) $d_\alpha - d_\beta \geq 1$ and $D' = \pi(D, \alpha, \beta)$ lies in (A, R') .

Since levelling operation preserves graphicity, D' is graphic. Thus $R' \in \mathcal{R}$, which shows that \mathcal{R} is closed under the levelling operation, and hence \mathcal{R}_L is non-empty. ◀

■ **Algorithm 2** CERTIFICATE-NON-REALIZABLE(A, B).

```

1  $(M_1, M_2) \leftarrow (B, \top)$ ;
2 while  $\mathbf{L}_1(M_1, M_2) \geq 2$  do
3    $M_0 \leftarrow$  a levelled sequence of volume  $\lfloor \mathbf{L}_1(M_1, M_2)/2 \rfloor$ ;
4   if (Interval-sequence  $(A, M_0)$  is realizable) then  $M_2 \leftarrow M_0$ ;
5   else  $M_1 \leftarrow M_0$ ;
6 if  $(A, M_1)$  is realizable then  $R \leftarrow M_1$ ;
7 else  $R \leftarrow M_2$ ;
8 Return CERTIFICATE-REALIZABLE( $A, R$ )

```

We now describe the algorithm for computing a graphic sequence with minimum deviation (refer to Algorithm 2 for a pseudocode). Recall that we assume that (A, B) is a non-realizable interval sequence. The first step is to compute a levelled sequence $R \in \mathcal{R}_L$, and the second is to use Algorithm 1 to find a graphic sequence in (A, R) .

We initialize two sequences M_1 and M_2 , resp., to B and \top , and these sequences serve as lower and upper boundaries for sequence R . The pair (M_1, M_2) is updated as long as $\text{sum}(M_2 - M_1) \geq 2$ as follows. We compute a levelled sequence M_0 having volume $\lfloor \mathbf{L}_1(M_1, M_2)/2 \rfloor$ with respect to the interval sequence (M_1, M_2) using Theorem 10. There are two cases:

Case 1. (A, M_0) is realizable.

Consider any sequence $R \in (M_1, M_2)$ that lies in \mathcal{R}_L . Since (A, M_0) is realizable, from the definition of R it follows that $\text{sum}(R - B) \leq \text{sum}(M_0 - B)$. As R and M_0 both belong to (M_1, M_2) , by Lemma 11, there exists a sequence R_0 similar to R lying in interval $(M_1, M_2) \subseteq (B, \top)$ such that $R_0 \leq M_0$. It is easy to check that $R_0 \in \mathcal{R}_L$, thus the search range of R which was (M_1, M_2) can be narrowed down to (M_1, M_0) , so we reset M_2 to M_0 .

Case 2. (A, M_0) is not realizable.

Consider any $R \in \mathcal{R}_L$, we first show that $\text{sum}(R - B) > \text{sum}(M_0 - B)$. Let us assume on the contrary, $\text{sum}(R - B) \leq \text{sum}(M_0 - B)$. In such a case, by Lemma 11, there exists a sequence R' similar to R lying in interval $(M_1, M_2) \subseteq (B, \top)$ such that $R' \leq M_0$. Also $R' \in \mathcal{R}_L$. Since, by definition of \mathcal{R}_L , (A, R') is realizable, it violates the fact that (A, M_0) is not realizable. Now as R, M_0 both belong to (M_1, M_2) , by Lemma 11, there exists a sequence R_0 similar to R lying in interval $(M_1, M_2) \subseteq (B, \top)$ such that $R_0 \geq M_0$. Also $R_0 \in \mathcal{R}_L$, thus the search range of R can be narrowed down to (M_0, M_2) , so we reset M_1 to M_0 .

We continue the process of shrinking the range (M_1, M_2) until $\mathbf{L}_1(M_1, M_2)$ decreases to a value smaller than 2. Finally there exists in range (M_1, M_2) at most two sequences, namely M_1 and M_2 . If (A, M_1) is graphic then we set R to M_1 , otherwise we set R to M_2 .

The running time analysis is similar to the one for Algorithm 1. Since the \mathbf{L}_1 -distance between M_1 and M_2 decreases by a factor of 2 in each successive call of the while loop of the algorithm, it follows that number of times the while loops run is $O(\log n)$. Verifying if an interval sequence is realizable, or a sequence D is graphic can be performed in $O(n)$ time. Also it takes $O(n)$ time to generate a levelled sequence of any given volume L by Theorem 10. Finally, the running time of Algorithm 1 is $O(n \log n)$. Thus, the total time complexity of algorithm is $O(n \log n)$.

This completes the proof of Theorem 1.

5 Most Regular Certificate in $O(n^2)$ time

In this section, we present an $O(n^2)$ -time algorithm for computing a most-regular certificate with respect to a given interval sequence $\mathcal{S} = (A, B)$. We assume that \mathcal{S} is realizable. Our algorithm involves a subroutine that given an integer $z \in [\min(A), \max(B) - 1]$, computes a most-regular graphic-sequence, say D , satisfying the condition $z \leq \ell = F^{-1}(\text{VOL}(D, \mathcal{S}), \mathcal{S}) \leq z + 1$. The following lemma is immediate from Theorem 9.

► **Lemma 18.** *Any levelled sequence $\bar{D} = (\bar{d}_1, \dots, \bar{d}_n)$ of volume L with respect to interval sequence $\mathcal{S} = (A, B)$, satisfies $z \leq \ell = F^{-1}(L, \mathcal{S}) \leq z + 1$ if and only if $\bar{d}_i = a_i$ for $a_i \geq z + 1$, $\bar{d}_i = b_i$ for $b_i \leq z$, and $\bar{d}_i \in \{z, z + 1\}$ for remaining indices i .*

We partition the set $[1, n]$ into three sets I_1, I_2, I_3 such that $I_1 = \{i \in [1, n] \mid a_i \geq z + 1\}$, $I_2 = \{i \in [1, n] \mid a_i \leq z \text{ and } z + 1 \leq b_i\}$, and $I_3 = \{i \in [1, n] \mid b_i \leq z\}$. Also, using integer sort in linear time, we rearrange the pairs in (A, B) along with the corresponding sets I_1, I_2, I_3 so that

- (i) for any $i \in I_1, j \in I_2, k \in I_3$, we have $i < j < k$, and
- (ii) the sub-sequences $A[I_1]$ and $B[I_3]$ are sorted in the non-increasing order.

47:12 Efficiently Realizing Interval Sequences

We initialize $D_z = (d_{z,1}, d_{z,2}, \dots, d_{z,n})$ by setting $d_{z,i}$ to a_i if $i \in I_1$, z if $i \in I_2$, and b_i if $i \in I_3$. The sequence D_z is sorted in non-increasing order, since the sub-sequences $A[I_1]$ and $B[I_3]$ are sorted in non-increasing order. Let $\alpha = |I_1|$ and $\beta = |I_1| + |I_2|$, so that $I_2 = [\alpha + 1, \alpha + 2, \dots, \beta]$. We would search all those indices $i \in [\alpha, \beta]$ such that on incrementing $d_{\alpha+1}, \dots, d_i$ to value $z + 1$, the resulting sequence is graphic; or equivalently, the sequence $D_z + E_{[\alpha+1,i]}$ is graphic. Note that for any index $i \in [\alpha, \beta]$,

- (i) the sequence $D_z + E_{[\alpha+1,i]}$ is non-increasing, and
- (ii) $A \leq D_z + E_{[\alpha+1,i]} \leq B$.

The next lemma, which follows from the definition of ϕ , will be used to compute $\phi(D_z + E_{[\alpha+1,i]})$ from $\phi(D_z)$. (The proof is deferred to the full version).

► **Lemma 19.** *For any index $i \in [\alpha + 1, \beta]$, $\phi(D_z + E_{[\alpha+1,i]}) = \phi(D_z) + (i - \alpha)(n - i - \alpha)$.*

For each z we compute the vectors $X(D_z)$ and $Y(D_z)$. For each integer $k \in [1, n]$, let

$$\text{AVOID}(k) = \{i \in [\alpha, \beta] \mid X_k(D_z + E_{[\alpha+1,i]}) > Y_k(D_z + E_{[\alpha+1,i]})\}, \text{ and}$$

$$\text{AVOID} = \bigcup_{k=1}^n \text{AVOID}(k).$$

By Theorem 3, for any $i \in [\alpha, \beta]$, the sequence $D_z + E_{[\alpha+1,i]}$ is graphic if and only if i does not lie in the set AVOID , and $\text{parity}(D_z + E_{[\alpha+1,i]}) = 0$. The following lemma (whose proof is deferred to the full version) shows that the set $\text{AVOID}(k)$, for any index k , is computable in $O(1)$ time.

► **Lemma 20.** *For each $k \in [1, n]$, $\text{AVOID}(k)$ is a contiguous sub-interval of $[1, n]$, and is computable in $O(1)$ time.*

Algorithm 3 presents the procedure for computing the most-regular certificate. For each $k \in [1, n]$, $\text{AVOID}(k)$ is a contiguous sub-interval of $[1, n]$, therefore, the union $\text{AVOID} = \bigcup_{k=1}^n \text{AVOID}(k)$ can be computed in linear time using simple stack based data-structure, once the intervals are sorted in order of their endpoints³ using integer sort. Let \mathcal{I}_z denote the set obtained by removing from $[\alpha, \beta] \setminus \text{AVOID}$ each index i for which $\text{parity}(D_z + E_{[\alpha+1,i]}) = \text{parity}(\text{sum}(D_z) + (i - \alpha))$ is non-zero. Since $\text{sum}(D_z)$ (or $\text{parity}(D_z)$) is computable in $O(n)$, the set \mathcal{I}_z can be computed in $O(n)$ time as well. Note that $D_z + E_{[\alpha+1,i]}$ is graphic if and only if $i \in \mathcal{I}_z$. By Lemma 19, for any index $i \in \mathcal{I}_z$, the value $\phi(D_z + E_{[\alpha+1,i]})$ is computable in $O(1)$ time, once we know $\phi(D_z)$. This shows that in just $O(n)$ time, we can compute the spread of all the levelled sequences D satisfying $z \leq F^{-1}(\text{VOL}(D), \mathcal{S}) < z + 1$, and also find a sequence having the minimum spread. All that remains is to efficiently computing $\phi(D_z)$ for each $z \in [\min(A), \max(B)]$. Observe that $D_{\min(A)} = A$, and so $\phi(D_{\min(A)}) = \sum_{1 \leq r < s \leq n} |a_r - a_s|$ is computable in $O(n^2)$ time. Next by Lemma 19, for any $z \in [\min(A), \max(B) - 1]$, $\phi(D_{z+1}) = \phi(D_z) + (\beta - \alpha)(n - \beta - \alpha)$ is computable in $O(1)$ time. Since z can take $\max(B) - \min(A) - 1$ values, our algorithm in total takes $O(n^2 + n(\max(B) - \min(A) - 1)) = O(n^2)$ time.

This completes the proof of Theorem 2.

³ We say $[r, s] \leq [r', s']$ if either

- (i) $r < r'$, or
- (ii) $r = r'$ and $s \leq s'$.

Algorithm 3 MOST-REGULAR-CERTIFICATE(A, B).

```

1 OPT  $\leftarrow \infty$ ;
2 foreach  $z \in [\min(A), \max(B) - 1]$  do
3    $I_1 \leftarrow \{i \in [1, n] \mid a_i \geq z + 1\}$ ;
4    $I_2 \leftarrow \{i \in [1, n] \mid a_i \leq z, z + 1 \leq b_i\}$ ;
5    $I_3 \leftarrow \{i \in [1, n] \mid b_i \leq z\}$ ;
6   Rearrange the pairs in  $(A, B)$  along with the corresponding sets  $I_1, I_2, I_3$  so that
   (i) for any triplet  $(i, j, k)$  satisfying  $i \in I_1, j \in I_2, k \in I_3$ , we have  $i < j < k$ , and
   (ii) the sub-sequences  $A[I_1]$  and  $B[I_3]$  are sorted in the non-increasing order;
7   Initialize  $D_z = (d_1, d_2, \dots, d_n)$ , where for  $i \in I_1, d_i = a_i$ ; for  $i \in I_2, d_i = z$ ; and
   for  $i \in I_3, d_i = b_i$ ;
8   if  $z = \min(A)$  then set  $\phi(D_z) = \sum_{1 \leq r < s \leq n} |a_r - a_s|$ ;
9   Compute  $X(D_z), Y(D_z)$ ;
10  Let  $\alpha = |I_1|$  and  $\beta = |I_1| + |I_2|$ ;
11  for  $k = 1$  to  $n$  do
12     $\text{AVOID}_1(k) = [\alpha, \beta] \cap [\alpha + 1 + Y_k(D_z) - X_k(D_z), k]$ ;
13    if  $\max\{k - \alpha, 0\} + X_k(D_z) > Y_k(D_z)$  and  $k \leq z$  then
14       $\text{AVOID}_2(k) = [\alpha, \beta] \cap [k, n]$ ;
15    else if  $\max\{k - \alpha, 0\} + X_k(D_z) \leq Y_k(D_z)$  then  $\text{AVOID}_2(k) = \emptyset$ ;
16    else  $\text{AVOID}_2(k) = [\alpha, \beta] \cap [k, \max\{k, \alpha\} + X_k(D_z) - Y_k(D_z) - 1]$ ;
17     $\text{AVOID}(k) = \text{AVOID}_1(k) \cup \text{AVOID}_2(k)$ ;
18  Compute  $\text{AVOID} = \bigcup_{k=1}^n \text{AVOID}(k)$ ;
19  foreach  $i \in [\alpha, \beta] \setminus \text{AVOID}$  do
20    if  $(\text{parity}(D_z) = (i - \alpha) \bmod 2)$  then
21      Compute  $\phi(D_z + E_{[\alpha+1, i]}) = \phi(D_z) + (i - \alpha)(n - i - \alpha)$ ;
22       $\text{OPT} = \min\{\text{OPT}, \phi(D_z + E_{[\alpha+1, i]})\}$ ;
23  Set  $\phi(D_{z+1}) = \phi(D_z) + (\beta - \alpha)(n - \beta - \alpha)$ ;
24 Return OPT and the corresponding graphic sequence;

```

6 Applications and Extensions

In this section, we discuss some related problems whose solutions follow as immediate application of our interval sequence work.

► **Problem 1** (Minimum Graphic extensions). *Given a sequence $A = (a_1, \dots, a_p)$ find the minimum integer $n (\geq p)$ such that a super sequence $D = (a_1, \dots, a_p, d_{p+1}, d_{p+2}, \dots, d_n)$ of sequence A is realizable.*

Solution: Let M denote the value $\max(A) = \max_{i \in [1, p]} a_i$. For any $n \geq p$, let $\mathcal{S}_n = ([a_1, a_1], \dots, [a_p, a_p], [1, n], \dots, [1, n])$ denote the sequence obtained by appending $n - p$ copies of interval $[1, n]$ to interval sequence (A, A) . Let n_0 denote the length of a minimum graphic extension of A . Observe that $n_0 \in [\max\{p, M\}, p + M]$. The lower limit is due to the fact that the length of minimum graphic extension of A must be at least $\max\{p, M\}$; the upper limit holds since one can have a bipartite graph with partitions $X = \{x_1, \dots, x_p\}$ and $Y = \{y_1, \dots, y_M\}$ of length p and M , and for $i \in [1, p]$, connect the vertex x_i to vertices y_1, \dots, y_{a_i} . It turns out that we need to find the smallest integer $n \in [\max\{p, M\}, p + M]$

such that \mathcal{S}_n is graphic. The minimum n can be obtained by a binary search over the range $[\max\{p, M\}, p + d]$ and using Theorem 4; this takes $O(\max\{p, M\} \log \max\{p, M\})$ time. Once n_0 is known, the optimal graphic extension can be computed using Theorem 14 for searching graphic certificate in $O(\max\{p, M\} \log \max\{p, M\})$ time.

► **Problem 2.** Given $A = (a_1, \dots, a_n)$, find a graphic sequence $D = (d_1, \dots, d_n)$ whose chebyshev distance (L_∞ distance) from A is minimum.

Solution: The above problem can be reduced to interval sequence problem, as we need to find smallest non-negative integer $c \in [1, n]$ such that $\mathcal{S}_c = ([a_1 - c, a_1 + c], \dots, [a_n - c, a_n + c])$ is realizable. To find the minimum c , we do a binary search with help of Theorem 4 for verification; this takes $O(n \log n)$ time. Once optimal c is known, the sequence D can be computed using Theorem 14 to search graphic certificate in \mathcal{S}_c , thus the time complexity for computing sequence D is $O(n \log n)$.

► **Problem 3.** Given $A = (a_1, \dots, a_n)$, find minimum fraction ϵ and a graphic sequence $D = (d_1, \dots, d_n)$ satisfying $a_i(1 - \epsilon) \leq d_i \leq a_i(1 + \epsilon)$.

Solution: Again we need to find smallest non-negative fraction ϵ such that the interval sequence $\mathcal{S}_\epsilon = ([a_1(1 - \epsilon), a_1(1 + \epsilon)], \dots, [a_n(1 - \epsilon), a_n(1 + \epsilon)])$ is realizable. To find the minimum ϵ , we do a binary search with help of Theorem 4; this takes $O(n \log n)$ time. Once ϵ is known, using Theorem 14, sequence D can be computed in $O(n \log n)$ time.

References

- 1 Martin Aigner and Eberhard Triesch. Realizability and uniqueness in graphs. *Discrete Mathematics*, 136:3–20, 1994.
- 2 Richard Anstee. Properties of a class of (0,1)-matrices covering a given matrix. *Can. J. Math.*, pages 438–453, 1982.
- 3 Richard Anstee. An algorithmic proof of Tutte’s f-factor theorem. *J. Algorithms*, 6(1):112–131, 1985.
- 4 M. Behzad and James E. Simpson. Eccentric sequences and eccentric sets in graphs. *Discrete Mathematics*, 16(3):187–193, 1976.
- 5 Mao-cheng Cai, Xiaotie Deng, and Wenan Zang. Solution to a problem on degree sequences of graphs. *Discrete Mathematics*, 219(1-3):253–257, 2000.
- 6 Wai-Kai Chen. On the realization of a (p,s)-digraph with prescribed degrees. *Journal of the Franklin Institute*, 281(5):406–422, 1966.
- 7 Paul Erdős and Tibor Gallai. Graphs with Prescribed Degrees of Vertices [Hungarian]. *Matematikai Lapok*, 11:264–274, 1960.
- 8 A. Frank. Augmenting graphs to meet edge-connectivity requirements. *SIAM J. Discrete Math.*, 5:25–43, 1992.
- 9 A. Frank. Connectivity augmentation problems in network design. In *Mathematical Programming: State of the Art*, pages 34–63. Univ. Michigan, 1994.
- 10 H. Frank and W. Chou. Connectivity considerations in the design of survivable networks. *IEEE Trans. Circuit Theory*, CT-17:486–490, 1970.
- 11 Satoru Fujishige and Sachin B. Patkar. Realization of set functions as cut functions of graphs and hypergraphs. *Discrete Mathematics*, 226(1-3):199–210, 2001.
- 12 D.R. Fulkerson. Zero-one matrices with zero trace. *Pacific J. Math.*, 12:831–836, 1960.
- 13 Ankit Garg, Arpit Goel, and Amitabha Tripathi. Constructive extensions of two results on graphic sequences. *Discrete Applied Mathematics*, 159(17):2170–2174, 2011.
- 14 R.E. Gomory and T.C. Hu. Multi-terminal network flows. *J. Soc. Industrial & Applied Math.*, 9, 1961.

- 15 Jiyun Guo and Jianhua Yin. A variant of Niessen's problem on degree sequences of graphs. *Discrete Mathematics and Theoretical Computer Science*, Vol. 16 no. 1 (in progress)(1):287–292, May 2014. Graph Theory. URL: <https://hal.inria.fr/hal-01179211>.
- 16 S. Louis 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.
- 17 Sepp Hartung and André Nichterlein. NP-Hardness and Fixed-Parameter Tractability of Realizing Degree Sequences with Directed Acyclic Graphs. *SIAM J. Discrete Math.*, 29(4):1931–1960, 2015.
- 18 V. Havel. A Remark on the Existence of Finite Graphs [in Czech]. *Casopis Pest. Mat.*, 80:477–480, 1955.
- 19 Katherine Heinrich, Pavol Hell, David G. Kirkpatrick, and Guizhen Liu. A simple existence criterion for $g < f$ -factors, with applications to $[a, b]$ -factors. *Discrete Mathematics*, 85:313–317, 1990.
- 20 Pavol Hell and David G. Kirkpatrick. Linear-time certifying algorithms for near-graphical sequences. *Discrete Mathematics*, 309(18):5703–5713, 2009.
- 21 Daniel J. Kleitman and D. L. Wang. Algorithms for constructing graphs and digraphs with given valences and factors. *Discrete Mathematics*, 6(1):79–88, 1973.
- 22 Linda Lesniak. Eccentric sequences in graphs. *Periodica Mathematica Hungarica*, 6(4):287–293, 1975.
- 23 László Lovász. Subgraphs with prescribed valencies. *J. Comb. Theory*, 8:391–416, 1970.
- 24 N.V.R. Mahadev and U.N. Peled. *Threshold Graphs and Related Topics*. Annals of Discrete Mathematics. Elsevier Science, 1995.
- 25 Akutsu Tatsuya and Hiroshi Nagamochi. Comparison and enumeration of chemical graphs. *Computational and structural biotechnology*, 5, 2013.
- 26 Amitabha Tripathi, Sushmita Venugopalan, and Douglas B. West. A short constructive proof of the Erdos-Gallai characterization of graphic lists. *Discrete Mathematics*, 310(4):843–844, 2010.
- 27 W. T. Tutte. Graph factors. *Combinatorica*, 1(1):79–97, March 1981.
- 28 D.L. Wang and D.J. Kleitman. On the existence of n -connected graphs with prescribed degrees ($n > 2$). *Networks*, 3:225–239, 1973.

Efficient Interactive Proofs for Linear Algebra

Graham Cormode 

University of Warwick, UK
g.cormode@warwick.ac.uk

Chris Hickey

University of Warwick, UK
c.hickey@warwick.ac.uk

Abstract

Motivated by the growth in outsourced data analysis, we describe methods for verifying basic linear algebra operations performed by a cloud service without having to recalculate the entire result. We provide novel protocols in the streaming setting for inner product, matrix multiplication and vector-matrix-vector multiplication where the number of rounds of interaction can be adjusted to tradeoff space, communication, and duration of the protocol. Previous work suggests that the costs of these interactive protocols are optimized by choosing $O(\log n)$ rounds. However, we argue that we can reduce the number of rounds without incurring a significant time penalty by considering the total end-to-end time, so fewer rounds and larger messages are preferable. We confirm this claim with an experimental study that shows that a constant number of rounds gives the fastest protocol.

2012 ACM Subject Classification Theory of computation → Interactive computation

Keywords and phrases Streaming Interactive Proofs, Linear Algebra

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.48

Funding *Graham Cormode*: Supported by European Research Council grant ERC-2014-CoG 647557.

Chris Hickey: Supported by European Research Council grant ERC-2014-CoG 647557.

1 Introduction

The pitch for cloud computing services is that they allow us to outsource the effort to store and compute over our data. The ability to gain cheap access to both powerful computing and storage resources makes this a compelling offer. However, it brings increased emphasis on questions of trust and reliability: to what extent can we rely on the results of computations performed by the cloud? In particular, the cloud provider has an economic incentive to take shortcuts or allow buggy code to provide fast results, if they are hardly noticed by the client.

Prior work has developed the idea of using interactive proofs to independently verify outsourced computations without duplicating the effort. Originally invented as tools in the realm of computational complexity, recent work has sought to argue that interactive proofs can indeed be practically used for verification. Modern research takes two main approaches, from highly general methods with currently far-from-practical costs, to tackling specific fundamental problems where the overhead of verification is negligible.

In this work, we focus on the “negligible” end of the spectrum and study primitive computations within linear algebra – a core set of tools with applications across engineering, data analysis and machine learning. We make four main contributions:

- We consider protocols for inner product and matrix multiplication and present lightweight tunable verification protocols for these problems. We also produce an entirely new protocol for vector-matrix-vector multiplication.
- Our protocols allow us to trade off computational effort and communication size against the number of rounds of interaction. We show it is often desirable to have fewer rounds of interaction.



© Graham Cormode and Chris Hickey;

licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 48; pp. 48:1–48:19

Leibniz International Proceedings in Informatics



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

- We optimize the costs for the cloud, and show that the protocols impose a computational overhead that is typically much smaller than the cost of the computation itself.
- Our experimental study confirms our analysis, and demonstrates that the absolute cost is minimal, with the client's cost significantly less than performing the computation independently.

1.1 Streaming Interactive Proofs

Our work adopts the model of *streaming interactive proofs* (SIPs), formalized in [7, 8].

► **Definition 1.** *We have two communicating computational entities, a helper, H , and a verifier, V , observing a stream \mathcal{S} . V wishes to know $f(\mathcal{S})$, for some function f . After viewing the stream, H and V have a conversation, culminating in V producing an output, $\text{Out}(V, \mathcal{S}, V_R, H)$, where V_R represents a private random string belonging to V , so that*

$$\text{Out}(V, \mathcal{S}, V_R, H) = \begin{cases} X & \text{if } V \text{ is convinced by } H \text{ that } f(\mathcal{S}) = X \\ \perp & \text{Otherwise} \end{cases}$$

We say the protocol used by the two parties is **complete** for f if there exists an honest helper H such that

$$\mathbb{P}[\text{Out}(V, \mathcal{S}, V_R, H) = f(\mathcal{S})] = 1$$

and **sound** if for any helper, H' , and any input, \mathcal{S}'

$$\mathbb{P}[\text{Out}(V, \mathcal{S}', V_R, H') \notin \{f(\mathcal{S}'), \perp\}] \leq \frac{1}{3}$$

Informally, complete protocols always accept an honest answer, and sound protocols reject an incorrect answer most of the time (the constant probability $\frac{1}{3}$ is arbitrary and can be reduced to be vanishingly small via standard amplification techniques). If a protocol for V is both complete and sound, we call it a *valid protocol* for f . A valid protocol is characterized by costs in terms of required space and communication.

► **Definition 2.** *For a function f we say that there is a d -round (h, v) -protocol if there is a valid protocol for f with*

- **Verifier Memory v** – Verifier uses $O(v)$ working memory.
- **Communication h** – The total communication between the two parties is $O(h)$. Note that we do not include the cost of sending the claimed solution in this cost.
- **Interactivity d** at most $2d$ messages sent from H to V or vice versa.

Furthermore, we quantify the computational costs by

- **Verifier Streaming Cost** – The work during the initial stream.
- **Verifier Checking Computation** – The work for the interactive stage.
- **Helper Overhead** – The additional work outside of solving the problem.

Problem Statement

We seek optimal or near optimal verification protocols for core linear algebra operations. The canonical (and previously studied) example is the multiplication of two matrices $A \in \mathbb{F}_q^{k \times n}$, $B \in \mathbb{F}_q^{n \times k'}$, where \mathbb{F}_q is the finite field of integers modulo q , for some prime $q > M^2 n$, where $M = \max_{i,j} (A_{ij}, B_{ij})$ or chosen sufficiently large to not incur overflows. Our protocols work on any prime size finite field, consistent with prior work. This allows computation over fixed precision rational numbers, with appropriate scaling. For ease of exposition, we assume

in this paper that $n = k = k'$, although all our algorithms work with rectangular matrices. The resulting matrix AB is assumed to be too large for the verifier to conveniently store, and so our aim is for the helper to allow the verifier to compute a *fingerprint* of AB [14], defined formally in Section 3.1, that can be used to check the helper's claimed answer.

1.2 Prior Work

Interactive proofs were introduced in the 1980s, primarily as a tool for reasoning about computational complexity [12]. A key result showed that the class of problems admitting interactive proofs is equivalent to the complexity class PSPACE [17]. Subsequent work in this direction led to the development of probabilistically checkable proofs (PCPs), where (in our terminology) the verifier only inspects a small fraction of the proof written by the helper. One distinction between this prior work and our setting is that PCPs consider a verifier who can devote polynomial time to inspecting the proof and has access to the full input; by contrast, we consider weaker verifiers, and try to more tightly bound their space and computational resources. The notion that interactive proofs could be a practical tool for verifying outsourced computation was advocated by Goldwasser, Kalai and Rothblum [11]. This paper introduced the powerful GKR (or “muggles”) protocol for verifying arbitrary computations specified as arithmetic circuits. Several papers have aimed to optimize the costs of the GKR protocol [7, 19, 18], or to provide systems for verifying general purpose computation under a variety of computational or cryptographic models [13, 16, 15]. The latter of which tackle large classes of problems using *arguments*, which consider a computationally bounded prover. We consider only proofs as we can achieve highly efficient protocols without requiring restriction on the prover, or use of cryptographic assumptions. Furthermore, some costs associated with such verification still remain high, such as requiring a large amount of pre-processing on the part of the helper, which can only be amortized over a large number of invocations. For the common and highly symmetric algebraic computations we work with in this paper, it is beneficial to build a specialised protocol.

Other work has considered engineering protocols for specific problems that are more lightweight, and so trade generality for greater practicality. The motivation is that some primitives are sufficiently ubiquitous that having special purpose protocols will outweigh the effort to design them. An early example of this is given by Frievalds' algorithm for verifying matrix multiplication [10]. This and similar algorithms unfortunately don't directly work for verifiers that can't store the entire input. This line of work was initiated for problems arising in the context of data stream processing, such as frequency analysis of vectors derived from streams [5]. Follow-up work addressed problems on graph data [8], data mining [9] and machine learning [6].

These papers tend to consider either the non-interactive case (minimizing the number of rounds), or have a poly-logarithmic number of rounds (minimizing the total communication). For example, [8] introduces an interactive inner product protocol which can accommodate a variable number of rounds. The development assumes that setting the number of rounds to be $\log(n)$ will be universally optimal, an assumption we reassess in this work. Similarly, in [18] the matrix multiplication protocol takes place over $O(\log(n))$ rounds. Our observation is that the pragmatic choice may fall between these extremes of non-interactive and highly interactive. Taking into account latency and round-trip time between participants, the preferred setting might be a constant number of rounds, which yields a communication cost which is a small polynomial in the input size, but which is not significantly higher in absolute terms from the minimal poly-logarithmic cost.

We summarize the current state of the art for the problems of computing inner product (Table 1) and matrix multiplication (Table 2), and show the results we obtain here for comparison.

■ **Table 1** Different SIPs for Inner Product with $u, v \in \mathbb{F}_q^n$, with $n = l^d$ and $a \in [0, 1]$.

Method	$O(h)$	$O(v)$	Rounds	H overhead	V overhead + checking
This Work	$O(ld)$	$O(l + d)$	$d - 1$	$O(n \log(l))$	$O(nld) + O(ld)$
Binary SC [8]	$O(\log(n))$	$O(\log(n))$	$\log(n)$	$O(n)$	$O(n \log(n)) + O(\log(n))$
FFT LDEs [7]	$O(n^{1-a})$	$O(n^a)$	1	$O(n \log(n))$	$O(n) + O(\log(n))$

■ **Table 2** Different SIPs for Matrix Multiplication with $A, B \in \mathbb{F}_q^{n \times n}$ and $n = l^d$.

Method	$O(h)$	$O(v)$	Rounds	H overhead	V overhead + checking
This Work	$O(ld)$	$O(l + d)$	d	$O(n^2)$	$O(n^2ld) + O(ld)$
Binary SC [18]	$O(\log(n))$	$O(\log(n))$	$\log(n) + 1$	$O(n^2)$	$O(n^2 \log(n)) + O(\log(n))$
Fingerprints [5]	$O(n^2)$	$O(1)$	1	$O(1)$	$O(n^2) + O(n^2)$

Lastly, we comment that our results are restricted to the information-theoretically secure model of Interactive Proofs, and are separate from recent results in the computational (cryptographic) security model [3, 4].

1.3 Contributions and outline

Our main contribution is an investigation into the time-optimal number of rounds for a variety of protocols. We adapt and improve protocols for inner product and matrix multiplication, as well as introducing an entirely new protocol for vector-matrix-vector multiplication. We then perform experiments in order to evaluate the time component of each stage of interaction.

We begin in Section 2 by re-evaluating how to measure the communication cost of a protocol, and propose to combine the competing factors of latency and bandwidth into a total time cost. This motivates generalized protocols that take a variable number of rounds, where we can pick a parameter setting to minimize the total completion time.

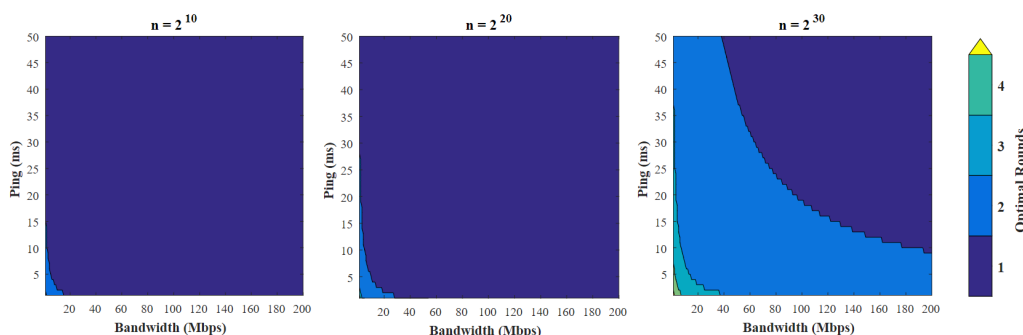
In Section 3 and 4 we build on previous protocols [8, 7] to construct novel efficient *variable round* protocols for core linear algebra operations. We begin by revisiting variable round protocols for inner product. We leverage these to obtain new protocols for matrix multiplication and vector-matrix-vector multiplication (which does not appear to have been studied previously) with similar asymptotic costs.

In Section 5, we thoroughly analyse the practical computation costs of the resulting protocols, and compare to existing verification methods. We perform a series of experiments to back up our claims, and draw conclusions on what we should want from interactive proofs. We show that it can be preferable to use fewer rounds, despite some apparently higher costs.

2 How Much Interaction Do We Want?

Prior work has sought to find “optimal” protocols which minimize the total communication cost. This is achieved by *increasing* the number of rounds of interaction, with the effect of driving down the amount of communication in each round. The minimum communication is typically attained when the number of rounds is polylogarithmic [7]. The non-interactive case represents another extreme in this regard, requiring a single message from the helper to verifier. This allows the parties to work asynchronously at the cost of larger total communication.

In this section we argue that the right approach is neither the non-interactive case *nor* the highly-interactive case. Rather, we argue that a compromise of “moderately interactive proofs” can yield better results. To do so we consider the overall time required to process the proof.



■ **Figure 1** Optimal number of rounds for matrix multiplication of various sizes when considering only communication, with a field size $q = O(n^3)$.

The key observation is that the time to process a proof depends not just on the amount of communication, but also the number of rounds. In the protocols from Table 1 and 2, each round cannot commence until the previous round completes, hence we incur a time penalty as a function of the *latency* between the two communicating parties. The duration of a round depends on the bandwidth between them. Thus, we aim to combine number of rounds and message size into a single intuitive quantity based on bandwidth and latency that captures the total wall-clock time cost of the protocol.

For matrix multiplication, the variable round protocols summarized in Table 2 spread the verification over d rounds, and have a total communication cost proportional to $dn^{1/d}$. Hence, we write the time to perform the communication of the protocol as $T = 2d\mathcal{L} + \frac{2dn^{1/d}\log(|\mathbb{F}|)}{\mathcal{B}}$, where latency (\mathcal{L}) is measured in seconds, and bandwidth (\mathcal{B}) in bits per second. This expression emerges due to the $2d$ changes in direction over the protocol, and considering a protocol that sends a total of $2dn^{1/d}$ field elements (from the analysis in Section 4.2).

We measured the cost using typical values of \mathcal{L} and \mathcal{B} observed on a university campus network, where the “ping” time to common cloud service providers (Google, Amazon, Microsoft) is of the order of 20ms, and the bandwidth is around 100Mbps. From the above equation for T we see that, for a constant field size $|\mathbb{F}|$, the value of $2n^{1/d}d\log(|\mathbb{F}|)/\mathcal{B}$ is dominated by $2d\mathcal{L}$ for even small d under such parameter settings. Hence, we should prefer fewer rounds as latency increases. Figure 1 shows the number of rounds which minimizes the communication time as a function of the size of the input. We observe that the answer is a small constant, at most just two or three rounds, even for the largest input sizes, corresponding to exabytes of data.

3 Primitives

Before we introduce our protocols, we first describe the building blocks they rely on.

3.1 Fingerprints

Fingerprints can be thought of as hash functions for large vectors and matrices with additional useful algebraic properties. For $A \in \mathbb{F}_q^{n \times n}$ and $x \in \mathbb{F}_q$, define the matrix fingerprint as $F_x(A) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} A_{ij}x^{in+j}$. Similarly, for $u \in \mathbb{F}_q^n$ we have the vector fingerprint $F_x^{\text{vec}}(u) = \sum_{i=0}^{n-1} u_i x^i$. The probability of two different matrices having the same fingerprint (over the random choice of x) can be made arbitrarily small by increasing the field size.

► **Lemma 3** ([14]). *Given $A, B \in \mathbb{F}_q^{n \times n}$ and $x \in_R \mathbb{F}_q$, we have $\mathbb{P}[F_x(A) = F_x(B) | A \neq B] \leq \frac{n^2}{q}$.*

A similar result holds for F_x^{vec} . In our model, fingerprints can be constructed in constant space, and with computation linear to the input size.

3.2 Low Degree Extensions

Low degree extensions (LDEs) have been used extensively in interactive proofs. LDEs have been used in conjunction with sum-check (Section 3.3) in a variety of contexts [11, 7, 8]. Formally, for a set of data S an LDE is a low degree polynomial that goes through each data point. Typically, we think of S as being laid out as a vector or d -dimensional tensor indexed over integer coordinates. This polynomial can then be evaluated at a random point r with the property that, like fingerprinting, two different data sets are unlikely to evaluate to the same value at r (inversely proportional to the field size).

Given input as a vector $u \in \mathbb{F}_q^n$, we consider two new parameters, l and d with $n \leq l^d$, and re-index u over $[l]^d$. The d -dimensional LDE of u satisfies $\tilde{f}_u(k_0, \dots, k_{d-1}) = u_k$ for $k \in [n]$ where $k_0 \dots k_{d-1}$ is the base l representation of k . For a random point $r = (r_0, \dots, r_{d-1}) \in \mathbb{F}_q^d$, we have

$$\tilde{f}_u(r_0, \dots, r_{d-1}) = \sum_{k_0}^{l-1} \cdots \sum_{k_{d-1}}^{l-1} u_k \chi_k(r) \quad (1)$$

$$\chi_k(r) = \prod_{j=0}^{d-1} \prod_{\substack{i=0 \\ i \neq k_j}}^{l-1} \frac{r_j - i}{k_j - i}, \quad (2)$$

where χ is the Lagrange basis polynomial. Note that $\tilde{f}_u : \mathbb{F}_q^d \rightarrow \mathbb{F}_q$ and $q \geq l$. A similar definition can be used for a matrix $A \in \mathbb{F}_q^{n \times n}$, by reshaping into a vector in $\mathbb{F}_q^{n^2}$.

The polynomials can be evaluated over a stream of updates in space $O(d)$ and time per update $O(ld)$ [8]. The time cost of our verifier to evaluate an LDE at one location, r , is $O(nld)$ (for sparse data, n can be replaced with the number of non-zeros in the input).

3.3 Sum-Check Protocol

Our final primitive is the sum-check protocol [12]. Sum-check is a multi-round protocol for verifying the sum

$$G = \sum_{k_0=0}^{l-1} \sum_{k_1=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} g(k_0, k_1, \dots, k_{d-1}) \text{ for } g : \mathbb{F}_q^d \rightarrow \mathbb{F}_q. \quad (3)$$

For our purposes, g will be a polynomial derived from the LDE of a dataset of size $n = l^d$ (i.e. the d -dimensional tensor representation of the data), and each polynomial used in the protocol will have degree λ , with $\lambda = O(l)$; however, we keep the parameter λ for completeness. Provided that all the checks are passed then the verifier is convinced that (except with small probability) the value G was as claimed in (3). The original descriptions of the sum-check protocol [12, 2] use $l = 2$, however we shift to using arbitrary l , similar to [1, 7, 8]. The protocol goes as follows:

Stream Processing: V randomly picks $r \in \mathbb{F}_q^d$ and computes $g(r_0, \dots, r_{d-1})$.

Round 1: H computes and sends G and $g_0 : \mathbb{F}_q \rightarrow \mathbb{F}_q$, where

$$g_0(k_0) = \sum_{k_1=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} g(k_0, k_1, \dots, k_{d-1}).$$

V checks that $G = \sum_{k_0=0}^{l-1} g_0(k_0)$, computes $g_0(r_0)$ and sends r_0 to H .

⋮

Round $j + 1$: H has received r_0, \dots, r_{j-1} from V , and sends $g_j : \mathbb{F}_q \rightarrow \mathbb{F}_q$, where

$$g_j(k_j) = \sum_{k_{j+1}=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} g(r_0, \dots, r_{j-1}, k_j, \dots, k_{d-1}).$$

V checks if $g_{j-1}(r_{j-1}) = \sum_{k_j=0}^{l-1} g_j(k_j)$, computes $g_j(r_j)$ and sends r_j to H .

⋮

Round d : H sends $g_{d-1} : \mathbb{F}_q \rightarrow \mathbb{F}_q$, where $g_{d-1}(k_{d-1}) = g(r_0, \dots, r_{d-3}, r_{d-2}, k_{d-1})$.

V checks that $g_{d-2}(r_{d-2}) = \sum_{k_{d-1}=0}^{l-1} g_{d-1}(k_{d-1})$, computes $g_{d-1}(r_{d-1})$, and finally checks this is $g(r_0, \dots, r_{d-2}, r_{d-1})$.

H can express the polynomial g_j as a set $G_j = \{(g_j(x), x) : x \in [\lambda]\}$. In each round V sums the first l elements of this set, and checks it is $g_{j-1}(r_{j-1})$ for $j > 0$, then evaluates the LDE of G_j at r_j , giving a computation cost per round of $O(l + \lambda)$. The verifier also has to do some work in the streaming phase, evaluating the function g at r , with cost $O(n\lambda d)$. The helper's computation time comes from having to evaluate g at l^{d-j} points in the j th round, and so ultimately evaluating g at $\sum_{j=1}^{d-1} l^{d-j} = O(n)$ points, with a cost per point of $O(\lambda d)$ (we subsequently show how this can be reduced in our protocols for linear algebra). The costs of performing sum-check are summarized as follows:

Communication $O(\lambda d)$ words, spread over d rounds.

Helper costs $O(n\lambda d)$ time for computation.

Verifier costs $O(\lambda + d)$ memory cost, $O(n\lambda d)$ overhead to compute LDE and checking cost $O(d(l + \lambda))$.

In our implementations, we will optimize our methods to “stop short” the sum-check protocol and terminate at round $d - 1$ (this idea is implicit in the work of Aaronson and Wigderson [1, Section 7.2]). In this setting, the verifier finds the set

$$\{g(r_0, \dots, r_{d-3}, r_{d-2}, k_{d-1}) : k_{d-1} \in [l]\}.$$

in the stream processing stage, and then checks this against the claimed set of values provided by the helper in round $d - 1$. This increases the space used by the verifier to maintain these l LDE evaluations. However, this does not affect the asymptotic space usage of the verifier, since we assume that V already keeps space proportional to l to handle H 's messages. It does not affect the streaming overhead time, since each update affects only the LDE point with which it shares the final coordinate. Equivalently, this can be viewed as running l instances of sum-check in parallel on the data divided into l partitions. Hence, this appears as an all-round improvement, at least in theory.

4 Protocols for Linear Algebra Primitives

Using the previously discussed primitives for SIPs, we show how they have been used in inner product [7]. We then use this to construct a new variable round method for matrix multiplication, and extend it to achieve a novel vector-matrix-vector multiplication protocol.

4.1 Inner Product

Given two vectors $a, b \in \mathbb{F}_q^n$, the verifier wishes to receive $a^T b \in \mathbb{F}_q$ from the helper. We give a straightforward generalization of the analysis of a protocol in [8], as an application of sum-check on the LDEs of a and b . This variable round protocol has costs detailed below.

► **Theorem 4.** *Given $a, b \in \mathbb{F}_q^n$, there is a $(d-1)$ -round $(ld, l+d)$ -protocol with $n = l^d$ for verifying $a^T b$ with helper computation time $O\left(\frac{n \log(n)}{d}\right)$, verifier overhead $O(nld)$, and checking cost $O(ld)$.*

The analysis from [8] sets $l = 2$ and $d = \log(n)$, and the computational cost for the verifier is $O(\log(n))$ while the cost for the helper is $O(n \log(n))$. For general l and d these costs become $O(ld)$ and $O(nld)$ for the verifier and helper respectively.

In [7] it is shown how the helper's cost can be reduced to $O(n \log(n))$ for $d = 2$ and $l = \sqrt{n}$ using the Discrete Fast Fourier Transform to make a fast non-interactive protocol. We extend this for arbitrary d and l , and show how by combining with sum-check we can keep the helper's computation low, proving Theorem 1.

► **Lemma 5.** *Given $a, b \in \mathbb{F}_q^n$ the sum*

$$a^T b = \sum_{k_0=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} \tilde{f}_a(k_0, \dots, k_{d-1}) \tilde{f}_b(k_0, \dots, k_{d-1}) \quad (4)$$

can be verified using a $(d-1)$ -round $(ld, l+d)$ -protocol with helper computation time $O\left(\frac{n \log(n)}{d}\right)$, and verifier computation time $O(ld)$, overhead time $O(nld)$.

Proof. First, set

$$g(k_0, \dots, k_{d-1}) = \tilde{f}_a(k_0, \dots, k_{d-1}) \tilde{f}_b(k_0, \dots, k_{d-1}).$$

$g : \mathbb{F}_q^d \rightarrow \mathbb{F}_q$ is a degree $2l$ polynomial in each variable. Now, consider round $j+1$ of the sum-check protocol, where the helper is required to send

$$g_j(x) = \sum_{k_{j+1}=0}^{l-1} \cdots \sum_{k_d=0}^{l-1} g(r_1, \dots, r_{j-1}, x, k_{j+1}, \dots, k_d).$$

Here, g is degree $2l$ polynomial, sent to V as a set $G_j^\Sigma = \{(g_j(x), x) : x \in [2l]\}$. To compute this set we have H find the individual summands as

$$G_j = \left\{ (g(r_1, \dots, r_{j-1}, x, k_{j+1}, \dots, k_{d-1}), x) : x \in [2l], k_{j+1}, \dots, k_{d-1} \in [l] \right\}.$$

Naive computation of all the values in G_j takes time $O(nd)$ each, for a total cost of $O(nl^{d-j}d)$. However, instead of computing the LDE at l^{d-j} points with cost $O(ld)$ we can sum l^{d-j} convolutions of length $2l$ vectors to obtain the same result. We present the full proof of this claim in the Appendix. The total cost of each convolution is $O(l \log(l))$. Summing these l^{d-j} convolutions gives the cost of the j th round for the helper as $O\left(\frac{l^{d-j} \log(n)}{d}\right)$. Summing $\sum_{j=0}^{d-1} l^{d-j}$ over the d rounds gives us our cost of $O\left(\frac{n \log(n)}{d}\right)$. The remaining costs are as in our version of the sum-check protocol (Section 3.3). ◀

4.2 Matrix Multiplication

By combining the power of LDEs with the matrix multiplication methods from [6], we can create a protocol with only marginally larger costs than inner product.

► **Theorem 6.** *Given two matrices $A, B \in \mathbb{F}_q^{n \times n}$, we can verify the product $AB \in \mathbb{F}_q^{n \times n}$ using a d -round $(ld, l + d)$ -protocol with verifier overhead time $O(n^2ld)$, checking time $O(ld)$ and helper computation time $O(n^2)$.*

Proof. We make use of the matrix fingerprints from [6], and generate the fingerprint of AB for some $x \in \mathbb{F}_q$ by expressing matrix multiplication as a sum of outer products.

$$F_x(AB) = \sum_{i=0}^{n-1} F_{x^n}^{\text{vec}}(A_i^\downarrow) F_x^{\text{vec}}(B_i^\rightarrow) \quad (5)$$

where A_i^\downarrow denotes the i th column of A and B_j^\rightarrow is the j th row of B . We also define:

$$A_{\text{col}} = (F_{x^n}^{\text{vec}}(A_1^\downarrow), \dots, F_{x^n}^{\text{vec}}(A_n^\downarrow)) \text{ and } B_{\text{row}} = (F_x^{\text{vec}}(B_1^\rightarrow), \dots, F_x^{\text{vec}}(B_n^\rightarrow)).$$

Our fingerprint $F_x(AB)$ is then given by the inner product of A_{col} and B_{row} . We apply the inner product protocol of Theorem 4, hence we need to show the verifier can evaluate the LDE of the product of these two vectors at a random point,

$$\sum_{k_{d-1}=0}^{l-1} \tilde{f}_{A_{\text{col}}}(r_0, \dots, r_{d-2}, k_{d-1}) \tilde{f}_{B_{\text{row}}}(r_0, \dots, r_{d-2}, k_{d-1}),$$

which we denote as $\Sigma \tilde{f}_{A_{\text{col}}}(r) \tilde{f}_{B_{\text{row}}}(r)$. We can construct this value in the initial stream by storing, for each value of k_{d-1} , $\tilde{f}_{A_{\text{col}}}(r_0, \dots, r_{d-1}, k_{d-1})$ and $\tilde{f}_{B_{\text{row}}}(r_0, \dots, r_{d-1}, k_{d-1})$, which is done in space $O(ld)$ for the verifier. Each of these requires an initial verifier overhead of $O(ld)$ for each of the n^2 elements, then checking requires $O(ld)$ as in Theorem 4. The helper has to fingerprint the matrices to form A_{col} and B_{row} , at a cost of $O(n^2)$. The result follows by using the generated fingerprint to compare to the fingerprint of the claimed result AB (which is provided by the helper in some suitable form, and excluded from the calculation of the protocol costs). ◀

Note that the helper is not required to follow any particular algorithm to compute the matrix product AB . Rather, the purpose of the protocol is for the helper to assist the verifier in computing a fingerprint of AB from its component matrices. The time cost of this is much faster: linear in the size of the input.

Fingerprinting versus LDEs. Our protocol in Theorem 6 is stated in terms of fingerprints. In [18], a d -round protocol is presented which uses

$$\tilde{f}_{AB}(R_1, R_2) = \sum_{k_0=0}^1 \cdots \sum_{k_{\log(n)-1}=0}^1 \tilde{f}_A(R_1, k) \tilde{f}_B(k, R_2).$$

This uses the inner product definition of matrix multiplication, whilst we use the outer product property of fingerprints. Finding $\tilde{f}_{AB}(R_1, R_2)$ during the initial streaming has cost per update $O(\log(n))$. For our method, we find $\Sigma \tilde{f}_{A_{\text{col}}}(r) \tilde{f}_{B_{\text{row}}}(r)$, which has cost $O(ld)$. In the case $l = 2, d = \log(n)$, we see these two methods are very similar. The methods differ in how we respond to receiving the result, AB . In [18], the verifier computes the LDE of

AB at a cost of $O(n^2ld)$, while our method takes time $\tilde{O}(n^2)$ to process the claimed AB , as we simply fingerprint the result. Thaler's method possesses some other advantages, for example it can chain matrix powers (finding A^m) without the Helper having to materialize the intermediate matrices. Nevertheless, in data analysis applications, it is often the case that only a single multiplication is required.

4.3 Vector-Matrix-Vector Multiplication

Vector-matrix-vector multiplication appears in a number of scenarios. A simple example arises in the context of graph algorithms: suppose that helper wishes to demonstrate that a graph, specified by an adjacency matrix A , is bipartite. Let v be an indicator vector for one part of the graph, then $v^T Av = (\mathbf{1} - v)^T A(\mathbf{1} - v) = 0$ iff v is as claimed. More generally, the helper can show a k colouring of a graph using k vector-matrix-vector multiplications between the adjacency matrix and the k disjoint indicator vectors for the claimed colour classes.

We reduce the problem of vector-matrix-vector multiplication (which yields a single scalar) to inner product computation, after reshaping the data as vectors. Formally, given $u, v \in \mathbb{F}^n$ and $A \in \mathbb{F}^{n \times n}$, we can compute $u^T Av$ as

$$u^T Av = \sum_{i=1}^n \sum_{j=1}^n u_i A_{ij} v_j = (uv^T)_{\text{vec}} \cdot A_{\text{vec}}$$

$u^T Av$ is equal to computing the inner product of A and uv^T written as length n^2 vectors. Protocols using this form will need to make use of an LDE evaluation of uv^T . We show that this can be built from independent LDE evaluations of each vector.

► **Lemma 7.** *Given $u, v \in \mathbb{F}^n$ and $r \in_R \mathbb{F}^d$, with $n = l^d$*

$$f_{uv^T}(r_0, \dots, r_{2d-1}) = f_u(r_0, \dots, r_{d-1}) f_v(r_d, \dots, r_{2d-1})$$

Proof. We abuse notation a little to treat uv^T as a vector of length n^2 , and we assume that $n = l^d$ (if not, we can pad the vectors with zeros without affecting the asymptotic behaviour). We write $R_1 = (r_0, \dots, r_{d-1})$ and $R_2 = (r_d, \dots, r_{2d-1})$. The proof follows by expanding out expression (2) to observe that $\chi_k(r_0 \dots r_{2d-1}) = \chi_{k_0, \dots, k_{d-1}}(R_1) \chi_{k_d, \dots, k_{2d-1}}(R_2)$ and so

$$\begin{aligned} f_{uv^T}(r_0, \dots, r_{2d-1}) &= \sum_{k_0=0}^{l-1} \dots \sum_{k_{2d-1}=0}^{l-1} [(uv^T)_k \chi_k(r)] \\ &= \sum_{i_0=0}^{l-1} \dots \sum_{i_{d-1}=0}^{l-1} \sum_{j_0=0}^{l-1} \dots \sum_{j_{d-1}=0}^{l-1} (u_i v_j) \chi_i(R_1) \chi_j(R_2) \\ &= f_u(R_1) f_v(R_2). \end{aligned} \quad \blacktriangleleft$$

The essence of the proof is that we can obtain all the needed cross-terms corresponding to entries of uv^T from the product involving all terms in f_u and all terms in f_v .

We can employ the protocol for inner product using f_A and f_{uv^T} , which we can compute in the streaming phase, as $f_{uv^T} = f_u f_v$ to give us Theorem 3.

► **Theorem 8.** *Given $u, v \in \mathbb{F}^n$ and $A \in \mathbb{F}^{n \times n}$, we can verify $u^T Av$ using a $(d-1)$ round $(ld, l+d)$ -protocol for $n^2 = l^d$, with helper computation $O\left(\frac{n^2 \log(n)}{d}\right)$, verifier overhead $O(nld)$ and checking cost $O(l)$.*

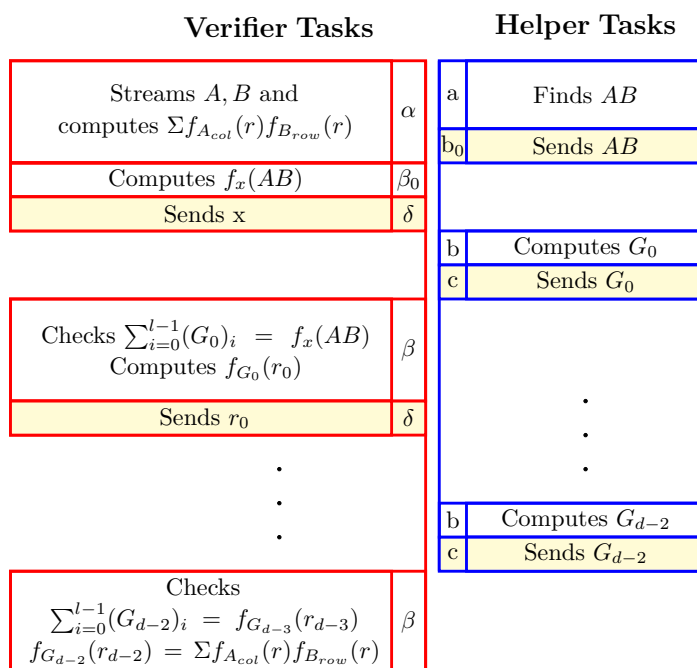


Figure 2 Detailed Matrix Multiplication Protocol.

5 Practical Analysis

To evaluate these protocols in practice, we focus on the core task of matrix multiplication. In order to discuss the time costs associated with execution of our protocols in more detail, we break down the various steps into components as illustrated in Figure 2. Here, we use Greek characters to describe the costs for the verifier: the initial streaming overhead ($t[\alpha]$), the checks performed in total in each round ($t[\beta]$), as well as the time to send responses ($t[\delta]$). For the helper, we identify four groups of tasks, denoted by Latin characters: the computation of the matrix product itself ($t[a]$), the communication of this result to the verifier ($t[b_0]$), and the time per round to compute and send the required message ($t[b]$ and $t[c]$ respectively).

Recall our discussion in Section 2 on the effects of communication bandwidth and latency on the optimal number of rounds. In our simple model we focused on the tasks most directly involved with communication (the verifier round cost $t[\delta]$ and helper round cost $t[c]$). We implicitly treated the corresponding round computation costs ($t[\beta]$ and $t[b]$) as nil. As the construction and sending of the solution ($t[a]$ and $t[b_0]$) will dominate the first stage of the protocol, we focus our experimental study on measuring values of $t[b]$, $t[b_0]$ and $t[\beta]$ to quantify a reasonable estimate for the length of time the interactive phase of the protocol takes with bandwidth \mathcal{B} and latency \mathcal{L} .

We account for the cost required for computation and communication separately to find the total time, T , as follows:

$$T = t[\text{work}] + t[\text{comm}] = (t[\beta_0] + t[\beta] + t[b]) + \left(2d\mathcal{L} + \frac{2dl \log(|\mathbb{F}|)}{\mathcal{B}} \right).$$

T is the total time for the protocol from receiving the answer to producing a conclusion of the veracity of the result. We can omit the verifier’s streaming computation time $t[\alpha]$ from the total protocol run time, as this can be overlapped with the helper’s computation of the true answer, which should always dominate.

48:12 Efficient Interactive Proofs for Linear Algebra

■ **Table 3** Interaction phase costs.

(a) $n = 2^{12}$, $t[\beta_0] = 149 \pm 15$ ms.

l	d	$t[b]$ (ms)	$t[\beta]$ (μ s)
2	12	0.230 ± 0.02	9 ± 2
4	6	0.120 ± 0.01	14 ± 1
8	4	0.099 ± 0.01	35 ± 7
16	3	0.097 ± 0.01	35 ± 7
64	2	0.110 ± 0.01	43 ± 5

(b) $n = 2^{16}$, $t[\beta_0] = 38.0 \pm 6.5$ s.

l	d	$t[b]$ (ms)	$t[\beta]$ (μ s)
2	16	3.5 ± 0.2	6 ± 1
4	8	2.0 ± 0.1	9 ± 1
16	4	1.6 ± 0.1	46 ± 3
256	2	1.8 ± 0.1	1700 ± 200

(c) $n = 2^{18}$, $t[\beta_0] = 603 \pm 63$ s.

l	d	$t[b]$ (ms)	$t[\beta]$ (μ s)
2	18	14.1 ± 0.9	6 ± 1
4	9	8.0 ± 0.5	11 ± 3
8	6	6.3 ± 0.5	30 ± 3
64	3	7.1 ± 0.6	270 ± 30
512	2	7.8 ± 0.7	6400 ± 650

■ **Table 4** Matrix Multiplication Timings.

n	$t[a]$ (s)
2^{10}	0.61 ± 0.06
2^{11}	5.61 ± 0.7
2^{12}	47.9 ± 4.3
2^{13}	403 ± 34

In what follows, we instantiate this framework and determine the costs of implementing protocols. These demonstrate that while computation cost for matrix multiplication ($t[a]$) grows superquadratically, the streaming cost ($t[\alpha]$) is linear in the input size n . The dominant cost during the protocol is $t[\beta_0]$, to fingerprint the claimed answer; other computational costs in the protocol are minimal. Factoring in the communication based on real-world latency and bandwidth costs, we conclude that latency dominates, and indeed we prefer to have fewer rounds. In all our experiments, the optimal number of rounds is just 2. Extrapolating to truly enormous values of n suggest that still three rounds would suffice.

5.1 Setup

The experiments were performed on a workstation with an Intel Core i7-6700 CPU @ 3.40GHz processor, and 16GB RAM. Our implementations were written in single-threaded C using the GNU Scientific Library with BLAS for the linear algebra, and FFTW3 library for the Fourier Transform. The programs were compiled with GCC 5.4.0 using the `-O3` optimization flag, under Linux (64-bit Ubuntu 16.04), with kernel 4.15.0. Timing was done using the `clock()` function for all readings except $t[\beta]$, which used `getrusage()` as the timings were so small.

For the various tests performed, the matrices and vectors were generated using the `C rand()` function. Note that the work of the protocols is not affected by the data values, so we are not much concerned with how the inputs are chosen. The arithmetic field used was \mathbb{F}_q with $q = 2^{31} - 1$ (larger fields, such as $q = 2^{61} - 1$ or $q = 2^{127} - 1$ could easily be substituted to obtain much lower probability of error, at a small increase in time cost). The work of the verifier and work of the helper were both simulated on the same machine.

■ **Table 5** Time taken for interactions (ping 20ms, bandwidth 100Mbps, $|\mathbb{F}|=2^{31}-1$).

n	l	d	Latency cost (ms)	Bandwidth cost (ms)
2^{12}	2	12	440	0.014
	4	6	200	0.012
	8	4	120	0.015
	16	3	80	0.020
	64	2	40	0.041
2^{16}	2	16	600	0.019
	4	8	280	0.018
	16	4	120	0.031
	256	2	40	0.163
2^{18}	2	18	680	0.022
	4	9	320	0.020
	8	6	200	0.026
	64	3	80	0.082
	512	2	40	0.328

■ **Table 6** Verifier matrix multiplication time (ping 20ms, bandwidth 100Mbps, $|\mathbb{F}|=2^{31}-1$).

n	l	d	t[comm] (s)	t[work] (s)	T (s)
2^{12}	2	12	0.44	0.149	0.589
	4	6	0.20		0.349
	8	4	0.12		0.269
	16	3	0.08		0.229
	64	2	0.04		0.189
2^{16}	2	16	0.60	38	38.6
	4	8	0.28		38.3
	16	4	0.12		38.1
	256	2	0.04		38.0
2^{18}	2	18	0.68	603	604
	4	9	0.32		603
	8	6	0.20		603
	64	3	0.08		603
	512	2	0.04		603

5.2 Matrix Multiplication Results

Table 3 shows the experimental results for the matrix multiplication protocol for matrix sizes ranging from $n = 2^{12}$ to 2^{18} . Note, this means we are tackling matrices with tens of billions of entries. For completeness, we timed BLAS matrix multiplication on our machine for $n = 2^{10}$ to 2^{13} to give an idea of the comparative magnitude of a (Table 4), although further results were restricted by machine memory. Due to memory limitations, we tested our algorithms using freshly drawn random values in place of stored values of the required vectors or matrices. This does not affect our ability to compare the data, and allows us to increase the data size beyond that of the machine memory.

The computation cost $t[a]$ grows with the cost of matrix multiplication, which is super-quadratic in n , while $t[\alpha]$ grows linearly with the size of the input, which is strictly quadratic in n . Further, the verifier does not need to retain whole matrices in memory, and can compute the needed quantities with a single linear pass over the input.

We next study the helper’s cost across all d rounds to compute the responses in each step of the protocol. Our analysis bounds this total cost as $O(\frac{n \log(n)}{d})$. However, we observe that in our experiments, this quantity tends to decrease as d decreases. We conjecture that while the cost does decrease each round, the amount of data needed to be handled quickly decreases to a point where it is cache resident, and the computation takes a negligible amount of time compared to the data access. Thus, this component of the helper’s time cost is driven by the number of rounds during which the relevant data is still “large”, which is greater for larger d .

When we look at the contributory factors to $t[\text{work}]$, we observe that the dominant term is by far $t[\beta_0]$, where the verifier reads through the claimed answer and computes the fingerprint. Thus, arguably, the *computational* cost of any such protocol once the prover finds the answer is dominated by the time the verifier takes to actually inspect the answer: all subsequent checks are minimal in comparison. This justifies our earlier modelling assumption to omit computational costs in our balancing of latency and bandwidth factors.

We now turn to the time due to communication, summarized in Table 5. Here, we can clearly see the huge difference of several orders of magnitude between the latency cost, $2d\mathcal{L}$, versus the bandwidth cost, $\frac{2dl \log(|\mathbb{F}|)}{B}$. Note that these timing figures are simulated, based on the average values of latency and the corresponding average bandwidth found when pinging

several cloud servers such as Google, Amazon and Microsoft from a university network. The dependencies on both latency and bandwidth are linear. Consequently, if the latency were reduced to 10ms, this would halve the times in the Latency cost column; similarly, if bandwidth were doubled, this would halve the times in the Bandwidth cost column. We observe then that for all but very low bandwidth scenarios, the latency cost will dominate.

Finally, we put these pieces together, and consider the total protocol time from both computation and communication components. We obtain the total time by summing $t[\text{work}]$ and $t[\text{comm}]$, in Table 6. These results confirm our earlier models, and the fastest time is achieved with a very small number of rounds. For all values of n tested in these experiments, we see the optimal value of d is 2, the minimally interactive scenario. The trend is such that, because of the sheer domination of latency and $t[\beta_0]$, it is unlikely that more than two or three rounds will ever be needed for even the largest data sets. As n increases, the size of $t[\text{work}]$ grows faster than $t[\text{comm}]$, predominantly due to $t[\beta_0]$. Therefore to minimize the cost of verification one should prefer a small constant number of rounds.

6 Concluding Remarks

Our experimental study supports the claim that fewer rounds of interaction are preferable to allow efficient interactive proofs for linear algebra primitives. For large instances in our experiments, the optimal number of rounds is just two. These primitives allow simple implementation of more complex tools such as regression and linear predictors [6]. Other primitive operations, such as scalar multiplication and addition, are trivial within this model (since LDE evaluations and fingerprints are linear functions), so these primitives collectively allow a variety of computations to be efficiently verified. Further operators, such as matrix (pseudo)inversion and factorization are rather more involved, not least since they bring questions of numerical precision and representation [6]. Nevertheless, it remains open to show more efficient protocols for other functions, such as matrix exponentiation, and to allow sequences of operations to be easily “chained together” to verify more complex expressions.

References

- 1 Scott Aaronson and Avi Wigderson. Algebrization: A New Barrier in Complexity Theory. *ACM Trans. Comput. Theory*, 1(1):2:1–2:54, February 2009. doi:10.1145/1490270.1490272.
- 2 Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.
- 3 Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive Oracle Proofs. Cryptology ePrint Archive, Report 2016/116, 2016. URL: <https://eprint.iacr.org/2016/116>.
- 4 Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, and Ron D. Rothblum. Fiat-Shamir From Simpler Assumptions. Cryptology ePrint Archive, Report 2018/1004, 2018. URL: <https://eprint.iacr.org/2018/1004>.
- 5 Amit Chakrabarti, Graham Cormode, and Andrew McGregor. Annotations in data streams. *Automata, Languages and Programming*, pages 222–234, 2009.
- 6 Graham Cormode and Chris Hickey. Cheap Checking for Cloud Computing: Statistical Analysis via Annotated Data Streams. In *International Conference on Artificial Intelligence and Statistics*, pages 1318–1326, 2018.
- 7 Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming interactive proofs. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 90–112. ACM, 2012.
- 8 Graham Cormode, Justin Thaler, and Ke Yi. Verifying computations with streaming interactive proofs. *Proceedings of the VLDB Endowment*, 5(1):25–36, 2011.

- 9 Samira Daruki, Justin Thaler, and Suresh Venkatasubramanian. Streaming verification in data analysis. In *International Symposium on Algorithms and Computation*, pages 715–726. Springer, 2015.
- 10 Rūsiņš Freivalds. Fast probabilistic algorithms. *Mathematical Foundations of Computer Science 1979*, pages 57–69, 1979.
- 11 Shafi Goldwasser, Yael Tauman Kalai, and Guy N Rothblum. Delegating computation: interactive proofs for muggles. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 113–122. ACM, 2008.
- 12 Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM (JACM)*, 39(4):859–868, 1992.
- 13 Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly Practical Verifiable Computation. *Commun. ACM*, 59(2):103–112, January 2016.
- 14 Michael O Rabin. *Fingerprinting by random polynomials*. Center for Research in Computing Techn., Aiken Computation Laboratory, Univ., 1981.
- 15 Srinath TV Setty, Richard McPherson, Andrew J Blumberg, and Michael Walfish. Making argument systems for outsourced computation practical (sometimes). In *NDSS*, volume 1, page 17, 2012.
- 16 Srinath TV Setty, Victor Vu, Nikhil Panpalia, Benjamin Braun, Andrew J Blumberg, and Michael Walfish. Taking Proof-Based Verified Computation a Few Steps Closer to Practicality. In *USENIX Security Symposium*, pages 253–268, 2012.
- 17 Adi Shamir. IP=PSPACE. *Journal of the ACM (JACM)*, 39(4):869–877, 1992.
- 18 Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In *Advances in Cryptology—CRYPTO 2013*, pages 71–89. Springer, 2013.
- 19 Victor Vu, Srinath Setty, Andrew J Blumberg, and Michael Walfish. A hybrid architecture for interactive verifiable computation. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 223–237. IEEE, 2013.

A Details of Proof of Lemma 5

► **Lemma 9** (Restatement of Lemma 5). *Given $a, b \in \mathbb{F}_p^n$ the sum*

$$a^T b = \sum_{k_0=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} f_a(k_0, \dots, k_{d-1}) f_b(k_0, \dots, k_{d-1})$$

can be verified using a $(d-1)$ -round $(ld, l+d)$ -protocol with helper overhead time $O(\frac{n \log(n)}{d})$, and verifier overhead time of $O(nld)$ and checking computation time $O(ld)$.

Proof. First, set

$$g(k_0, \dots, k_{d-1}) = f_a(k_0, \dots, k_{d-1}) f_b(k_0, \dots, k_{d-1})$$

$g : \mathbb{F}_q \times \dots \times \mathbb{F}_q \rightarrow \mathbb{F}_q$ is a degree $2l$ polynomial in each variable. Now, consider round $j+1$ of the sum-check protocol, where the helper is required to send

$$g_j(x) = \sum_{k_{j+1}=1}^l \cdots \sum_{k_d=1}^l g(r_1, \dots, r_{j-1}, x, k_{j+1}, \dots, k_d)$$

Here, g is degree $2l$ polynomial, sent to V as a set $G_j^\Sigma = \{(g_j(x), x) : x \in [2l]\}$. To compute this set we have H find the individual summands as

$$G_j = \left\{ (g(r_1, \dots, r_{j-1}, x, k_{j+1}, \dots, k_{d-1}), x) : x \in [2l], k_{j+1}, \dots, k_{d-1} \in [l] \right\}$$

Naive computation of all the values in G_j takes time $O(nd)$ each, for a total cost of $O(nl^{d-j}d)$. However, instead of computing the LDE at l^{d-j} points with cost $O(ld)$ we can sum l^{d-j} convolutions of length $2l$ vectors to obtain the same result (See below). The total cost of the convolution is $O(l \log(l)) = O(\frac{l \log(n)}{d})$, using $n = l^d$. Summing these l^{d-j} convolutions gives the cost of the j th round for the helper as $O(\frac{l^{d-j} \log(n)}{d})$. Summing over the d rounds gives us our cost of $O(\frac{n \log(n)}{d})$. ◀

A.1 Finding G_j with Convolution

To simplify the argument, we consider the computation of $a^T a$ (also referred to as F_2). The general case of $a^T b$ follows the same steps but the notation quickly becomes cumbersome. So, given a vector $a \in \mathbb{F}_q^n$, we want to find $\sum_{i=0}^{n-1} a_i^2$. This is equivalent to finding the inner product of a with itself.

Consider a $d - 1$ round protocol for the F_2 problem on $a \in \mathbb{F}_q^n$. We have $n = l^d$, and so for each round of interaction the helper sends

$$g_j(x) = \sum_{k_{j+1}=1}^l \cdots \sum_{k_{d-1}=1}^l f_A(r_0, \dots, r_{j-1}, x, k_{j+1}, \dots, k_{d-1})^2,$$

where the input is reshaped as the d -dimensional $A \in \mathbb{F}^{l \times l \times \dots \times l}$. There are $d - 1$ such polynomials to send over the course of the protocol, and each one has degree $2l - 1$.

Round 1

Consider first the opening round

$$g_0(x) = \sum_{k_1=1}^l \cdots \sum_{k_{d-1}=1}^l f_A(x, k_1, \dots, k_{d-1})^2$$

This can be found by materializing the set of values $G_0 = \left\{ (f_A(x, k_1, \dots, k_d), x) : x \in [2l], k_1, \dots, k_{d-1} \in [l] \right\}$, and then summing over k_1, \dots, k_d to obtain G_0^Σ .

For the first half of the G_0^Σ , the computation is closely linked to the original input, and so we can simply compute the partial sums

$$\sum_{k_1=1}^l \cdots \sum_{k_{d-1}=1}^l f_A(x, k_1, \dots, k_{d-1})^2.$$

These sums partition the input, so the total time is $O(n)$ to obtain the values for all $x \in [l]$.

However, for x values in the range $l + 1 \dots 2l$, we need to evaluate the LDE at locations not present in the original input. To avoid the higher cost associated with naive computation of all terms, we expand the definition of LDEs:

$$f_A(k_0, \dots, k_{d-1}) = \sum_{p_0=0}^{l-1} \cdots \sum_{p_{d-1}=0}^{l-1} A_{p_0 p_1 \dots p_{d-1}} \chi_{p_0 p_1 \dots p_{d-1}}(k_0, \dots, k_{d-1})$$

$$\chi_{p_0 p_1 \dots p_{d-1}}(k_0, \dots, k_{d-1}) = \prod_{j=0}^{d-1} \prod_{i=0, i \neq p_j}^{l-1} \frac{k_j - i}{p_j - i}$$

In what follows, we can make use of the fact that not all input values contribute to every LDE evaluation needed. We expand as follows:

$$\begin{aligned}
g_0(x) &= \sum_{k_1=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} f_A(x, k_1, \dots, k_{d-1}) \\
&= \sum_{k_1=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} \left(\sum_{p_0=0}^{l-1} \sum_{p_1=0}^{l-1} \cdots \sum_{p_{d-1}=0}^{l-1} \right. \\
&\quad \left. \left(A_{p_0 p_1 \dots p_{d-1}} \left[\prod_{i=0, i \neq p_0}^{l-1} \frac{x-i}{p_0-i} \right] \left[\prod_{i=0, i \neq p_1}^{l-1} \frac{k_1-i}{p_1-i} \right] \cdots \left[\prod_{i=0, i \neq p_{d-1}}^{l-1} \frac{k_{d-1}-i}{p_{d-1}-i} \right] \right) \right)^2 \\
&= \sum_{k_1=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} \left(\sum_{p_0=0}^{l-1} \left[A_{p_0 k_1 \dots k_{d-1}} \prod_{i=0, i \neq p_0}^{l-1} \frac{x-i}{p_0-i} \right] \right)^2 \\
&= \sum_{k_1=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} \left(\sum_{p_0=0}^{l-1} \left[\left(A_{p_0 k_1 \dots k_{d-1}} \prod_{i=0, i \neq p_0}^{l-1} \frac{1}{p_0-i} \right) \left(\prod_{i=0}^{l-1} (x-i) \right) \left(\frac{1}{x-p_0} \right) \right] \right)^2 \\
&= \sum_{k_1=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} \left(\left(\prod_{i=0}^{l-1} (x-i) \right) \sum_{p_0=0}^{l-1} \left[\left(A_{p_0 k_1 \dots k_{d-1}} \prod_{i=0, i \neq p_0}^{l-1} \frac{1}{p_0-i} \right) \left(\frac{1}{x-p_0} \right) \right] \right)^2
\end{aligned}$$

Note in the second step we use that

$$\sum_{p_j=0}^{l-1} \prod_{i=0, i \neq p_j}^{l-1} \frac{k_j-i}{p_j-i} = \begin{cases} 0 & p_j \neq k_j \\ 1 & p_j = k_j \end{cases}$$

We now introduce the helper functions

$$g(p) = \frac{1}{p} \quad ; \quad h(x) = \prod_{i=1}^l (x-i) \quad \text{and} \quad q(p) = \prod_{i=0, i \neq p}^{l-1} \frac{1}{p-i} \quad (6)$$

to simplify the notation. We define the vectors

$$b_{k_1 \dots k_{d-1}}(p) = \begin{cases} A_{p, k_1 \dots k_{d-1}} q(p) & \text{for } p \in [0, l-1], k_1, \dots, k_{d-1} \in [0, l-1] \\ 0 & \text{for } p \in [l, 2l-1], k_1, \dots, k_{d-1} \in [0, l-1] \end{cases}$$

and use these to rewrite in terms of convolutions

$$\begin{aligned}
g_0(x) &:= \sum_{k_1=1}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} \left(h(x) \sum_{p_0=0}^{l-1} [b_{k_1 \dots k_{d-1}}(p_0) g(x-p_0)] \right)^2 \\
&= h(x)^2 \sum_{k_1=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} (\text{conv}(b_{k_1 \dots k_{d-1}}, g)[x])^2 \\
&= h(x)^2 \left(\sum_{k_2=1}^l \cdots \sum_{k_d=1}^l \text{DFT}^{-1}(\text{DFT}(b_{k_1 \dots k_{d-1}}) \cdot \text{DFT}(g)) \right) [x]^2.
\end{aligned}$$

Thus, by precomputing some arrays of values, we reduce the computation to several convolutions that can be evaluated quickly via fast Fourier transform. Observe that this FFT does not need to be computed over the same field as the matrix multiplication: we can choose any suitably large field for which there is an FFT (say, real vectors of size 2^j for some j), and then map the result back into \mathbb{F}_q . Forming $b_{k_1 \dots k_d}(p)$ takes time $O(l^d)$. We have to do $O(l^{d-1})$ convolutions on vectors of length $O(l)$, so each convolution takes time $O(l \log(l))$. Since $\log(l) = \log(n^{\frac{1}{d}})$, we can write the helper's time cost for the first round as $O(\frac{n}{d} \log(n))$.

Round j

Similar rewritings are possible in subsequent rounds. Initially, it may seem that things are more complex for G_j , as each $f_A(r_0, \dots, r_{j-1}, x, k_{j+1}, \dots, k_{d-1})$ appears to require full inspection of the input to evaluate at (r_0, \dots, r_{j-1}) . However, we can again define an ancillary array $b_{k_1 \dots k_{d-1}}$ to more easily compute this. In the sum-check protocol after the helper sends G_0 , it receives r_0 , with which we define the array over $[l]^{d-1}$:

$$A_{r_0 k_1 \dots k_{d-1}}^{(1)} = \sum_{p=0}^{l-1} b_{k_1 \dots k_{d-1}}(p) \prod_{i=0, i \neq p}^{l-1} (r_0 - i)$$

This allows the Helper to form G_1 using the same idea as above, but with $A^{(1)}$ instead of A . Working in terms of $A^{(1)}$ reduces the Helper's cost from $O(l^{d-1}ld)$ for computing the $f_A(r_0, k_1, \dots, k_{d-1})$ for each $k_i \in [l]$ to just $O(l^2)$ when combined with using $b_{k_1 \dots k_{d-1}}$.

In more detail, and with more generality, let us consider the j th round, where we are forming G_j and G_j^Σ . We define

$$A_{r_0, \dots, r_{j-1}, k_j \dots k_{d-1}}^{(j)} = \sum_{p=0}^{l-1} b_{k_j \dots k_{d-1}}(p) \prod_{i=0, i \neq p}^{l-1} (r_{j-1} - i)$$

Then we have the following computation for $x \in [l, 2l - 1]$:

$$\begin{aligned} g_j(x) &= \sum_{k_{j+1}=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} f_A(r_0, \dots, r_{j-1}, x, k_{j+1}, \dots, k_{d-1})^2 \\ &= \sum_{k_{j+1}=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} \left(\sum_{p_0=0}^{l-1} \cdots \sum_{p_{d-1}=0}^{l-1} \left(A_{p_0 \dots p_{d-1}} \left[\prod_{i=0, i \neq p_0}^{l-1} \frac{r_0 - i}{p_0 - i} \right] \cdots \left[\prod_{i=0, i \neq p_{j-1}}^{l-1} \frac{r_{j-1} - i}{p_{j-1} - i} \right] \right. \right. \\ &\quad \left. \left. \left[\prod_{i=0, i \neq p_j}^{l-1} \frac{x - i}{p_j - i} \right] \left[\prod_{i=0, i \neq p_{j+1}}^{l-1} \frac{k_{j+1} - i}{p_{j+1} - i} \right] \cdots \left[\prod_{i=0, i \neq p_{d-1}}^{l-1} \frac{k_{d-1} - 1}{p_{d-1} - 1} \right] \right) \right)^2 \\ &= \sum_{k_{j+1}=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} \left(\sum_{p_j=0}^{l-1} \left[A_{r_0 \dots r_{j-1} p_j k_{j+1} \dots k_{d-1}}^{(j)} \prod_{i=0, i \neq p_j}^{l-1} \frac{x - i}{p_j - i} \right] \right)^2 \\ &= \sum_{k_{j+1}=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} \left(\sum_{p_j=0}^{l-1} \left[\left(A_{r_0 \dots r_{j-1} p_j k_{j+1} \dots k_{d-1}}^{(j)} \prod_{i=0, i \neq p_j}^{l-1} \frac{1}{p_j - i} \right) \left(\prod_{i=0}^{l-1} (x - i) \right) \left(\frac{1}{x - p_j} \right) \right] \right)^2 \\ &= \sum_{k_{j+1}=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} \left(\left(\prod_{i=0}^{l-1} (x - i) \right) \sum_{p_j=0}^{l-1} \left[\left(A_{r_0 \dots r_{j-1} p_j k_{j+1} \dots k_{d-1}}^{(j)} \prod_{i=0, i \neq p_j}^{l-1} \frac{1}{p_j - i} \right) \left(\frac{1}{x - p_j} \right) \right] \right)^2. \end{aligned}$$

We make use of the same set of helper functions specified in equation (6), and define the vectors

$$b_{k_{j+1} \dots k_d}(p) = \begin{cases} A_{r_0 \dots r_{j-1} p k_{j+1} \dots k_{d-1}}^{(j)} q(p) & \text{for } p \in [0, l-1], k_{j+1}, \dots, k_d \in [0, l-1] \\ 0 & \text{for } p \in [l, 2l-1], k_{j+1}, \dots, k_{d-1} \in [0, l-1] \end{cases}$$

We can now continue to express the computation in terms of convolutions

$$\begin{aligned}
g_j(x) &:= \sum_{k_{j+1}=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} \left(h(x) \sum_{p_j=0}^{l-1} [b_{k_{j+1}\dots k_{d-1}}(p_j)g(x-p_j)] \right)^2 \\
&= \sum_{k_{j+1}=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} (h(x) \text{conv}(b_{k_{j+1}\dots k_{d-1}}, g)[x])^2 \\
&= h(x)^2 \left(\sum_{k_{j+1}=0}^{l-1} \cdots \sum_{k_{d-1}=0}^{l-1} \text{DFT}^{-1}(\text{DFT}(b_{k_j\dots k_d}) \cdot \text{DFT}(g)) \right) [x]^2.
\end{aligned}$$

We can think of $A^{(j)}$ as a shrinking input array, where $A^{(j)} \in \mathbb{F}^{l \times l \times \dots \times l}$ is $d-j$ dimensional, and

$$\begin{aligned}
b_{k_{j+1}\dots k_d}(p_j) &= A_{r_1\dots r_{j-1}p_j k_{j+1}\dots k_d}^{(j)} \prod_{i=1, i \neq p_j}^l \frac{1}{p_j - i} \\
A_{r_0, \dots, r_{j-1}, k_j, \dots, k_{d-1}}^{(j)} &= \sum_{p_{j-1}=0}^{l-1} A_{r_1\dots r_{j-2}p_{j-1}k_j\dots k_d}^{(j-1)} \prod_{i=0, i \neq p_{j-1}}^{l-1} \frac{r_{j-1} - i}{p_{j-1} - i}.
\end{aligned}$$

Using this formulation, the dominant computation cost in round j will be from the FFT, which involves l^{d-j-1} convolutions of cost $O(\frac{l}{d} \log(n))$ each. Thus the final cost for the round is $O(\frac{l^{d-j}}{d} \log(n))$. The cost of running the entire protocol requires $d-1$ rounds, making the computational cost for the helper

$$O\left(\sum_{j=0}^{d-2} \frac{l^{d-j}}{d} \log(n)\right) = O\left(n \log(n) \frac{\sum_{j=0}^{d-2} l^{-j}}{d}\right) = O\left(\frac{n \log(n)}{d}\right)$$

since $l \geq 2$. Note that when $d = \log(n)$ and $l = 2$, we achieve $O(n)$ time for the helper. The cost increases with fewer rounds, up to a maximum of $O(n \log n)$ for a constant round protocol.

Cost summary

For the verifier, the checking computation cost is $O(ld)$, which emerges from the d rounds, where in each round the verifier sums the first l elements of G_j^Σ , before evaluating the LDE of G_j^Σ at r_j , making for a total cost of $O(l)$. The streaming overhead for the verifier involves evaluating the LDE of the input A , for a cost of $O(nld)$. The verifier requires $O(l+d)$ memory to find the LDE of a at $r \in \mathbb{F}^d$. The communication will be $O(ld)$ as we have the helper sending d sets G_j of size $O(l)$. Hence, we summarize the various costs as

Rounds $d-1$

Communication $O(ld)$

Verifier Memory $O(l+d)$

Helper Computation Time $O(\frac{n \log(n)}{d})$

Verifier Overhead Time $O(nld)$

Verifier Checking Computation Time $O(ld)$

When Maximum Stable Set Can Be Solved in FPT Time

Édouard Bonnet 

Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France
edouard.bonnet@ens-lyon.fr

Nicolas Bousquet

CNRS, G-SCOP laboratory, Grenoble-INP, France
nicolas.bousquet@grenoble-inp.fr

Stéphan Thomassé

Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France
Institut Universitaire de France
stephan.thomasse@ens-lyon.fr

Rémi Watrigant 

Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France
remi.watrigant@ens-lyon.fr

Abstract

MAXIMUM INDEPENDENT SET (MIS for short) is in general graphs the paradigmatic $W[1]$ -hard problem. In stark contrast, polynomial-time algorithms are known when the inputs are restricted to structured graph classes such as, for instance, perfect graphs (which includes bipartite graphs, chordal graphs, co-graphs, etc.) or claw-free graphs. In this paper, we introduce some variants of co-graphs with *parameterized noise*, that is, graphs that can be made into disjoint unions or complete sums by the removal of a certain number of vertices and the addition/deletion of a certain number of edges per incident vertex, both controlled by the parameter. We give a series of FPT Turing-reductions on these classes and use them to make some progress on the parameterized complexity of MIS in H -free graphs. We show that for every fixed $t \geq 1$, MIS is FPT in $P(1, t, t, t)$ -free graphs, where $P(1, t, t, t)$ is the graph obtained by substituting all the vertices of a four-vertex path but one end of the path by cliques of size t . We also provide randomized FPT algorithms in dart-free graphs and in cricket-free graphs. This settles the FPT/ $W[1]$ -hard dichotomy for five-vertex graphs H .

2012 ACM Subject Classification Theory of computation \rightarrow Graph algorithms analysis; Theory of computation \rightarrow Fixed parameter tractability

Keywords and phrases Parameterized Algorithms, Independent Set, H -Free Graphs

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.49

Related Version A full version of the paper is available at <http://arxiv.org/abs/1909.08426>.

1 Introduction

A *stable set* or *independent set* in a graph is a subset of vertices which are pairwise non-adjacent. Finding an independent set of maximum cardinality, called MAXIMUM INDEPENDENT SET (or MIS for short), is not only NP-hard to solve [19] but also to approximate within ratio $n^{1-\varepsilon}$ [24, 39]. One can then wonder if efficient algorithms exist with the additional guarantee that k , the size of the maximum stable set, is fairly small compared to n , the number of vertices of the input (think, for instance, $k \leq \log n$). It turns out that, for any computable function $h = \omega(1)$ (but whose growth can be arbitrarily slow), MIS is unlikely to admit a polynomial-time algorithm even when $k \leq h(n)$. In parameterized complexity terms, MIS is $W[1]$ -hard [17]. More quantitatively, MIS cannot be solved in time $f(k)n^{o(k)}$ for any computable function f , unless the Exponential Time Hypothesis fails. This is quite



© Édouard Bonnet, Nicolas Bousquet, Stéphan Thomassé, and Rémi Watrigant;
licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 49; pp. 49:1–49:22

Leibniz International Proceedings in Informatics



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

a statement when a trivial algorithm for MIS runs in time n^{k+2} , and a simple reduction to triangle detection yields a $n^{\frac{\omega k}{3} + O(1)}$ -algorithm, where ω is the best exponent known for matrix multiplication.

It thus appears that MIS on general graphs is totally impenetrable. This explains why efforts have been made on solving MIS in subclasses of graphs. The most emblematic result in that line of works is a polynomial-time algorithm in perfect graphs [21]. Indeed, perfect graphs generalize many graph classes for which MIS is in P: bipartite graphs, chordal graphs, co-graphs, etc. In this paper, we put the focus on classes of graphs for which MIS can be solved in FPT time (rather than in polynomial-time). For graphs with bounded degree Δ , the simple branching algorithm has FPT running time $(\Delta + 1)^k n^{O(1)}$. The same observation also works more generally for graphs with bounded average degree, or even d -degenerate graphs. A non-trivial result is that MIS remains FPT in arguably the most general class of sparse graphs, *nowhere dense graphs*. Actually, deciding first-order formulas of size k can be done in time $f(k)n^{1+\varepsilon}$ on any nowhere dense class of graphs [20]. Since MIS and the complement problem, MAXIMUM CLIQUE, are both expressible by a first-order formula of length $O(k^2)$, $\exists v_1, \dots, v_k \bigwedge_{i,j} (\neg)E(v_i, v_j)$, there is an FPT algorithm on nowhere dense graphs and also on complements of nowhere dense graphs. A starting point of the present paper is to design FPT Turing-reductions in classes containing both very dense and very sparse graphs.

Co-graphs with parameterized noise. If G and H are two graphs, we can define two new graphs: $G \cup H$, their disjoint union, and $G \oplus H$ their (complete) sum, obtained from the disjoint union by adding all the edges from a vertex of G to a vertex of H . Then, the hereditary class of co-graphs can be inductively defined by: K_1 (an isolated vertex) is a co-graph, and $G \cup H$ and $G \oplus H$ are co-graphs, if G and H are themselves co-graphs. So the construction of a co-graph can be seen as a binary tree whose internal nodes are labeled by \cup or \oplus , and leaves are K_1 . Finding the tree of operations building a given co-graph, sometimes called *co-tree*, can be done in linear time [11]. This gives a simple algorithm to solve MIS on co-graphs: $\alpha(K_1) = 1$, $\alpha(G \cup H) = \alpha(G) + \alpha(H)$, and $\alpha(G \oplus H) = \max(\alpha(G), \alpha(H))$.

We add a *parameterized noise* to the notion of co-graphs. More precisely, we consider graphs that can be made disjoint unions or complete sums by the deletion of $g(k)$ vertices and the edition (*i.e.*, addition or deletion) of $d(k)$ edges per incident vertex. We design a series of FPT Turing-reductions on several variants of these classes using the so-called *iterative expansion* technique [10, 4], Cauchy-Schwarz-like inequalities, and Kővári-Sós-Turán's theorem. This serves as a crucial foundation for the next part of the paper, where we explore the parameterized complexity of MIS in H -free graphs (*i.e.*, graphs not containing H as an induced subgraph). However, we think that the FPT routines developed on *co-graphs with parameterized noise* may also turn out to be useful outside the realm of H -free graphs.

Classical and parameterized dichotomies in H -free graphs. The question of whether MIS is in P or NP-complete in H -free graphs, for each connected graph H , goes back to the early eighties. However, a full dichotomy is neither known nor does it seem within reach in the near future. For three positive integers i, j, k , let $S_{i,j,k}$ be the tree with exactly one vertex of degree three, from which start three paths with i, j , and k edges, respectively. The claw is the graph $S_{1,1,1}$, thus $\{S_{i,j,k}\}_{i \leq j \leq k}$ is the set of all the subdivided claws. We denote by P_ℓ the path on ℓ vertices.

If G' is the graph obtained by subdividing each edge of a graph G exactly $2t$ times, Alekseev observed that $\alpha(G') = \alpha(G) + t|E(G)|$ [1]. This shows that MIS remains NP-hard on graphs which locally look like paths or subdivided claws (one can perform the subdivision on sub-cubic graphs G , on which MIS remains NP-complete). In other words, if a connected graph H is not a path nor a subdivided claw then MIS is NP-complete

on H -free graphs [1]. The MIS problem is easy on P_4 -free graphs, which are exactly the co-graphs. Already on P_5 -free graphs, a polynomial algorithm is much more difficult to obtain. This was done by Lokshtanov et al. [28] using the framework of potential maximal cliques. A quasi-polynomial algorithm was proposed for P_6 -free graphs [27], and recently, a polynomial-time algorithm was found by Grzesik et al. [22]. Brandstädt and Mosca showed how to solve MIS in polynomial-time on $(P_7, \text{triangle})$ -free graphs [8]. This result was then generalized by the same authors on $(S_{1,2,4}, \text{triangle})$ -free graphs [9], and by Maffray and Pastor on (P_7, bull^1) -free graphs (as well as $(S_{1,2,3}, \text{bull})$ -free graphs) [33]. Bacsó et al. [3] presented a subexponential-time $2^{O(\sqrt{tn \log n})}$ -algorithm in P_t -free graphs, for every integer t . Nevertheless, the classical complexity of MIS remains wide open on P_t -free graphs, for $t \geq 7$.

On claw-free graphs MIS is known to be polynomial-time solvable [36, 37]. Recently, this result was generalized to ℓ claw-free graphs [7] (where ℓ claw is the disjoint union of ℓ claws). On fork-free graphs (the fork is $S_{1,1,2}$) MIS admits a polynomial-time algorithm [2], and so does its weighted variant [31]. The complexity of MIS is open for $S_{1,1,3}$ -free graphs and $S_{1,2,2}$ -free graphs, and there is no triple $i \leq j \leq k$, for which we know that MIS is NP-hard on $S_{i,j,k}$ -free graphs. Some subclasses of $S_{i,j,k}$ -free graphs are known to admit polynomial algorithms for MIS: for instance $(S_{1,1,3}, K_{t,t})$ -free graphs [15], subcubic $S_{2,t,t}$ -free graphs [23] (building upon [32], and generalizing results presented in [34, 35] for subcubic *planar* graphs), bounded-degree $tS_{1,t,t}$ -free graphs [30], for any fixed positive integer t . This leads to the following conjecture:

- **Classical MIS Dichotomy Conjecture(H).** *For every connected graph H ,*
 MAXIMUM INDEPENDENT SET *in H -free graphs is in P iff $H \in \{P_\ell\}_\ell \cup \{S_{i,j,k}\}_{i \leq j \leq k}$.*

An even stronger conjecture is postulated by Lozin (see Conjecture 1 in [29]). Dabrowski et al. initiated a systematic study of the parameterized complexity of MIS on H -free graphs [13, 14]. In a nutshell, parameterized complexity aims to design $f(k)n^{O(1)}$ -algorithms (FPT algorithm, for Fixed-Parameter Tractable), where n is the size of the input, and k is the size of the solution (or another well-chosen parameter), for most often NP-hard problems. The so-called W -hierarchy (and in particular, $W[1]$ -hardness) and the Exponential Time Hypothesis (ETH) both provide a framework to rule out such a running time. We refer the interested reader to two recent textbooks [17, 12] and to a survey on the ETH and its consequences [26]. In the language of parameterized complexity, the dichotomy problem is the following:

- **Parameterized MIS Dichotomy(H).** *Is MIS (randomized) FPT or $W[1]$ -hard in H -free graphs?*

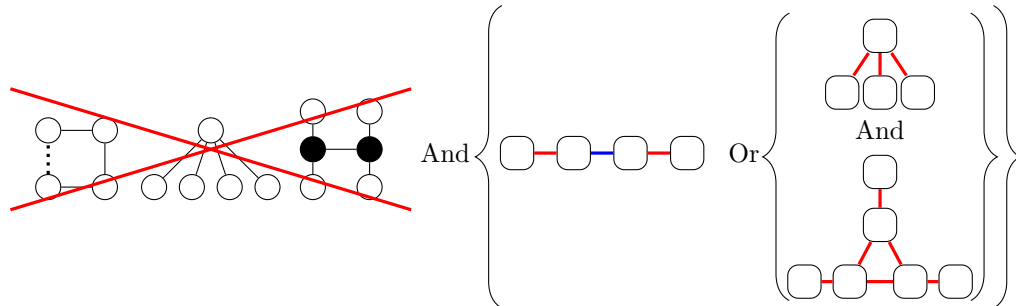
This question may be even more challenging than its classical counterpart. Indeed, there is no FPT algorithm known for the classical open cases: P_7 -, $S_{1,1,3}$ -, and $S_{1,2,2}$ -free graphs. Besides, the reduction of Alekseev [1] that we mentioned above does not show $W[1]$ -hardness. Thus, there are *a priori* more candidates H for which the parameterized status of MIS is open. For instance, by Ramsey's theorem, MIS is FPT on K_t -free graphs, for any fixed t . Observe that a randomized FPT algorithm for a $W[1]$ -hard problem is highly unlikely, as it would imply a randomized algorithm solving 3-SAT in subexponential time.

Dabrowski et al. showed that MIS is FPT² in H -free graphs, for all H on four vertices, except $H = C_4$ (the cycle on four vertices). Thomassé et al. presented an FPT algorithm on bull-free graphs [38], whose running time was later improved by Perret du Cray and Sau [18].

¹ The bull is obtained by adding a pendant neighbor to two distinct vertices of the triangle (K_3).

² Here and in what follows, the parameter is the size of the solution.

Bonnet et al. provided three variants of a parameterized counterpart of Alekseev’s reduction [4, 5]. Although the description of the open cases (see Figure 1) is not nearly as nice and compact as for the classical dichotomy, it is noteworthy that they almost correspond to paths and subdivided claws where vertices are blown-up into cliques.



■ **Figure 1** The dotted edge represents a path with at least one edge. The filled vertices emphasize two vertices with degree at least three in a tree. The rounded boxes are cliques. A red edge corresponds to a complete bipartite minus at most one edge. A blue edge correspond to a $2K_2$ -free bipartite graph. The FPT connected candidates H have to be chordal, without induced $K_{1,4}$ or trees with two branching vertices (*i.e.*, vertices of degree at least 3), and have to fit on a path with at most one blue edge (and the rest of red edges) or both in a subdivided claw and a line-graph of a subdivided claw with red edges only. A further restriction in the line-graph of subdivided claw is that three vertices each in a different clique of the triangle of red edges cannot induce a $K_1 \cup K_2$ (see [4]).

Let us make that idea more formal. Substituting a graph H at a vertex v of a graph G gives a new graph with vertex set $(V(G) \setminus \{v\}) \cup V(H)$, and the same edges as in G and H , plus all the edges xy where $x \in V(H)$, $y \in V(G)$, and $vy \in E(G)$. For a sequence of positive integers a_1, a_2, \dots, a_ℓ , we denote by $P(a_1, a_2, \dots, a_\ell)$ the graph obtained by substituting a clique K_{a_i} at the i -th vertex of a path P_ℓ , for every $i \in [\ell]$. We also denote by $T(a, b, c)$ the graph obtained by substituting a clique K_a, K_b , and K_c to the first, second, and third leaves, respectively, of a claw. Thus, $T(1, 1, 1)$ is the claw and $T(1, 1, 2)$ is called the cricket (see Figure 3d).

We show in this paper that MIS is (randomized) FPT in $T(1, 1, 2)$ -free graphs (or cricket-free graphs). This is in sharp contrast with the $W[1]$ -hardness for $T(1, 2, 2)$ -free graphs [5] (see Figure 2e). It disproves a seemingly reasonable conjecture that FPTness is preserved by adding a true twin to a vertex of H . We thus have a fairly good understanding of the parameterized complexity of MIS when H is obtained by substituting cliques on a claw. We therefore turn towards the graphs H obtained by substituting cliques on a path. MIS was shown FPT on $P(t, t, t)$ -free graphs [4]. A natural next step is to attack the following conjecture.

► **Conjecture 1.** *For any integer t , MIS can be solved in FPT time in $P(t, t, t)$ -free graphs.*

We denote by $P_\ell(t)$ the graph $P(t, t, \dots, t)$ where the sequence t, t, \dots, t is of length ℓ . We further conjecture the following, which is a far more distant milestone.

► **Conjecture 2.** *For any integers t and ℓ , MIS is FPT in $P_\ell(t)$ -free graphs.*

Let us recall though that the parameterized complexity of MIS is open in P_7 -free graphs, and no *easy* FPT algorithm is known on P_5 -free graphs. In general, we believe that there will be very few connected candidates (as described by Figure 1) which will not end up in (randomized) FPT. As a first empirical evidence, we show that the four candidates remaining among the 34 graphs on five vertices indeed all lead to (randomized) FPT algorithms.

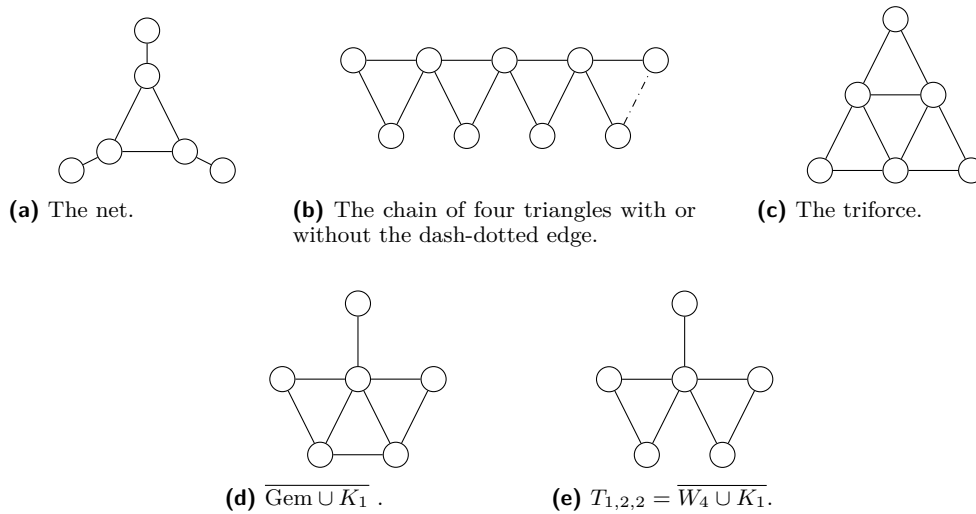


Figure 2 Some connected chordal $K_{1,4}$ -free graphs H for which H -free MIS is $W[1]$ -hard (see [4]). These graphs do not fit the candidate forms of Figure 1 for subtle reasons and illustrate how delicate the parameterized dichotomy promises to be. In particular, observe that MIS is $W[1]$ -hard on $T(1, 2, 2)$ -graphs, whereas we will show in this paper that it is FPT in $T(1, 1, 2)$ -graphs (a.k.a. cricket-free graphs).

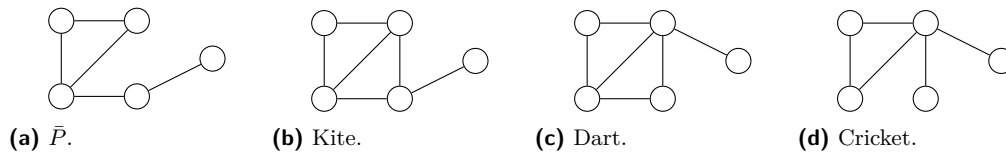


Figure 3 The four (out of 34) remaining cases on five vertices for the FPT/ $W[1]$ -hard dichotomy (see [5]). In this paper, we come up with new tools and solve all of them in (randomized) FPT.

Organization of our results. The rest of the paper is organized as follows. In Section 2, we introduce FPT Turing-reductions relevant to the subsequent section. In Section 3, we give a series of FPT algorithms in far-reaching generalizations of co-graphs: graphs where the deletion of $g(k)$ vertices leads to a separation which is either very sparse or very dense, in a way that is controlled by the parameter. In this section the proofs of two lemmas and one theorem, marked with a \star symbol, are deferred to the appendix. In Section 4, we use these results to obtain an FPT algorithm on $P(1, t, t, t)$ -free graphs for any positive integer t , taking a stab at Conjecture 1. Observe that this result settles at the same time \bar{P} ($=P(1, 1, 1, 2)$) and the kite ($=P(1, 1, 2, 1)$). The pseudo-code of the algorithm can be found in the appendix. In Section 5, we finish the FPT/ $W[1]$ -hard classification for five-vertex graphs by designing randomized FPT algorithms on dart-free graphs and cricket-free graphs. These results are marked with a \spadesuit symbol, which means that their proof can only be found in the long version of the paper [6].

We believe that the results of Section 3 as well as the techniques developed in Sections 4 and 5 may help in settling Conjecture 1. For $P(t, t, t, t)$ -free graphs, it is possible that one will have to combine the framework of potential maximal cliques with our techniques. To solve Conjecture 2, let alone the full parameterized dichotomy, some new ideas will be needed. The FPT algorithms of the current paper merely serve for classification purposes, and are not practical. A possible line of work is to get improved running times for the already established FPT cases. We also hope that the results of Section 3 will prove useful in a context other than H -free graphs.

2 Preliminaries

Here, we introduce some basics about graph notations, Ramsey numbers, and FPT algorithms.

2.1 Notations

For any pair of integers $i \leq j$, we denote by $[i, j]$ the set of integers $\{i, i + 1, \dots, j - 1, j\}$, and for any positive integer i , $[i]$ is a shorthand for $[1, i]$. We use the standard graph terminology and notations [16]. All our graphs are finite and simple, *i.e.*, they have no multiple edge nor self-loop. For a vertex v , we denote by $N(v)$ the set of neighbors of v , and $N[v] := N(v) \cup \{v\}$. For a subset of vertices S , we set $N(S) := \bigcup_{v \in S} N(v) \setminus S$ and $N[S] := N(S) \cup S$. The *degree* (resp. *co-degree*) of a vertex v is $|N(v)|$ (resp. $|V \setminus N[v]|$). If G is a graph and X is a subset of its vertices, $G[X]$ is the subgraph induced by X and $G - X$ is a shorthand for $G[V(G) \setminus X]$. We denote by $\alpha(G)$ the independence number, that is the size of a maximum independent set. If H and G are two graphs, we write $H \subseteq_i G$ to mean that H is an induced subgraph of G , and $H \subset_i G$ if H is a proper induced subgraph of G . We denote by K_ℓ , P_ℓ , C_ℓ , the clique, path, cycle, respectively, on ℓ vertices, and by $K_{s,t}$ the complete bipartite graph with s vertices on one side and t , on the other. The claw is $K_{1,3}$, and the paw is the graph obtained by adding one edge to the claw. If H is a graph and t is a positive integer, we denote by tH the graph made of t disjoint copies of H . For instance, $2K_2$ corresponds to the disjoint union of two edges. We say that a class of graphs \mathcal{C} is *hereditary* if it is closed by induced subgraph, *i.e.*, $\forall H, G, (G \in \mathcal{C} \wedge H \subseteq_i G) \Rightarrow H \in \mathcal{C}$.

2.2 Ramsey numbers

For two positive integers a and b , $R(a, b)$ is the smallest integer such that any graph with at least that many vertices has an independent set of size a or a clique of size b . By Ramsey's theorem, $R(a, b)$ always exists and is no greater than $\binom{a+b}{a}$. For the sake of convenience, we set $Ram(a, b) := \binom{a+b}{a} = \binom{a+b}{b}$. We will use repeatedly a constructive version of Ramsey's theorem.

► **Lemma 3** (folklore). *Let a and b be two positive integers, and let G be a graph on at least $Ram(a, b)$ vertices. Then an independent set of size a or a clique of size b can be found in linear time.*

Proof. We show this lemma by induction on $a + b$. For $a = 1$ (or $b = 1$), any vertex of G works (it is a clique and an independent set at the same time). And G is non-empty since it has at least $\binom{1+b}{1}$ (or $\binom{a+1}{1}$) vertices. We assume $a, b \geq 2$ and consider any vertex v of G . Let $G_1 := G - N[v]$ and $G_2 := G[N(v)]$, so $|V(G)| = 1 + |V(G_1)| + |V(G_2)|$.

Since $|V(G)| \geq \binom{a+b}{a} = \binom{a+b-1}{a-1} + \binom{a+b-1}{a}$, it cannot be that both $|V(G_1)| \leq Ram(a - 1, b) - 1$ and $|V(G_2)| \leq Ram(a, b - 1) - 1$. If G_1 has at least $Ram(a - 1, b)$ vertices, we find by induction an independent set I of size $a - 1$ or a clique of size b . Thus $I \cup \{v\}$ is an independent set of size a in G . If instead G_2 has at least $Ram(a, b - 1)$ vertices, we find by induction an independent set of size a or a clique C of size $b - 1$. Thus $C \cup \{v\}$ is an independent set of size b in G . ◀

For two positive integers a and b , we denote by $Ram_a(b)$ the smallest integer n such that any edge-coloring of K_n with a colors has a monochromatic clique of size b . In particular, $Ram_2(b) = Ram(b, b)$ (one color for the edges, and one color for the non-edges). Again, $Ram_a(b)$ always exists and a monochromatic clique of size b in an a -edge-colored clique of size at least $Ram_a(b)$ can be found in polynomial-time (whose exponent does *not* depend on a and b).

2.3 FPT Turing-reductions

For an instance (I, k) of MIS, let $\text{yes}(I, k)$ be the Boolean function which equals *True* if and only if (I, k) is a positive instance.

► **Definition 4.** A decreasing FPT g -Turing-reduction is an FPT algorithm which, given an instance (I, k) , produces $\ell := g(k)$ instances $(I_1, k_1), \dots, (I_\ell, k_\ell)$, for some computable function g , such that:

- (i) $\text{yes}(I, k) \Leftrightarrow \phi(\text{yes}(I_1, k_1), \dots, \text{yes}(I_\ell, k_\ell))$, where ϕ is a fixed FPT-time checkable formula³,
- (ii) $|I_j| \leq |I|$ for every $j \in [\ell]$, and
- (iii) $k_j \leq k - 1$ for every $j \in [\ell]$.

Note that conditions (ii) and (iii) prevent the instance size from increasing and force the parameter to strictly decrease, respectively.

► **Lemma 5.** Assume there is a decreasing FPT g -Turing-reduction for MIS on every input $(G \in \mathcal{C}, k)$, running in time $h(k)|V(G)|^\gamma$ (this includes the time to check ϕ). Let $f : [k - 1] \rightarrow \mathbb{N}$ be a non-decreasing function. If any instance (H, k') with $k' < k$ can be solved in time $f(k')|V(H)|^c$ with $c \geq \gamma$, then MIS can be solved in FPT time $f(k)|V(G)|^c$ in \mathcal{C} , with $f(k) := h(k) + g(k)f(k - 1)$.

Proof. We show the lemma by induction. If $k = 1$, this is immediate. We therefore assume that $k \geq 2$. We apply the decreasing FPT g -Turing-reduction to (G, k) . That creates at most $g(k)$ instances with parameter at most $k - 1$. We solve each instance in time $f(k - 1)n^c$ with $n := |V(G)|$. The overall running time is bounded by $h(k)n^\gamma + g(k)f(k - 1)n^c \leq f(k)n^c$ by extending the partial function f with $f(k) := h(k) + g(k)f(k - 1)$. ◀

This corollary follows by induction on the parameter k .

► **Corollary 6.** If MIS admits a decreasing FPT g -Turing-reduction on a hereditary class, then MIS can be solved in FPT time in \mathcal{C} .

► **Definition 7.** An improving FPT g -Turing-reduction is an FPT time $h(k)|V(G)|^\gamma$ algorithm which, given an instance (I, k) , produces some instances $(I_1, k_1), \dots, (I_\ell, k_\ell)$, and can check a formula ϕ , such that:

- (i) $\text{yes}(I, k) \Leftrightarrow \phi(\text{yes}(I_1, k_1), \dots, \text{yes}(I_\ell, k_\ell))$, and
- (ii) $\exists c_0, f_0, \forall c \geq c_0, f \in \Omega(f_0), h(k)|V(G)|^\gamma + \sum_{j \in [\ell]} f(k_j)|I_j|^c \leq f(k)|I|^c$.

► **Lemma 8.** Assume there is an improving FPT g -Turing-reduction for MIS on every input $(I \in \mathcal{C}, k)$, producing in time $h(k)|I|^\gamma$, some instances $(I_1, k_1), \dots, (I_\ell, k_\ell)$. If each instance (I_j, k_j) can be solved in time $h(k_j)|I_j|^{c'}$, then MIS can be solved in FPT time in \mathcal{C} .

Proof. Let $c := \max(c_0, c')$ and $f := \max(f_0, h)$, for the c_0 and f_0 of Definition 8. *A fortiori*, instances (I_j, k_j) can be solved in time $f(k_j)|I_j|^c$. We call the Turing-reduction on (I, k) , solve every subinstances (I_j, k_j) , and check ϕ . By item (ii), the overall running time $h(k)|V(G)|^\gamma + \sum_{j \in [\ell]} f(k_j)|I_j|^c$ is bounded by $f(k)|I|^c$. By item (i), this decides (I, k) . ◀

When trying to compute MIS in FPT time, one can assume that there is no vertex of bounded degree or bounded co-degree (in terms of a function of k).

³ By *FPT-time checkable formula*, we mean that there exists an algorithm which takes as input ℓ Booleans b_1, \dots, b_ℓ and tests whether $\phi(b_1, \dots, b_\ell)$ is true in FPT time parameterized by ℓ .

► **Observation 9.** *Let (G, k) be an input of MIS with a vertex v of degree $g(k)$ for some computable function g . Then the instance admits a decreasing FPT Turing-reduction.*

Proof. A maximal independent set has to intersect $N[v]$. So, we can branch on $g(k) + 1$ instances with parameter $k - 1$. ◀

► **Observation 10.** *Let (G, k) be an input of MIS with a vertex v of co-degree $g(k)$ for some computable function g . Then the instance admits an improving FPT Turing-reduction.*

Proof. We can find the vertex v in time $ng(k)$ with $n := |V(G)|$, and we assume $n \geq 2$. By branching on v , we define two instances $(G - N[v], k - 1)$ and $(G - \{v\}, k)$ (which corresponds to including v to the solution, or not). The first instance can be further reduced in time $g(k)^{k-1}$ (by actually solving it). So the two instances output by the Turing-reduction are Bool and $(G - \{v\}, k)$, where Bool is the result of solving $(G - N[v], k - 1)$. The formula ϕ is just $\text{Bool} \vee \text{yes}(G - \{v\}, k)$. Let $c_0 := 2$ and $f_0(k) := g(k)^{k-1}$. For all $c \geq c_0$ and $f \in \Omega(f_0)$, $ng(k) + g(k)^{k-1} + f(k)(n - 1)^c \leq nf(k) + f(k) + f(k)(n - 1)^c \leq f(k)(n + 1 + (n - 1)^c) \leq f(k)n^c$. ◀

3 Almost disconnected and almost join graphs

We say that a graph is a join or a *complete sum*, if there is a non-trivial bipartition (A, B) of its vertex set (*i.e.* A and B are non-empty) such that every pair of vertices $(u, v) \in A \times B$ is linked by an edge. Equivalently, a graph is a complete sum if its complement is disconnected. In the following subsection, we define a series of variants of complete sums and disjoint unions in the presence of a *parameterized noise*.

3.1 Definition of the classes

In all the following definitions, we say that a tripartition (A, B, R) is *non-trivial* if A and B are non-empty and $|R| < \min(|A|, |B|)$. Notice that we do not assume R is non-empty.

► **Definition 11.** *Graphs in a class \mathcal{C} are (g, d) -almost disconnected if there exist two computable functions g and d , such that for every $G \in \mathcal{C}$ and $k \geq \alpha(G)$, there is a non-trivial tripartition (A, B, R) of $V(G)$ satisfying:*

- $|R| \leq g(k)$, and
- $\forall v \in A, |N(v) \cap B| \leq d(k)$ and $\forall v \in B, |N(v) \cap A| \leq d(k)$.

Similarly, we define a generalization of a complete sum.

► **Definition 12.** *Graphs in a class \mathcal{C} are (g, d) -almost bicomplete if there exist two computable functions g and d , such that for every $G \in \mathcal{C}$ and $k \geq \alpha(G)$, there is a non-trivial tripartition (A, B, R) of $V(G)$ satisfying:*

- $|R| \leq g(k)$, and
- $\forall v \in A, |B \setminus N(v)| \leq d(k)$ and $\forall v \in B, |A \setminus N(v)| \leq d(k)$.

By extension, if \mathcal{C} only contains graphs which are almost disconnected (resp. (g, d) -almost disconnected, almost bicomplete, (g, d) -almost bicomplete), then we say that \mathcal{C} is almost disconnected (resp. (g, d) -almost disconnected, almost bicomplete, (g, d) -almost bicomplete). Note that we do not require an almost disconnected or an almost bicomplete class to be hereditary. For $G \in \mathcal{C}$, we call a satisfying tripartition (A, B, R) a *witness of almost disconnectedness* (resp. *witness of almost bicompleteness*).

We define the one-sided variants.

► **Definition 13.** *Graphs in a class \mathcal{C} are one-sided (g, d) -almost disconnected if there exist two computable functions g and d , such that for every $G \in \mathcal{C}$ and $k \geq \alpha(G)$, there is a non-trivial tripartition (A, B, R) of $V(G)$ satisfying:*

- $|R| \leq g(k)$,
- $|B| > kd(k)$, and
- $\forall v \in A, |N(v) \cap B| \leq d(k)$.

In the above definition, the second condition is purely a technical one. Observe, though, that any tripartition (A, B, R) with $|R| < |B| \leq d(k)$ trivially satisfies the third condition (provided $|R| < d(k)$). So a condition forcing B to have more than $d(k)$ vertices is somehow needed. Now, we set the lower bound on $|B|$ a bit higher to make Lemma 18 work. Similarly, we could define the one-sided generalization of a complete sum.

► **Definition 14.** *Graphs in a class \mathcal{C} are one-sided (g, d) -almost bicomplete if there exist two computable functions g and d , such that for every $G \in \mathcal{C}$ and $k \geq \alpha(G)$, there is a non-trivial tripartition (A, B, R) of $V(G)$ satisfying:*

- $|R| \leq g(k)$,
- if there is an independent set of size k , there is one that intersects A , and
- $\forall v \in B, |A \setminus N(v)| \leq d(k)$.

Again, the second condition is there to make Theorem 20 work.

3.2 Improving and decreasing FPT Turing-reductions

The following technical lemma will be used to bound the running time of recursive calls on two almost disjoint parts of the input.

► **Lemma 15.** *Suppose $\gamma \geq 0$ and $c \geq \max(2, \gamma + 2)$ are two constants, and n_1, n_2, n, u are four positive integers such that $n_1 + n_2 + u = n$ and $\min(n_1, n_2) > u$. Then,*

$$n^\gamma + (n_1 + u)^c + (n_2 + u)^c < n^c.$$

Proof. First we observe that $n^2 - ((n_1 + u)^2 + (n_2 + u)^2) = n_1^2 + n_2^2 + u^2 + 2(n_1n_2 + n_1u + n_2u) - (n_1^2 + 2n_1u + u^2 + n_2^2 + 2n_2u + u^2) = 2n_1n_2 - u^2 > 2u^2 - u^2 = u^2 \geq 1$. Now, $n^c = n^{c-2}n^2 \geq n^{c-2}(1 + (n_1 + u)^2 + (n_2 + u)^2) \geq n^{c-2}(n^{\gamma-c+2} + (n_1 + u)^2 + (n_2 + u)^2) = n^\gamma + n^{c-2}(n_1 + u)^2 + n^{c-2}(n_2 + u)^2 > n^\gamma + (n_1 + u)^c + (n_2 + u)^c$. The last inequality holds since $n > n_1 + u$ and $n > n_2 + u$. ◀

We start with an improving FPT Turing-reduction on almost bicomplete graphs. It finds a kernel for solutions intersecting both A and B , solves recursively on $A \cup R$ and $B \cup R$ for the other solutions, and uses Lemma 15 to bound the overall running time.

► **Lemma 16.** *Let \mathcal{C} be a (g, d) -almost bicomplete class of graphs. Suppose for every $G \in \mathcal{C}$, a witness (A, B, R) of almost bicompleteness can be found in time $h(k)|V(G)|^\gamma$. Then, MIS admits an improving FPT Turing-reduction in \mathcal{C} . In particular, MIS can be solved in FPT time if both $(G[A \cup R], k)$ and $(G[B \cup R], k)$ can.*

Proof. We can detect a potential solution S intersecting both A and B in time $n^2(2d(k) + g(k))^k = n^2s(k)$, with $n := |V(G)|$, by setting $s(k) := (2d(k) + g(k))^{k-2}$. We exhaustively guess one vertex $a \in S \cap A$ and one vertex $b \in S \cap B$. For each of these quadratically many choices, there are at most $d(k)$ non-neighbors of a in B and at most $d(k)$ non-neighbors of b in A . So the remaining instance $G - (N(a) \cup N(b))$ has at most $2d(k) + g(k)$ vertices; hence the running time.

49:10 When Maximum Stable Set Can Be Solved in FPT Time

We are now left with potential solutions intersecting A but not B , or B but not A . These are fully contained in $A \cup R$ or in $B \cup R$. Let $n_1 := |A|$ and $n_2 := |B|$ (so $n = n_1 + n_2 + |R|$). The two last branches just consist of recursively solving the instances $(G[A \cup R], k)$ and $(G[B \cup R], k)$. Let $c_0 := \max(4, \gamma + 2)$ and $f_0 := h + s$. For all $c \geq c_0$ and $f \in \Omega(f_0)$,

$$\begin{aligned} & h(k)n^\gamma + s(k)n^2 + f(k)(n_1 + g(k))^c + f(k)(n_2 + g(k))^c \\ & \leq f(k)n^{\max(\gamma, 2)} + f(k)(n_1 + g(k))^c + f(k)(n_2 + g(k))^c \leq f(k)n^c. \end{aligned}$$

The last inequality holds by Lemma 15, since $\max(\gamma, 2) + 2 \leq c$ and $\min(n_1, n_2) > g(k)$. The conclusion holds by Lemma 8. \blacktriangleleft

If we only have *one-sided* almost bicompleteness, we need some additional conditions on the solution: at least one solution should intersect A (see Definition 14). We recall that $H \subseteq_i G$ means that H is a proper induced subgraph of G .

► **Lemma 17** (\star). *Let \mathcal{C} be a one-sided (g, d) -almost bicomplete class of graphs. Suppose for every $G \in \mathcal{C}$, a witness (A, B, R) of one-sided almost bicompleteness can be found in time $h(k)|V(G)|^\gamma$. Then, MIS admits an improving FPT Turing-reduction in \mathcal{C} . In particular, MIS can be solved in FPT time if $(G[A \cup R], k)$ and $\forall k' \leq k - 1, \forall H \subseteq_i G, (H, k')$ all can.*

We now turn our attention to almost disconnected classes. For these classes, we obtain decreasing FPT Turing-reductions, *i.e.*, where the produced instances have a strictly smaller parameter than the original instance.

► **Lemma 18** (\star). *Let \mathcal{C} be a one-sided (g, d) -almost disconnected class of graphs. Suppose for every $G \in \mathcal{C}$, a witness (A, B, R) of one-sided almost disconnectedness can be found in time $h(k)|V(G)|^\gamma$. Then, MIS admits a decreasing FPT Turing-reduction in \mathcal{C} . In particular, MIS can be solved in FPT time if $\forall k' \leq k - 1$ and $\forall H \subseteq_i G$, instances (H, k') can.*

Let $\mathcal{B}(A, B)$ be the bipartite graph between two disjoint vertex-subsets A and B (ignoring the edges internal to A and to B). We can further generalize the previous result to tripartitions (A, B, R) such that $\mathcal{B}(A, B)$ is $K_{d(k), d(k)}$ -free.

► **Definition 19**. *Graphs in a class \mathcal{C} are (g, d) -weakly connected if there exist two computable functions g and d , such that for every $G \in \mathcal{C}$ and $k \geq \alpha(G)$, there is a non-trivial tripartition (A, B, R) of $V(G)$ satisfying:*

- $|R| \leq g(k)$,
- $|A|, |B| > \lceil d(k)^{d(k)} k^{2d(k)-1} \rceil + 1$, and
- $\mathcal{B}(A, B)$ is $K_{d(k), d(k)}$ -free.

Again, if we do not require $|A|$ and $|B|$ to be larger than $d(k)$, such a tripartition may trivially exist. We force A and B to be even larger than that to make the next theorem work. We show this theorem by combining ideas of the proof of Lemma 18 with the extremal theory result, known as Kővári-Sós-Turán's theorem, that $K_{t,t}$ -free n -vertex graphs have at most $tn^{2-\frac{1}{t}}$ edges [25].

► **Theorem 20** (\star). *Let \mathcal{C} be a (g, d) -weakly connected class of graphs. Suppose for every $G \in \mathcal{C}$, a witness (A, B, R) of weakly connectedness can be found in time $h(k)|V(G)|^\gamma$. Then, MIS admits a decreasing FPT Turing-reduction in \mathcal{C} . In particular, MIS can be solved in FPT time if $\forall k' \leq k - 1$ and $\forall H \subseteq_i G$, the instance (H, k') can.*

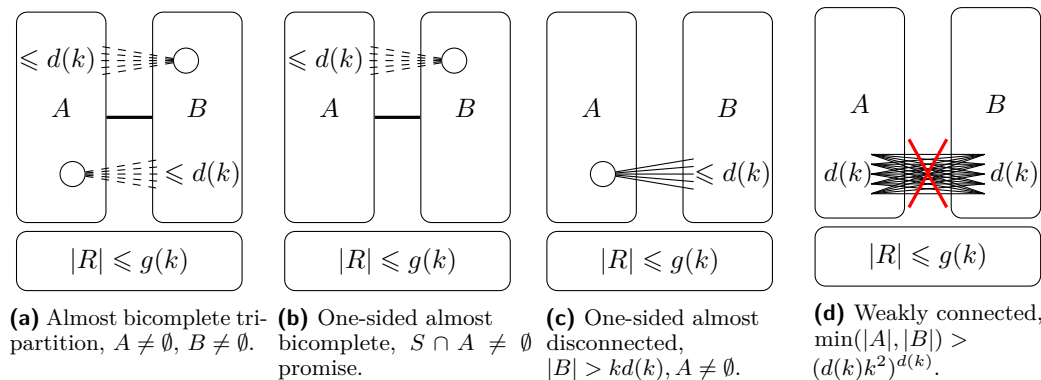
A class of *co-graphs with parameterized noise* is a hereditary class in which all the graphs are almost bicomplete or almost disconnected. The following is a direct consequence of the previous lemmas.

► **Corollary 21.** *Given an FPT oracle finding the corresponding tripartitions, MIS is FPT in co-graphs with parameterized noise.*

The corollary still holds by replacing *almost disconnected* by *one-sided almost disconnected*, or even by *weakly connected*.

3.3 Summary and usage

Figure 4 sums up the four FPT Turing-reductions that we obtained on almost disconnected and almost join graphs.



■ **Figure 4** Summary of the FPT Turing-reductions and their hypotheses, provided we can efficiently find such tripartitions. For (c) and (d), the FPT Turing-reductions are decreasing, while for (a) and (b) they are just improving.

We know provide a few words in order to understand how to use these results. An obvious caveat is that, even if such a tripartition exists, computing it (or even, approximating it) may not be fixed-parameter tractable. What we hope is that on a class \mathcal{C} , we will manage to exploit the class structure in order to eventually find such tripartitions, in the cases we cannot conclude by more direct means. One of our main results, Theorem 22, illustrates that mechanism, when the algorithm is centered around getting to the hypotheses of Lemma 17 or Theorem 20.

4 FPT algorithm in $P(1, t, t, t)$ -free graphs

We denote by $P(a, b, c, d)$ the graph made by substituting the vertices of P_4 by cliques of size $a, b, c,$ and $d,$ respectively. For instance, $P(1, 1, 1, 2)$ is \overline{P} and $P(1, 1, 2, 1)$ is the kite. We settle the parameterized complexity of MIS on \overline{P} -free and kite-free graphs simultaneously (see Figure 3), by showing that MIS is FPT even in the much wider class of $P(1, t, t, t)$ -free graphs.

► **Theorem 22.** *For every integer $t,$ MIS is FPT in $P(1, t, t, t)$ -free graphs.*

Proof. Let t be a fixed integer, and (G, k) be an input such that G is $P(1, t, t, t)$ -free and $\alpha(G) \leq k.$ We assume that $k \geq 3,$ otherwise we conclude in polynomial-time.

The global strategy is the following. First we extract a collection \mathcal{C} of disjoint and non-adjacent cliques with minimum and maximum size requirements, and some maximality condition. Then we partition the remaining vertices into equivalence classes with respect to their neighborhood in $\mathcal{C}.$ The maximum size imposed on the cliques of \mathcal{C} ensures that

the number of equivalence classes is bounded by a function of k . Setting C and the small⁴ equivalence classes apart, we show that the rest of the graph is partitionable into (A, B) such that either $\mathcal{B}(A, B)$ is $K_{d(k), d(k)}$ -free, in which case we conclude with Theorem 20, or $\mathcal{B}(A, B)$ is almost a complete bipartite graph, in which case we conclude with Lemma 17 (see Algorithm 2 in the appendix for the pseudo-code).

As for the running time, we are looking for an algorithm in time $f(k)n^c$ for some fixed constant $c \geq 2$, and f an increasing computable function. We see f as a partial function on $[k - 1]$, and extend it to $[k]$ in the recursive calls.

Building the clique collection C . For technical reasons, we want our collection C to contain at least two cliques, at least one of which being fairly large (larger than we can allow ourselves to brute-force). So we proceed in the following way. We find in polynomial-time $n^{8t+O(1)}$ a $2K_{4t}$. If G is $2K_{4t}$ -free, an FPT algorithm already exists [4]. We see these two K_{4t} as the two initial cliques of our collection. Let X be the set of vertices with less than t neighbors in at least one of these two K_{4t} . We partition X into at most 2^{8t} vertex-sets (later they will be called *subclasses*) with the same neighborhood on the $2K_{4t}$. If all these sets contain less than $Ram(k + 1, 2kt)$ vertices, X is fairly small: it contains less than $2^{8t} Ram(k + 1, 2kt)$. The other vertices have at least t neighbors in both K_{4t} . We will show (Lemma 24) that this implies that these vertices are completely adjacent to both K_{4t} . Hence, vertices in the $2K_{4t}$ would have at most $2^{8t} Ram(k + 1, 2kt)$ non-neighbors. In that case, we can safely remove the $2K_{4t}$ from G , by Observation 10.

So we can safely assume that (eventually) one subclass of X has more than $Ram(k + 1, 2kt)$ vertices. We can find in polynomial-time a clique C_2 of size $2kt$. We build a new collection with $3t$ vertices of the first K_{4t} , that we name C_1 . We take these vertices not adjacent to C_2 (this is possible since vertices in C_2 have the same at most $t - 1$ neighbors in K_{4t}). Now we have in C a clique C_1 of size $3t$ and a clique C_2 of size $2kt$.

We say that a clique of C is *large* if its size is above kt , and *small* otherwise. We can now specify the requirements on the collection C .

- (1) C is a vertex-disjoint and independent⁵ collection of cliques.
- (2) all the cliques have size at least $3t$ and at most $2kt$.
- (3) the number of cliques is at least 2.
- (4) if we find a way to strictly increase the number of large cliques in C , we do it.

As $\alpha(G) \leq k$, the number of cliques in C cannot exceed k . This has two positive consequences. The first is in conjunction with the way we improve the collection C : by always increasing the number of large cliques by 1. Therefore, we can improve the collection C at most $k - 1$ times. In particular, the improving process of C terminates (in polynomial time). The second benefit is that the total number of vertices of C is always bounded by $2k^2t$. Hence, the number of subclasses (sets of vertices with the exact same neighborhood in C) is bounded by a function of k (and the constant t).

As a slight abuse of notation, C_1, \dots, C_s will always be the current collection C ($s < k$). We say that a vertex of $G - C$ *t-sees* a clique C_j of C if it has at least t neighbors in C_j . A *class* is a set of vertices *t-seeing* the same set of cliques of C . A *subclass* is a set of vertices with the same neighborhood in C . Both classes and subclasses partition $G - C$. Observe that subclasses naturally refine classes. By extension, we say that a (sub)class *t-sees* a clique $C_i \in C$ if one vertex or equivalently all the vertices of that (sub)class *t-see* C_i .

⁴ the ones whose size is bounded by a later-specified function of k

⁵ There is no edge between two cliques of the collection.

Let $\eta := \lceil (2\text{Ram}(k+1, t))^{\text{Ram}(k+1, t)} 2^{2\text{Ram}(k+1, t)-1} \rceil + 1$. We choose this value so that $\eta^2/2 > \text{Ram}(k+1, t)(2\eta)^{2-1/\text{Ram}(k+1, t)}$ (it will become clear why in the proof of Lemma 27). We say that a subclass is *big* if it has more than $\max(\text{Ram}(k+1, 2kt), \eta) = \eta$ vertices, and *small* otherwise. Since $\alpha(G) \leq k$, here are two convenient properties on a big subclass:

- a clique of size t can be found in polynomial-time, in order to build a potential $P(1, t, t, t)$,
- a clique of size $2kt$ can be found, in order to challenge the maximality of C .

We will come back to the significance of η later.

We can now specify item (4) of the clique-collection requirements. We resume where we left off the collection C , that is $\{C_1 = K_{3t}, C_2 = K_{2kt}\}$. While there is a big subclass that does not t -see any large clique of C , we find a clique of size $2kt$ in that subclass, and add it to the collection. We then remove the small clique (K_{3t}) potentially left, and in each large clique of C , we remove from C all neighbors of the subclass (they are at most $t-1$ many of them). This process adds a large clique to C , and decreases the size of the previous large cliques by at most $t-1$. Since the large cliques all enter C with size $2kt$, and the number of improvements is smaller than k , a large clique will remain large throughout the entire process. Therefore, the number of large cliques in C increases by 1. Since we started with one large clique among the first two cliques, the number of cliques remains at least 2. Note that, at each iteration, we update the subclasses with respect to the new collection C (see Algorithm 1 for the pseudo-code).

■ **Algorithm 1** Routine for computing the clique collection C .

Precondition: k is a positive integer, G is not $2K_{4t}$ -free, $\alpha(G) \leq k$

```

1: function BUILDCLIQUECOLLECTION( $G, k$ ):
2:    $C \leftarrow \{K_{4t}, K_{4t}\}$  ▷ computed by brute-force
3:   if  $\exists$  big subclass not  $t$ -seeing both  $K_{4t}$  then
4:      $C_2 \leftarrow K_{2kt}$  in the subclass ▷ by Ramsey
5:      $C_1 \leftarrow 3t$  vertices not adjacent to  $C_2$  from one of the  $K_{4t}$  not  $t$ -seen by the subclass
6:      $C \leftarrow \{C_1, C_2\}$ 
7:   else every big subclasses  $t$ -see both  $K_{4t}$ 
8:     vertices in  $C$  have bounded co-degree ▷ Lemma 24
9:     we can safely delete them ▷ Observation 10
10:    and call BuildCliqueCollection( $G', k$ ) with the new graph  $G'$ 
11:   end if
12:   while  $\exists$  big subclass not  $t$ -seeing any large clique do
13:      $C_j \leftarrow K_{2kt}$  in the subclass ▷ by Ramsey
14:      $C' \leftarrow C \setminus \{\text{small clique}\}$  ▷ this is actually done at most once
15:      $C'' \leftarrow \text{map}(C', \text{deleteNeighborsOf}(C_j))$  ▷ remove  $C_i \cap N(C_j)$  from each  $C_i \in C$ 
16:      $C \leftarrow C'' \cup \{C_j\}$  ▷ the new  $C$  contains one more large clique,  $C_j$ .
17:   end while
18:   return  $C$ 
19: end function

```

Postcondition: output C is a collection of at least two (and at most $k-1$) pairwise independent cliques of size between $3t$ and $2tk$, and every big subclass t -sees at least one large clique (*i.e.*, clique of C of size at least tk).

Small subclasses are set aside as their size is bounded by a function of k . Therefore, from hereon, all the subclasses are supposed big. We denote by $P(I)$ the class for which $I \subseteq [s]$ represents the indices of the cliques it t -sees. A first remark is that all the subclasses of $P(\emptyset)$ are small (so we “get rid of” the whole class $P(\emptyset)$).

► **Lemma 23.** *If P' is a subclass of $P(\emptyset)$, then $|P'| \leq \text{Ram}(k+1, 2kt)$.*

Proof. P' does not t -see any (large) clique of C . So by the maximality property of C , it cannot contain a clique of size $2kt$ (see Algorithm 1). In particular, it cannot have more than $\text{Ram}(k+1, 2kt)$ vertices. ◀

We turn our attention to classes $P(I)$ with $|I| \geq 1$ and their subclasses.

Structure of the classes $P(I)$. We show a series of lemmas explaining how classes are connected to C and, more importantly, how they are connected to each other. This uses the ability to build cliques of size t at will, in big subclasses. Avoiding the formation of $P(1, t, t, t)$ will imply relatively dense or relatively sparse connections between classes $P(I)$ and $P(J)$.

► **Lemma 24.** *If a big subclass t -sees at least two cliques C_i and C_j of C , then all the vertices of that subclass are adjacent to all the vertices of both cliques.*

Proof. We find D , a clique of size t in the subclass. Let D_i and D_j be t neighbors of the subclass in C_i and C_j , respectively. Assume that the subclass has a non-neighbor $v \in C_i$. Then vD_iDD_j is a $P(1, t, t, t)$. ◀

In light of the previous lemma, if $|I| \geq 2$, the cliques of C that the class $P(I)$ t -sees are completely adjacent to $P(I)$.

► **Lemma 25.** *Let $I \subsetneq J \subseteq [s]$. Then, every vertex of $P(I)$ is adjacent to every vertex of $P(J)$ except at most $\text{Ram}(k+1, t)$.*

Proof. Let $i \in I$ and $j \in J \setminus I$. By Lemma 24, all vertices of $P(J)$ are adjacent to all vertices of $C_i \cup C_j$. Suppose, by contradiction, that there is a vertex $u \in P(I)$ with more than $\text{Ram}(k+1, t)$ non-neighbors in $P(J)$. We find a clique D of size t in $G[P(J) \setminus N(u)]$. Let D_i be t neighbors of u in C_i . Let $D_j \subset C_j$ be t neighbors of $P(J)$ which are not neighbors of u . Such a set D_j necessarily exists since u has at most $t-1$ neighbors in C_j , while $P(J)$ is completely adjacent to C_j , and $|C_j| \geq 3t$. Then uD_iDD_j is a $P(1, t, t, t)$. ◀

We say that two sets I, J are *incomparable* if I is not included in J , and J is not included in I . Recall that $\mathcal{B}(A, B)$ stands for the bipartite graph between vertex-set A and vertex-set B . Let $p(t, k) := 2^{2k^2t}$ be a crude upper bound on the total number of subclasses.

► **Lemma 26.** *Let $I, J \subseteq [s]$ be two incomparable sets, and $P_\ell(I), P_{\ell'}(J)$ be any pair of subclasses of $P(I)$ and $P(J)$, respectively. Then, $\mathcal{B}(P_\ell(I), P_{\ell'}(J))$ is $K_{\text{Ram}(k+1, t), \text{Ram}(k+1, t)}$ -free. Hence, $\mathcal{B}(P(I), P(J))$ is $K_{p(t, k)\text{Ram}(k+1, t), p(t, k)\text{Ram}(k+1, t)}$ -free.*

Proof. Let $i \in I \setminus J$ and $j \in J \setminus I$. We first assume that one of I, J , say I , has at least two elements. Suppose, by contradiction, that there is a set $B_I \subseteq P_\ell(I)$ and a set $B_J \subseteq P_{\ell'}(J)$ both of size $\text{Ram}(k+1, t)$, such that there is no non-edge between B_I and B_J . Let u be a vertex of C_j which is adjacent to $P_{\ell'}(J)$ but not to $P_\ell(I)$. We find D_I , a clique of size t in $G[B_I]$, and D_J , a clique of size t in $G[B_J]$. Let D_i be t neighbors of $P_\ell(I)$ in C_i that are not adjacent to $P_{\ell'}(J)$. Those t vertices exist since, by Lemma 24, $P_\ell(I)$ is completely adjacent to C_i (by assumption $|I| \geq 2$). And $P_{\ell'}(J)$ has more than t non-neighbors in C_i . Then, $uD_JD_ID_i$ is a $P(1, t, t, t)$.

We now have to settle the remaining case: $|I| = |J| = 1$ ($I = \{i\}$ and $J = \{j\}$). If $P_\ell(I)$ has at least $2t$ neighbors in C_i or $P_{\ell'}(J)$ has at least $2t$ neighbors in C_j , we conclude as in the previous paragraph. So we assume that it is not the case. We distinguish two cases.

Either $P_\ell(I)$ has at least one neighbor in C_j , say u . Let D_I be a clique of size t in $P_\ell(I)$, $D_i \subseteq C_i$ be t neighbors of $P_\ell(I)$, and $D'_i \subseteq C_i$ be t non-neighbors of $P_\ell(I)$. D_i and D'_i exist since $P_\ell(I)$ has between t and $2t - 1$ neighbors in C_j , and $|C_j| \geq 3t$. Then, $uD_I D_i D'_i$ is a $P(1, t, t, t)$.

Or $P_\ell(I)$ has no neighbor in C_j . Let u be a non-neighbor of $P_{\ell'}(J)$ in C_j , and $D_j \subseteq C_j$ be t neighbors of $P_{\ell'}(J)$. If there is a set $B_I \subseteq P_\ell(I)$ and a set $B_J \subseteq P_{\ell'}(J)$ both of size $Ram(k + 1, t)$, such that B_I and B_J are completely adjacent to each other. We can find D_I , a clique of size t in $G[B_I]$, and D_J , a clique of size t in $G[B_J]$. Then, $uD_J D_I$ is a $P(1, t, t, t)$. This implies that, in any case, there cannot be a $K_{p(t,k)Ram(k+1,t), p(t,k)Ram(k+1,t)}$ in $\mathcal{B}(P(I), P(J))$. ◀

We say that the sets I and J *overlap* if all three of $I \cap J$, $I \setminus J$, $J \setminus I$ are non-empty.

► **Lemma 27.** *Let $I, J \subseteq [s]$ be two overlapping sets. Then, at least one of $P(I)$ and $P(J)$ have only small subclasses.*

Proof. Suppose, by contradiction, that there is a big subclass $P_\ell(I)$ of $P(I)$, and a big subclass $P_{\ell'}(J)$ of $P(J)$. Observe that, for I and J to overlap, their size should be at least 2. Let $i \in I \setminus J$, $j \in J \setminus I$, $h \in I \cap J$. By the arguments of Lemma 25 applied to the restriction to $P(I)$, $P(J)$, C_h , and C_j , a vertex in $P(I)$ has at most $Ram(k + 1, t)$ non-neighbors in $P(J)$. Let us consider η vertices in $P_\ell(I)$ and η vertices in $P_{\ell'}(J)$. Since $\eta \geq 2Ram(k + 1, t)$, the previous observation implies that the number of edges between them is at least $\eta^2/2$. But by Lemma 26, the bipartite graph linking them should be $K_{Ram(k+1,t), Ram(k+1,t)}$ -free. By Kővári-Sós-Turán's theorem, this number of edges is bounded from above by $Ram(k + 1, t)(2\eta)^{2-1/Ram(k+1,t)} < \eta^2/2$, a contradiction. ◀

Hence, the remaining (not entirely made of small subclasses) classes define a laminar⁶ set-system. We denote by R the union of the vertices in all the small subclasses and C . We now add a new condition to be a small subclass (condition that we did not need thus far). A subclass is also small if it has at most $|R|$ vertices. Note that this condition can snowball. But eventually R has size bounded by $g(k) := 2^{p(t,k)}(p(t,k)\eta + 2k^2t)$. A class is *remaining* if it contains at least one big subclass. By Lemma 23, $P(\emptyset)$ cannot be remaining. If no class is remaining, then the whole graph is a kernel. So we can assume that there is at least one remaining class. Let $P(I)$ be a remaining class in $G - R$ such that I is maximal among the remaining classes. We distinguish two cases: either there is at least one other remaining class $P(J)$ ($I \neq J$), or $P(I)$ is the unique remaining class.

At least two remaining classes $P(I)$ and $P(J)$. By Lemma 27, any other class $P(J)$ satisfies $J \subsetneq I$ or $I \cap J = \emptyset$. Let $\iota, \delta \leq 2^k$ be the number of remaining classes such that $J \subsetneq I$ and such that $I \cap J = \emptyset$, respectively. Again, we distinguish two cases: $\delta > 0$, and $\delta = 0$. If $\delta > 0$, we build the partition (A, B, R) of $V(G)$ such that A contains the $\iota + 1$ classes whose set is included in I and B contains the δ classes whose set is disjoint from I . By Lemma 26, the bipartite graph between any of the $(\iota + 1)\delta$ pairs of classes made of one class whose set is contained in I and one class whose set is disjoint from I is $K_{p(t,k)Ram(k+1,t), p(t,k)Ram(k+1,t)}$ -free. Hence, the bipartite graph between A and B is $K_{2^k p(t,k)Ram(k+1,t), 2^k p(t,k)Ram(k+1,t)}$ -free. Thus we conclude by Theorem 20 with $d(k) = 2^k p(t, k)Ram(k + 1, t)$.

⁶ where two sets are nested or disjoint

We now tackle the case $\delta = 0$, that is, all the remaining classes $P(J)$ satisfy $J \subseteq I$. We first assume that there are two remaining classes with disjoint sets. A laminar set-system with a unique maximal set can be represented as a rooted tree, where nodes are in one-to-one correspondence with the sets, and the parent-to-child arrow represents the partial order of inclusion. Here, the root is labeled by I (since I is the unique maximal set), and all the nodes are labeled by a subset of $[s]$ corresponding to a remaining class. Let $I = I_1 \supseteq I_2 \supseteq \dots \supseteq I_h$ be the path from the root to the first node with out-degree at least 2. Observe that C contains at most k cliques, so $h \leq k$. Let J_1, J_2, \dots, J_ℓ be the ℓ children of I_h (with $\ell \geq 2$). Let \mathcal{P}_1 be the remaining classes whose set is included in J_1 , and \mathcal{P}_{2+} be the remaining classes whose set is included in one J_i for some $i \in [2, \ell]$. Let $A := \bigcup_{q \in [h]} P(I_q)$, and $B := V(G) \setminus (A \cup R)$. By Lemma 25, vertices of B have at most $h \text{Ram}(k+1, t) \leq k \text{Ram}(k+1, t)$ non-neighbors in A . We apply Lemma 17 with the tripartition (A, B, R) and $d_1(k) = k \text{Ram}(k+1, t)$. Only we did not cover the case in which the solution does not intersect A . We do so by applying Theorem 20 to the tripartition $(\mathcal{P}_1, \mathcal{P}_{2+}, R)$ with $d_2(k) = 2^k p(t, k) \text{Ram}(k+1, t)$. A priori, what we just did is not bounded by $f(k)|V(G)|^c$, hence not legal. Let us go back to the last lines of Lemma 17 and of Theorem 20. Our running time is bounded by $f(k)|A \cup R|^c + k^2 \binom{d_1(k)}{k} d_1(k)^c f(k-1)|B|^c + k(k+2)(\lceil d_2(k)^{d_2(k)} k^{2d_2(k)-1} \rceil + 1)f(k-1)|B|^c$, where the two first terms come from the application of Lemma 17, and the third term, from Theorem 20. This is at most $f(k)|A \cup R|^c + f(k)|B|^c \leq f(k)|V(G)|^c$ by Cauchy-Schwarz inequality, with $f(k) := (k^2 \binom{d_1(k)}{k} d_1(k)^c + k(k+2)(\lceil d_2(k)^{d_2(k)} k^{2d_2(k)-1} \rceil + 1))f(k-1)$.

Let now assume that all the remaining classes have nested sets (no two sets are disjoint). Let $I = I_1 \supseteq I_2 \supseteq \dots \supseteq I_h$ the sets of *all* the remaining classes ($h \leq k$). Suppose $h \geq 3$. We apply Lemma 17 to the tripartition $(P(I_1) \cup P(I_2), \bigcup_{j \in [3, h]} P(I_j), R)$ with $d(k) = 2 \text{Ram}(k+1, t)$. Indeed, by Lemma 25, vertices of $\bigcup_{j \in [3, h]} P(I_j)$ have at most $\text{Ram}(k+1, t)$ non-neighbors in $P(I_1)$ and at most $\text{Ram}(k+1, t)$ non-neighbors in $P(I_2)$. We deal with the case in which the solution does not intersect $P(I_1) \cup P(I_2)$ in the following way. Let C_q be the clique of C only t -seen by $P(I_1)$ and $C_{q'}$ the clique of C only t -seen by $P(I_1) \cup P(I_2)$. One of these two cliques has to be large (since there is at most one small clique). We branch on the at least tk and at most $2tk$ vertices of that large clique, say C' . A maximal independent set cannot be fully contained in $\bigcup_{j \in [3, h]} P(I_j)$. Indeed, any choice of at most k vertices in this set dominates at most $k(t-1)$ vertices of C' . Thus, we cannot miss a solution. Let us turn to the running time. Once again, we cannot use Lemma 17 as a total black-box. Our running time is bounded by $f(k)|A \cup R|^c + k^2 \binom{d(k)}{k} d(k)^c f(k-1)|B|^c + 2tkf(k-1)|B \cup R|^c \leq f(k)|A \cup R|^c + f(k)|B \cup R|^c$ with $f(k) := (k^2 \binom{d(k)}{k} d(k)^c + 2tkf) f(k-1)$, and $f(k)|A \cup R|^c + f(k)|B \cup R|^c \leq f(k)|V(G)|^c$, by Lemma 15. Here we need that $|A| > |R|$ and $|B| > |R|$ which is the case: recall that we added that requirement to be a big subclass.

The last case is the following. There are exactly two remaining classes associated to sets $I = I_1 \supseteq I_2$. If a clique not t -seen by $P(I_2)$ is large or if $P(I_2)$ is $2K_{4t}$ -free, we conclude with Lemma 17 (recall that this finds a solution if there is one intersecting $P(I_1)$). In both cases, if the solution does not intersect $P(I_1)$, we can find it with only a small overhead cost. If a clique not t -seen by $P(I_2)$ is large, we branch on the at most $2kt$ vertices of that clique. If $P(I_2)$ is $2K_{4t}$ -free, an independent set of size k can be found in $G[P(I_2)]$ in FPT time [4].

Finally, we can assume that $G[P(I_2)]$ contains a $2K_{4t, 4t}$ and does not t -see a small clique in C . Note that this implies that C is made of two cliques K_{3t} and K_{2kt} . We call *critical* such a case where $C = \{K_{3t}, K_{2kt}\}$ and a $2K_{4t}$ can be found in a class not t -seeing K_{3t} .

For this very specific case (that may also arise with a unique remaining class, see below), we perform the following refinement of the clique-collection computation. We compute a new clique collection, say C^2 , in $G - C$, starting with a $2K_{4t, 4t}$ found in the class not t -seeing

the previous K_{3t} . If C^2 is not of the form $\{K_{3t}, K_{2kt}\}$, we add C to the bounded-in- k set R , and we follow our algorithm (that is, a non-critical case). If $C^2 = \{K_{3t}, K_{2kt}\}$, we compute a new clique collection C^3 in $G - (C^1 \cup C^2)$ (with $C^1 = C$), again starting with a $2K_{4t,4t}$ found in the class not t -seeing the previous K_{3t} , and so on. Let us assume that we are always in a critical case, with $C^h = \{C_1^h = K_{3t}, C_2^h = K_{2kt}\}$. We stop after $\zeta := \text{Ram}_{2(3t)^2}(4kt)$ iterations, leading to disjoint (though not independent) clique collections $C = C^1, C^2, \dots, C^\zeta$. In particular, $|\bigcup_{h \in \zeta} C^h|$ is still bounded by a function of k , namely $\zeta(3t + 2kt)$. We claim that we can find a $2K_{2kt,2kt}$ in $G[\bigcup_{h \in \zeta} C_1^h]$.

Because of the number of iterations, one can extract $4kt$ cliques C_1^h (of size $3t$) with the same bipartite graph linking any pair of C_1^h (with a fixed but arbitrary ordering of each C_1^h). This common bipartite graph has to be empty, complete, or a half-graph. Let us show that it can only be a half-graph. For any $i \in [3t]$, the i -th vertices in the C_1^h should be adjacent (otherwise they form an independent set of size $2kt$). That excludes the empty bipartite graph. Let h_1 be the smallest index such that we have extracted $C_1^{h_1}$. The common bipartite graph cannot be complete either, since all the vertices of $G - (\bigcup_{h \in [h_1]} C_1^h)$ have at most $t - 1$ neighbors in $C_1^{h_1}$. This was one of the condition of a critical case. So the bipartite graph is a half-graph. Then we find our $2K_{2kt,2kt}$ as the first vertex (or last vertex) of the first $2kt$ extracted cliques, and the last vertex (or first vertex) of the last $2kt$ extracted cliques. Now we finally have a clique collection with two independent *large* cliques, depending on the orientation of the half-graph. So we can start again without reaching the problematic case.

Unique remaining $P(I)$. If $|I| \geq 2$, by Lemma 24, $P(I)$ is completely adjacent to one clique C_i (with $i \in I$). Any vertex of C_i has at most $g(k)$ non-neighbors. This case is handled by Observation 10. So we now suppose that $|I| = 1$ (and $I = \{i\}$). If $P(I)$ does not t -see a large clique C_j , we can branch on the at most $2kt$ vertices of that clique. Indeed, there is a solution that intersects it, since $k - 1$ vertices in $G - R$ can dominate at most $(k - 1)(t - 1) < kt$ vertices. Thus, we can further assume that $P(I)$ t -sees all the large cliques. This forces that there is at most one large clique, since $|I| = 1$. There cannot be at least three cliques in C . Indeed, the way the collection is maintained, that would imply that there are at least two large cliques. So, $C = \{C_1 = K_{3t}, C_2 = K_{2kt}\}$ and $I = \{2\}$. This is a *critical* case, which we handle as in the previous paragraph (with two remaining classes). ◀

5 Randomized FPT algorithms in dart-free and cricket-free graphs

In this section, we consider the case of dart-free and cricket-free graphs, and prove that there is a randomized FPT algorithm for MIS in both graph classes. To this end, we use the technique of iterative expansion together with a Ramsey extraction, as well as the results developed in Section 3. The proofs can be found in the long version of the paper [6].

► **Theorem 28 (♠).** *There is a randomized FPT algorithm for MIS in dart-free graphs.*

► **Theorem 29 (♠).** *There is a randomized FPT algorithm for MIS in cricket-free graphs.*

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(1-3):3–16, 2004. doi:10.1016/S0166-218X(02)00290-1.

- 3 Gábor Bacsó, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Zsolt Tuza, and Erik Jan van Leeuwen. Subexponential-Time Algorithms for Maximum Independent Set in P_t -Free and Broom-Free Graphs. *Algorithmica*, 81(2):421–438, 2019. doi:10.1007/s00453-018-0479-5.
- 4 Édouard Bonnet, Nicolas Bousquet, Pierre Charbit, Stéphan Thomassé, and Rémi Watrigant. Parameterized Complexity of Independent Set in H-Free Graphs. In *13th International Symposium on Parameterized and Exact Computation, IPEC 2018, August 20-24, 2018, Helsinki, Finland*, pages 17:1–17:13, 2018. doi:10.4230/LIPIcs.IPEC.2018.17.
- 5 Édouard Bonnet, Nicolas Bousquet, Pierre Charbit, Stéphan Thomassé, and Rémi Watrigant. Parameterized Complexity of Independent Set in H-Free Graphs. *CoRR*, abs/1810.04620, 2018. arXiv:1810.04620.
- 6 Édouard Bonnet, Nicolas Bousquet, Stéphan Thomassé, and Rémi Watrigant. When Maximum Stable Set can be solved in FPT time. *CoRR*, abs/1909.08426, 2019. arXiv:1909.08426.
- 7 Andreas Brandstädt and Raffaele Mosca. Maximum weight independent set for ℓ claw-free graphs in polynomial time. *Discrete Applied Mathematics*, 237:57–64, 2018. doi:10.1016/j.dam.2017.11.029.
- 8 Andreas Brandstädt and Raffaele Mosca. Maximum Weight Independent Sets for $(P_7, \text{triangle})$ -free graphs in polynomial time. *Discrete Applied Mathematics*, 236:57–65, 2018. doi:10.1016/j.dam.2017.10.003.
- 9 Andreas Brandstädt and Raffaele Mosca. Maximum Weight Independent Sets for $(S_{1,2,4}, \text{Triangle})$ -Free Graphs in Polynomial Time. *CoRR*, abs/1806.09472, 2018. arXiv:1806.09472.
- 10 Jianer Chen, Yang Liu, Songjian Lu, Sing-Hoi Sze, and Fenghui Zhang. Iterative Expansion and Color Coding: An Improved Algorithm for 3D-Matching. *ACM Trans. Algorithms*, 8(1):6:1–6:22, 2012.
- 11 Derek G. Corneil, Yehoshua Perl, and Lorna K. Stewart. A Linear Recognition Algorithm for Cographs. *SIAM J. Comput.*, 14(4):926–934, 1985.
- 12 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 13 Konrad Dabrowski. *Structural Solutions to Maximum Independent Set and Related Problems*. PhD thesis, University of Warwick, 2012.
- 14 Konrad Dabrowski, Vadim V. Lozin, Haiko Müller, and Dieter Rautenbach. Parameterized complexity of the weighted independent set problem beyond graphs of bounded clique number. *J. Discrete Algorithms*, 14:207–213, 2012.
- 15 Konrad K. Dabrowski, Vadim V. Lozin, Dominique de Werra, and Victor Zamaraev. Combinatorics and Algorithms for Augmenting Graphs. *Graphs and Combinatorics*, 32(4):1339–1352, 2016. doi:10.1007/s00373-015-1660-0.
- 16 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.
- 17 Rod G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.
- 18 Henri Perret du Cray and Ignasi Sau. Improved FPT algorithms for weighted independent set in bull-free graphs. *Discrete Mathematics*, 341(2):451–462, 2018.
- 19 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- 20 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding First-Order Properties of Nowhere Dense Graphs. *J. ACM*, 64(3):17:1–17:32, 2017. doi:10.1145/3051095.
- 21 Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981. doi:10.1007/BF02579273.
- 22 Andrzej Grzesik, Tereza Klimosova, Marcin Pilipczuk, and Michał Pilipczuk. Polynomial-time algorithm for Maximum Weight Independent Set on P_6 -free graphs. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1257–1271, 2019. doi:10.1137/1.9781611975482.77.

- 23 Ararat Harutyunyan, Michael Lampis, Vadim V. Lozin, and Jérôme Monnot. Maximum Independent Sets in Subcubic Graphs: New Results. In *Graph-Theoretic Concepts in Computer Science - 45th International Workshop, WG 2019, Vall de Núria, Spain, June 19-21, 2019, Revised Papers*, pages 40–52, 2019. doi:10.1007/978-3-030-30786-8_4.
- 24 Johan Håstad. Clique is Hard to Approximate Within $n^{1-\epsilon}$. In *37th Annual Symposium on Foundations of Computer Science, FOCS '96, Burlington, Vermont, USA, 14-16 October, 1996*, pages 627–636, 1996. doi:10.1109/SFCS.1996.548522.
- 25 Tamás Kovári, Vera Sós, and Pál Turán. On a problem of K. Zarankiewicz. In *Colloquium Mathematicum*, volume 1, pages 50–57, 1954.
- 26 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the Exponential Time Hypothesis. *Bulletin of the EATCS*, 105:41–72, 2011. URL: <http://eatcs.org/beatcs/index.php/beatcs/article/view/92>.
- 27 Daniel Lokshtanov, Marcin Pilipczuk, and Erik Jan van Leeuwen. Independence and Efficient Domination on P_6 -free Graphs. *ACM Trans. Algorithms*, 14(1):3:1–3:30, 2018. doi:10.1145/3147214.
- 28 Daniel Lokshtanov, Martin Vatshelle, and Yngve Villanger. Independent Set in P_5 -Free Graphs in Polynomial Time. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014*, pages 570–581, 2014.
- 29 Vadim V. Lozin. From matchings to independent sets. *Discrete Applied Mathematics*, 231:4–14, 2017. doi:10.1016/j.dam.2016.04.012.
- 30 Vadim V. Lozin and Martin Milanič. Maximum independent sets in graphs of low degree. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 874–880, 2007. URL: <http://dl.acm.org/citation.cfm?id=1283383.1283477>.
- 31 Vadim V. Lozin and Martin Milanič. A polynomial algorithm to find an independent set of maximum weight in a fork-free graph. *J. Discrete Algorithms*, 6(4):595–604, 2008. doi:10.1016/j.jda.2008.04.001.
- 32 Vadim V. Lozin, Jérôme Monnot, and Bernard Ries. On the maximum independent set problem in subclasses of subcubic graphs. *J. Discrete Algorithms*, 31:104–112, 2015. doi:10.1016/j.jda.2014.08.005.
- 33 Frédéric Maffray and Lucas Pastor. Maximum weight stable set in (P_7, bull) -free graphs and $(S_{1,2,3}, \text{bull})$ -free graphs. *Discrete Mathematics*, 341(5):1449–1458, 2018. doi:10.1016/j.disc.2017.10.004.
- 34 Dmitry S. Malyshev. Classes of subcubic planar graphs for which the independent set problem is polynomially solvable. *Journal of Applied and Industrial Mathematics*, 7(4):537, 2013.
- 35 Dmitry S. Malyshev and Dmitrii V. Sirotkin. Polynomial-time solvability of the independent set problem in a certain class of subcubic planar graphs. *Journal of Applied and Industrial Mathematics*, 11(3):400–414, 2017.
- 36 George J. Minty. On maximal independent sets of vertices in claw-free graphs. *J. Comb. Theory, Ser. B*, 28(3):284–304, 1980. doi:10.1016/0095-8956(80)90074-X.
- 37 Najiba Sbihi. Algorithme de recherche d’un stable de cardinalité maximum dans un graphe sans étoile. *Discrete Mathematics*, 29(1):53–76, 1980. doi:10.1016/0012-365X(90)90287-R.
- 38 Stéphan Thomassé, Nicolas Trotignon, and Kristina Vuskovic. A Polynomial Turing-Kernel for Weighted Independent Set in Bull-Free Graphs. *Algorithmica*, 77(3):619–641, 2017.
- 39 David Zuckerman. Linear Degree Extractors and the Inapproximability of Max Clique and Chromatic Number. *Theory of Computing*, 3(1):103–128, 2007.

A Proof of Lemma 17

Proof. Let S be an unknown solution. Let $k_1 := S \cap A$ and $k_2 := S \cap B$. Let us anticipate on an FPT running time $f(k)n^c$ for instances of size n and parameter k (the definition of f will be given later). For instance, covering the case $k_2 = 0$ takes time $f(k)|A \cup R|^c$, since it consists in solving $(G[A \cup R], k)$. By assumption, we do not have to consider the case $k_1 = 0$. For each pair k_1, k_2 such that $k_1 \geq 1, k_2 \geq 1, k_1 + k_2 \leq k$, we do the following.

An independent set of size k_1 in $G[A]$ is *candidate* if it is in the non-neighborhood of at least one vertex $v \in B$. Since $k_2 \geq 1$, we can restrict the search in A to candidate independent sets of size k_1 . Indeed, any independent set in A , not in the non-neighborhood of any vertex of B , cannot be extended to k_2 (≥ 1) more vertices of B . For each candidate independent set I_1 of size k_1 , we compute an independent set of size k_2 in $B \setminus N(I_1)$. This takes time

$$\sum_{\substack{I_1 \text{ candidate} \\ |I_1|=k_1}} f(k_2)|B \setminus N(I_1)|^c = f(k_2) \sum_{\substack{I_1 \text{ candidate} \\ |I_1|=k_1}} |B \setminus N(I_1)|^c \leq f(k_2) \left(\sum_{\substack{I_1 \text{ candidate} \\ |I_1|=k_1}} |B \setminus N(I_1)| \right)^c$$

by Cauchy-Schwarz inequality (since $c \geq 2$). Now, since $k_1 > 0$,

$$\sum_{\substack{I_1 \text{ candidate} \\ |I_1|=k_1}} |B \setminus N(I_1)| \leq \sum_{\substack{I_1 \text{ candidate} \\ |I_1|=k_1}} |I_1| \cdot |B \setminus N(I_1)| \leq \binom{d(k)}{k_1} d(k) |B|.$$

The last inequality holds since $\sum_{I_1 \text{ candidate}, |I_1|=k_1} |I_1| \cdot |B \setminus N(I_1)|$ counts the number of non-edges between A and B with multiplicity at most $\binom{d(k)}{k_1}$. Indeed a same non-edge uv (with $u \in A$, $v \in B$) is counted for at most $\binom{d(k)}{k_1}$ candidate independent sets (since they have to be in the non-neighborhood of v). Since, by assumption, vertices in B have at most $d(k)$ non-neighbors in A , the total number of non-edges is $d(k)|B|$. Let $c_0 \geq \gamma + 2$ and $f_0 := \max(h, k \mapsto k^{2k} \binom{d(k)}{k}^{ck} d(k)^{ck})$. For any $c \geq c_0$ and $f \in \Omega(f_0)$,

$$\begin{aligned} & h(k)|V(G)|^\gamma + f(k)|A \cup R|^c + \sum_{k_1 \in [k-1], k_2 \in [k-k_1]} f(k_2) \left(\sum_{\substack{I_1 \text{ candidate} \\ |I_1|=k_1}} |B \setminus N(I_1)| \right)^c \\ & \leq h(k)|V(G)|^\gamma + f(k)|A \cup R|^c + k^2 f(k-1) \binom{d(k)}{k}^c d(k)^c |B|^c \\ & \leq f(k)|V(G)|^\gamma + f(k)|A \cup R|^c + f(k)|B|^c \leq f(k)|V(G)|^c \end{aligned}$$

since $f(k) \geq k^2 \binom{d(k)}{k}^c d(k)^c f(k-1)$. The last inequality holds by Lemma 15. The conclusion holds by Lemma 8. \blacktriangleleft

B Proof of Lemma 18

Proof. Let S be an unknown but supposed independent set of G of size k . In time $h(k)n^c$ with $n := |V(G)|$, we compute a witness (A, B, R) . For each $u \in R$, we branch on including u to our solution. This represents at most $g(k)$ branches with parameter $k-1$. Now, we can focus on the case $S \cap R = \emptyset$.

We first deal separately with the special cases of $|S \cap A| = k$, $|S \cap B| = 0$ (a), and of $|S \cap A| = 0$, $|S \cap B| = k$ (b). As by assumption $|B| > kd(k)$, no maximal independent set has k vertices in A and zero in B . Indeed, by the one-sided almost disconnectedness, any k vertices in A dominate at most k^2 vertices in B . Hence at least one vertex of B could be added to this independent set of size k . So case (a) is actually impossible.

For case (b), we proceed as follows. We compute an independent set of size $k-1$ in $G[B]$. We temporarily remove it from the graph, without removing its neighborhood. We compute a second independent set of size $k-1$ in $G[B]$ (without the first independent set); then a third one (in the graph deprived of the first two). We iterate this process until no

independent set of size $k - 1$ is found or we reach a total of $d(k) + 1$ (disjoint) independent sets of size $k - 1$ excavated in B . If we stop because of the former alternative, we know that an independent set of size k (actually even of size $k - 1$) in B has to intersect the union of at most $d(k)$ independent sets of size $k - 1$; so at most $(k - 1)d(k)$ vertices in total. In that case, we branch on each vertex of this set of size at most $(k - 1)d(k)$ with parameter $k - 1$. If we stop because of the latter condition, we can include an arbitrary vertex w of A in the solution. By assumption, w has at least one neighbor in at most $d(k)$ independent sets of size $k - 1$ in B . So at least one independent set of size $k - 1$ of the collection is not adjacent to w , and forms with w a solution.

Now we are done with cases (a) and (b), we can assume that $k_1 := |S \cap A|$, $k_2 := |S \cap B| = k - k_1$ are both non-zero. Equivalently, $1 \leq k_1 \leq k - 1$. We try out all the $k - 1$ possibilities. For each, we perform a similar trick to the one used for case (b). We compute an independent set I_1 of size k_2 in $G[B]$. We then compute an independent set I_2 of size k_2 in $G[B \setminus I_1]$. Observe that there may be edges between I_1 and I_2 . We compute an independent set I_3 in $G[B \setminus (I_1 \cup I_2)]$, and so on. We iterate this process until no independent set of size k_2 is found or we reach a total of $d(k)k_1 + 1$ (disjoint) independent sets of size k_2 excavated in B .

Say, we end up with the sets I_1, \dots, I_s . Let $I := \bigcup_{j \in [s]} I_j$. If $s \leq f(k)k_1$, then we stopped because there was no independent set of size k_2 in $G[B \setminus I]$. This means that S intersects I . In that case, we branch on each vertex of I .

The other case is that $s = f(k)k_1 + 1$ and we stopped because we had enough sets I_j . In that case, we compute one independent set A_1 of size k_1 in $G[A]$. By assumption, $|N_B(A_1)| \leq k_1 d(k)$. In particular, there is at least one I_j which does not intersect $N_B(A_1)$. And $A_1 \cup I_j$ is our independent of size k .

Our algorithm makes at most

$$g(k) + d(k) + 1 + \sum_{k_1 \in [k-1]} (d(k)k_1 + 1) + 1 \leq g(k) + d(k) + 2 + k^2 d(k) + k$$

recursive calls to instances with parameter $k - 1$, and we conclude by Lemma 5. ◀

C Proof of Theorem 20

Proof. Let S be an unknown solution with $k_1 := |S \cap A|$ and $k_2 := |S \cap B| = k - k_1$. As previously, we try out all the $k + 1$ values for k_1 , setting k_2 to $k - k_1$. Let us first consider the $k - 1$ branches in which $k_1 \neq 0$ and $k_2 \neq 0$.

Let $s := \lceil d(k)^{d(k)} k^{2d(k)-1} \rceil + 1$. Using the same process as in Lemma 18, we compute s disjoint independent sets A_1, \dots, A_s of size k_1 in $G[A]$ and s disjoint independent sets B_1, \dots, B_s of size k_2 in $G[B]$. Again, if the process stops before we reach s independent sets, we know that a solution (with k_1 vertices of A and k_2 vertices of B) intersects a set of size at most $k_1(s - 1)$ or $k_2(s - 1)$ and we can branch (since s is bounded by a function of k).

Now we claim that there is at least one pair (A_i, B_j) (among the s^2 pairs) without any edge between A_i and B_j ; hence $A_i \cup B_j$ is an independent of size k . Suppose that this is not the case. Then, there is at least one edge between each pair (A_i, B_j) . Therefore the bipartite graph $\mathcal{B} := \mathcal{B}(\bigcup_{i \in [s]} A_i, \bigcup_{i \in [s]} B_i)$ has at least s^2 edges, and $sk_1 + sk_2 = sk$ vertices. As \mathcal{B} is also $K_{d(k), d(k)}$ -free, it has, by Kővári-Sós-Turán's theorem, at most $d(k)(sk)^{2 - \frac{1}{d(k)}}$ edges. But, by the choice of s , $s^2 > d(k)(sk)^{2 - \frac{1}{d(k)}}$, a contradiction.

We now deal with the case $k_1 = 0$. We show that if a solution exists with $k_1 = 0, k_2 = k$, then the branch $k_1 = 1, k_2 = k - 1$ also leads to a solution. Let us revisit that latter branch. We compute s disjoint independent sets B_1, \dots, B_s of size $k - 1$ in $G[B]$. Again, if this process stops before we reach s independent sets, we can branch on each vertex of a set of size at most $(k - 1)(s - 1)$. This branching also covers the case $k_2 = k$, since clearly, an independent

49:22 When Maximum Stable Set Can Be Solved in FPT Time

set of size k in $G[B]$ intersects those at most $(k-1)(s-1)$ vertices. Now, let A' be any set of s vertices in A and $\mathcal{B} := \mathcal{B}(A', \bigcup_{i \in [s]} B_i)$. By applying Kővári-Sós-Turán's theorem to \mathcal{B} as in the previous paragraph, there should be at least one pair $(u, B_j) \in A' \times \{B_1, \dots, B_s\}$ such that u is not adjacent to B_j .

We handle the case $k_2 = 0$ similarly, the conclusion being that we do not need to explore these branches. So we have described a decreasing FPT Turing-reduction creating less than $k(k+2)s$ instances (each with parameter $k' \leq k-1$), and we conclude by Lemma 5. ◀

D Pseudo-code for $P(1, t, t, t)$ -free graphs

■ **Algorithm 2** FPT algorithm for MIS on $P(1, t, t, t)$ -free graphs.

Precondition: G is $P(1, t, t, t)$ -free, $k \geq \alpha(G)$

```

1: function STABLE( $G, k$ ):
2:   if  $k \leq 2$  then solve in  $n^2$  by brute-force
3:   end if ▷ now  $k \geq 3$ 
4:   if  $G$  is  $2K_{4t}$ -free then solve in FPT time
5:   end if ▷ see [4]
6:    $C \leftarrow$  BuildCliqueCollection( $G, k$ )
7:    $R \leftarrow C \cup$  subclasses of size less than  $\eta$  ▷ small subclasses are set aside
8:   while  $\exists$  subclass  $Q$  of size at most  $|R|$  do
9:      $R \leftarrow R \cup Q$ 
10:  end while
11:   $\mathcal{P} \leftarrow$  remaining classes
12:  if  $\mathcal{P} = \emptyset$  then input is a kernel
13:  end if
14:   $P(I) \leftarrow$  remaining class with  $I$  maximal for inclusion
15:  if  $|\mathcal{P}| \geq 2$  then
16:    if  $\exists P(J) \in \mathcal{P}$  such that  $I \cap J = \emptyset$  then
17:       $(A, B, R)$  with  $\mathcal{B}(A, B)$   $K_{d(k), d(k)}$ -free ▷ Theorem 20
18:    end if
19:    if  $\forall P(J) \in \mathcal{P}, J \subseteq I$  then
20:       $(A, B, R)$  with  $\forall v \in B, v$  has co-degree  $\leq d_1(k)$  in  $A$  ▷ Lemma 17
21:      and  $(B_1, B_2, R)$  in  $G[B \cup R]$  with  $\mathcal{B}(B_1, B_2)$   $K_{d_2(k), d_2(k)}$ -free, ▷ Theorem 20
22:      or branching on  $2tk$  vertices,
23:      or critical case, when repeated, yields a  $2K_{2kt, 2kt}$ 
24:    end if
25:  end if
26:  if  $\mathcal{P} = \{P(I)\}$  then a vertex of  $C$  has small co-degree, ▷ see Observation 10
27:    or branching on  $2tk$  vertices,
28:    or critical case, when repeated, yields a  $2K_{2kt, 2kt}$ 
29:  end if
30: end function

```

The k -Fréchet Distance: How to Walk Your Dog While Teleporting

Hugo Alves Akitaya

Department of Computer Science, Tufts University, Massachusetts, USA
hugo.alves_akitaya@tufts.edu

Maike Buchin

Department of Mathematics, Ruhr University Bochum, Germany
maike.buchin@rub.de

Leonie Ryvkin

Department of Mathematics, Ruhr University Bochum, Germany
leonie.ryvkin@rub.de

Jérôme Urhausen

Department of Information and Computing Sciences, Utrecht University, Netherlands
J.E.Urhausen@uu.nl

Abstract

We introduce a new distance measure for comparing polygonal chains: the k -Fréchet distance. As the name implies, it is closely related to the well-studied Fréchet distance but detects similarities between curves that resemble each other only piecewise. The parameter k denotes the number of subcurves into which we divide the input curves (thus we allow up to $k - 1$ “teleports” on each input curve). The k -Fréchet distance provides a nice transition between (weak) Fréchet distance and Hausdorff distance. However, we show that deciding this distance measure turns out to be NP-hard, which is interesting since both (weak) Fréchet and Hausdorff distance are computable in polynomial time. Nevertheless, we give several possibilities to deal with the hardness of the k -Fréchet distance: besides a short exponential-time algorithm for the general case, we give a polynomial-time algorithm for $k = 2$, i.e., we ask that we subdivide our input curves into two subcurves each. We can also approximate the optimal k by factor 2. We then present a more intricate FPT algorithm using parameters k (the number of allowed subcurves) and z (the number of segments of one curve that intersect the ε -neighborhood of a point on the other curve).

2012 ACM Subject Classification Theory of computation \rightarrow Computational geometry; Theory of computation \rightarrow Design and analysis of algorithms; Theory of computation \rightarrow Fixed parameter tractability

Keywords and phrases Measures, Fréchet distance, Hardness, FPT

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.50

Related Version A full version of the paper is available at <http://arxiv.org/abs/1903.02353>.

Funding *Hugo Alves Akitaya*: supported by NSF awards CCF-1422311 and CCF-1423615, and the Science Without Borders scholarship program.

Jérôme Urhausen: supported by the Netherlands Organisation for Scientific Research under project 612.001.651.

Acknowledgements We would like to thank Erik Demaine for contributing the key idea for proving hardness in the free space diagram in Section 3.1, as well as the organizers and other participants of the Intensive Research Program in Discrete, Combinatorial and Computational Geometry in Barcelona, 2018, for providing the perfect environment to meet other researchers.



© Hugo Alves Akitaya, Maike Buchin, Leonie Ryvkin, and Jérôme Urhausen;
licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 50; pp. 50:1–50:15

Leibniz International Proceedings in Informatics



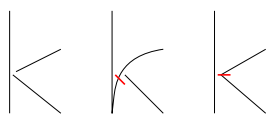
LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

During the last decades, several methods for comparing geometrical shapes have been studied in a variety of applications, e.g., analyzing geographic data, such as trajectories, or comparing chemical structures like protein chains or DNA. The Fréchet distance has been well-studied in the past since it has proven to be helpful in several of the mentioned applications. The Hausdorff distance, another similarity measure, has also proven useful in applications and can be computed more efficiently. However, it provides us with less information by taking only the overall positioning of curves into consideration, not how they are traversed.

We introduce the k -Fréchet distance as a distance measure in between Hausdorff and (weak) Fréchet distance. This measure allows us to compare shapes consisting of several parts: we cover the input curves by at most k (possibly overlapping) subcurves each and ask for a matching of the subcurves such that each pair of matched subcurves has at most weak Fréchet distance ε (where $\varepsilon > 0$ is a given constant). Note that there are in fact two variants of the k -Fréchet distance: the *cover variant* described above and the *cut variant*, where we partition the input curves into k disjoint subcurves each. These subcurves are then matched if and only if their (weak) Fréchet distance is small. In this paper, by k -Fréchet distance we always refer to the cover variant and mention the cut variant only briefly.

Thus the new measure allows us to find similarities between curves that need to be cut and reordered to be similar under the Fréchet distance. For instance, this could be objects of rearranged pieces such as a set of trajectories of tourists visiting several sights in a city. If the k -Fréchet distance of two trajectories is small, the respective tourists used similar routes to get to the sights. For small k we can also conclude that the tourists visited many sights in the same order. Other examples would be chemical structures or handwritten characters and symbols. An example is displayed in Figure 1, where we compare three variants of writing the letter k by hand. Note that we deal with disconnected curves by concatenating the respective subcurves. Of course, we can easily identify that all three of them are k's by using the Hausdorff distance to compare them to a “generic” k, but the k -Fréchet distance provides us with more information: the 2-Fréchet distance between the second and the third k is large because the strokes are set differently. Those k's are unlikely to be written by the same person. The 3-Fréchet distance, however, is small, because the letter consists of at most 3 strokes in general.



■ **Figure 1** Three ‘k’s written in a different way. For the middle and the right one, the 2-Fréchet distance is large and the 3-Fréchet distance is small.

Characterizing the mentioned variants of the Fréchet distance next to the Hausdorff distance intuitively shows that the new distance measure bridges between weak Fréchet and Hausdorff distance. As is common for the Fréchet distance, we use the following analogy: we interpret our input curves as two paths, which have to be traversed by a man and a dog, each of them walking on one of the paths. For the (weak) Fréchet distance we ask for the length of the shortest leash so that man and dog can traverse their curves. They may choose their speeds independently. For the weak Fréchet distance, man and dog are allowed to backtrack.

The Hausdorff distance finds for each point on either curve the closest point on the other curve and takes the largest of the obtained distances. In terms of man and dog we do not ask for traversal as such, we simply need that for any fixed position on either path there is a

position on the other one such that man and dog can stand on their respective positions using a leash of fixed length. One could say they may “teleport” on their curves any number of times as long as both man and dog can reach all positions on their respective curves without exceeding the given maximum distance, i.e., the leash length. The k -Fréchet distance limits this number of teleports to a constant k (actually, we have $k - 1$ teleports), so we want man and dog to traverse their paths piecewise. Note that we use the weak Fréchet distance as an underlying distance measure. As we picture man and dog to teleport, it is more natural to allow backtracking, especially since we allow any point on a curve to be the target point of a teleport. Imagine a subcurve oriented in the opposite direction than the subcurve closest to it: we would like to teleport the dog to the end of that curve and have it traverse the curve backward. Moreover, we do not ask to match the endpoints of our input curves.

Related work. Efficient algorithms were presented for computing the Fréchet distance and the weak Fréchet distance by Alt and Godau in 1995. They first introduced the concept of the free space diagram, which is key to computing this distance measure and its variants [3]. Following their work, numerous variants and extensions have been considered. Here we mention only a few results related to our work. Alt, Knauer and Wenk compared Hausdorff to Fréchet distance and discussed κ -bounded curves as a special input instance [4]. In particular, they showed that for convex closed curves Hausdorff distance equals Fréchet distance. For curves in one dimension Buchin et al. [6] proved the equality of Hausdorff and weak Fréchet distance using the well-known Mountain climbing theorem [18]. For computing the Hausdorff distance, Alt et. al. [2] gave a thorough overview. Buchin [9] gave the characterization of these measures in free space, which motivated our study of k -Fréchet distance.

For c -packed curves, Driemel, Har-Peled and Wenk presented a $(1 + \varepsilon)$ -approximation algorithm, which determines the Fréchet distance in near linear time [15]. For general polygonal curves, Buchin et al. [7] recently slightly improved the original algorithm of Alt and Godau, while Bringmann [5] showed that unless SETH fails no strongly subquadratic algorithm for the Fréchet distance exists. An interesting variant was presented by Gheibi et al.: they studied the weak Fréchet distance but minimized the length of the subcurves on which backtracking is necessary [17]. Buchin, Buchin and Wang studied partial curve matching, where they presented a polynomial-time algorithm to compute the “partial Fréchet similarity” [8], and a variation of this similarity was presented by Scheffer in [20]. Also, Driemel and Har-Peled defined a Fréchet distance with shortcuts [14], which was proven to be the first NP-hard variant of the Fréchet distance in [10].

Interestingly, both Hausdorff and (weak) Fréchet distance are computable in polynomial time. However, the k -Fréchet distance, as a distance measure that bridges between the two of them, proves to be NP-complete.

Overview. In the next chapter, we introduce and formally define the k -Fréchet distance. In Chapter 3, we determine its hardness in two steps: first, we prove NP-hardness of a simpler auxiliary problem to gain some intuition (Section 3.1) for the then following reduction. The most intricate part of our work is the construction of said reduction and analyzing its correctness, both of which are presented in Section 3.2. Finally, we present our algorithmic findings in Chapter 4. We give an XP-algorithm with parameter k , which even works in polynomial time for small k . A greedy approach leads to a 2-approximation on the optimal k . We then make use of two parameters, again the selection size k , and the parameter z , which indicates how “entangled” the input curves are, to construct our FPT algorithm.

2 Preliminaries

First we define the *Hausdorff distance* [4] for curves $P, Q: [0, 1] \rightarrow \mathbb{R}^d$ as

$$\delta_H(P, Q) = \max(\tilde{\delta}_H(P, Q), \tilde{\delta}_H(Q, P)), \text{ where}$$

$$\tilde{\delta}_H(P, Q) = \max_{t_1 \in [0, 1]} \min_{t_2 \in [0, 1]} \|P(t_1) - Q(t_2)\|$$

denotes the directed Hausdorff distance from P to Q . By $\|\cdot\|$ we refer to the Euclidean norm in \mathbb{R}^d . Now recall the *Fréchet distance* [3]: For curves $P, Q: [0, 1] \rightarrow \mathbb{R}^d$ it is given by

$$\delta_F(P, Q) = \inf_{\sigma} \max_{t \in [0, 1]} \|P(t) - Q(\sigma(t))\|,$$

where the reparametrizations $\sigma: [0, 1] \rightarrow [0, 1]$ range over all orientation-preserving homeomorphisms. A variant is the *weak Fréchet distance* δ_{wF} where both curves are reparameterised by σ and τ , respectively, which range over all continuous surjective functions.

As mentioned, the Fréchet distance is often illustrated by a man and a dog walking on two curves where both may choose their speed independently. For the Fréchet distance, man and dog may not backtrack, for the weak Fréchet distance they may. The (weak) Fréchet distance corresponds to the shortest leash length allowing them to traverse the curves.

A well-known characterization, which is key to efficient algorithms for computing both weak and (strong) Fréchet distance uses the free space diagram, which was introduced by Alt and Godau [3]. First we recall the free space F_ε :

$$F_\varepsilon(P, Q) = \{(t_1, t_2) \in [0, 1]^2: \|P(t_1) - Q(t_2)\| \leq \varepsilon\}.$$

For piecewise-linear P and Q , the free space diagram puts this information into an $(n \times m)$ -grid where n and m are the numbers of segments in P and Q respectively. For the rest of this paper we assume that $m = \mathcal{O}(n)$ to simplify runtime expressions.

The Fréchet distance of two curves is at most a given value ε if there exists a monotone path through the free space connecting the bottom left to the top right corner. For the weak Fréchet distance, this path need not be monotone. It may also start and end somewhere other than the corners of the diagram, as long as it touches all four boundaries.

We now define further terms regarding the free space diagram: A *component* of a free space diagram is a connected subset $c \subseteq F_\varepsilon(P, Q)$. A set S of components *covers* a set $I \subseteq [0, 1]_P$ of the parameter space (corresponding to the curve P) if I is a subset of the projection of S onto said parameter space, i.e., $\forall x \in I: \exists c \in S, y \in [0, 1]_Q: (x, y) \in c$. Covering on the second parameter space is defined analogously. This means the weak Fréchet distance is smaller than ε if there is one component in $F_\varepsilon(P, Q)$ that covers both parameter spaces. Similarly, the Hausdorff distance can be tested by checking whether the set of all components covers both parameter spaces. In this paper we extend this concept to also account for the number of components needed to cover the parameter spaces.

We define the *k -Fréchet distance* $\delta_{kF}(P, Q)$ as the minimal ε such that there is a set of at most k components of $F_\varepsilon(P, Q)$ covering both parameter spaces. That is, we cover the curves P and Q by at most k pieces (i.e., subcurves) such that there is a matching of the subcurves where a matched pair has small weak Fréchet distance. Note that the subcurves may overlap. In the analogy, we allow man and dog to “teleport” on their respective curves, i.e., they may skip parts of their paths and come back later. We still ask for a complete traversal, but some parts of the curves may be traversed multiple times with teleports in between.

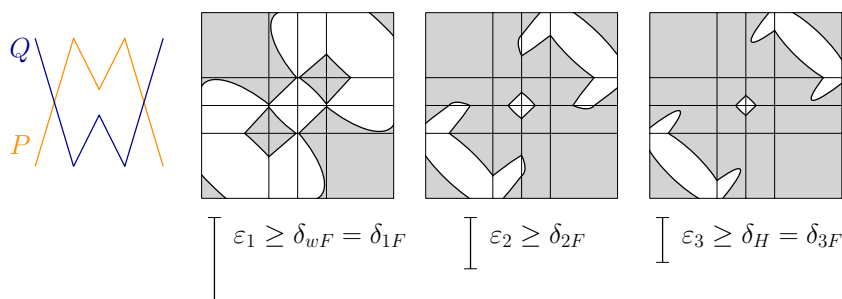
The decision problem for this distance measure asks whether for a fixed value of k , $\delta_{kF}(P, Q)$ is smaller than or equal to a given ε . Naturally, for a fixed real $\varepsilon > 0$, we would

like to cut the curves into as few subcurves as possible (optimization version). By definition, the k -Fréchet distance lies in between the Hausdorff and the (weak) Fréchet distances:

$$\delta_H(P, Q) \leq \delta_{kF}(P, Q) \leq \delta_{wF}(P, Q) \leq \delta_F(P, Q).$$

Also, the k -Fréchet distance decreases as k increases: for $k = 1$ it equals the weak Fréchet distance, whereas for k sufficiently large, e.g., $k \geq n^2$, it equals the Hausdorff distance.

Figure 2 illustrates this property. The diagram on the left corresponds to a fixed ε_1 . We observe that there is one connected component in the free space $F_{\varepsilon_1}(P, Q)$ that projects surjectively onto both parameter spaces. We therefore have $\varepsilon_1 \geq \delta_{wF} (= \delta_{1F})$. The diagram in the middle depicts $F_{\varepsilon_2}(P, Q)$ for a value ε_2 slightly smaller than ε_1 . In that case two components cover the parameter spaces, which means $\varepsilon_2 \geq \delta_{2F}$. The free space $F_{\varepsilon_3}(P, Q)$ shown on the right for an ε_3 smaller than ε_2 consists of three components and all three are necessary to cover the parameter spaces. Furthermore, reducing the value of ε_3 even more would not split up the components into smaller subcomponents, but would just result in the set of all components not covering the parameter spaces any more. So we have $\varepsilon_3 \geq \delta_H = \delta_{3F}$.



■ **Figure 2** Comparison of weak Fréchet, 2-Fréchet and Hausdorff distance of curves P and Q .

3 Hardness results

In this section, we prove that deciding the k -Fréchet distance for fixed ε is NP-hard.

To give some intuition for the later proof, we first present a reduction from the well-known 3-SAT problem to the problem of covering two sides of a rectangle by selecting a number of smaller rectangles, or boxes, that are situated inside. This problem (we call it the box problem) mimics selecting the components in the free space to cover the parameter spaces. However, we do not ask to find curves that realize this specific free space.

Afterwards we reduce from rectilinear monotone planar 3-SAT [13] to prove hardness of the actual k -Fréchet distance problem.

3.1 Gaining intuition: The box problem

We want to reduce from the following classical NP-hard satisfiability problem [16]:

3-SAT:

INPUT: a boolean formula with n variables written as a conjunction of m clauses, where a clause is a disjunction of at most 3 literals;

OUTPUT: “Yes” if there exists a satisfying variable assignment, “No” otherwise.

Box problem:

INPUT: a set A of aligned, interior-disjoint rectangles b_i , their bounding box B , $k \in \mathbb{N}$;
 OUTPUT: “Yes” if there exists a selection of at most k rectangles from A such that their union surjectively projects onto the bottom and left boundary of B , “No” otherwise.

Given any instance of a 3-SAT formula, we build a bounding box B containing a number of boxes b_i such that we can find a covering selection of size k if and only if there is an assignment for the formula that outputs true. A *covering selection* of boxes is a subset of the b_i that projects surjectively onto the bottom and left boundaries of B . For this we build boxes b_i that correspond to the variables and any satisfying assignment of the variables can be directly “translated” into a covering selection of the b_i .

First, note that we assume that no clause contains duplicates, i.e., no clause is of the form $v \vee v \vee w$. The duplicates can be deleted without changing the boolean function induced by the formula. Note that clauses of the form $v \vee \neg v \vee w$ are allowed. Additionally, we require that throughout the formula each literal appears at least once, i.e., each variable appears at least once in its positive and in its negated form. For each variable v where this is not the case we add the clause $v \vee \neg v$ (colored dark green in Figure 3). These clauses are always fulfilled and therefore do not change the output of our boolean formula. We add at most n clauses in this way, which means that the size of the formula only changes polynomially in the input size.

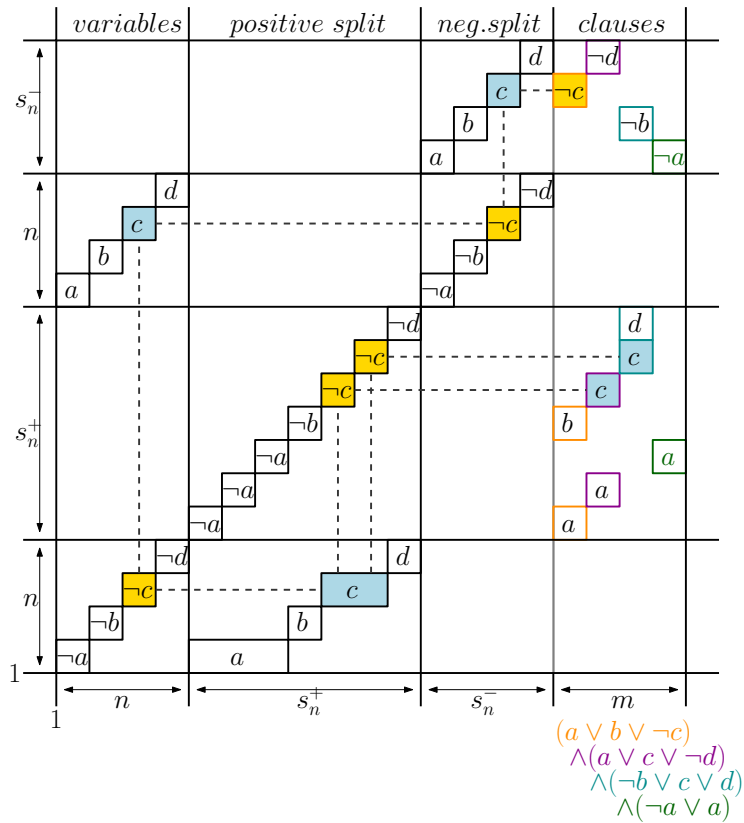
Now we give the detailed construction of our box problem instance derived from some 3-SAT formula: Let $V = \{v_1, \dots, v_n\}$ be the set of variables and let $C = \{c_1, \dots, c_m\}$ be the set of clauses. For each variable v_i , let a_i^+ (respectively a_i^-) be the number of clauses in which v_i appears positive (respectively negatively), and let $\{c_{i,1}^+, c_{i,2}^+, \dots, c_{i,a_i^+}^+\}$ (respectively $\{c_{i,1}^-, \dots, c_{i,a_i^-}^-\}$) be the set of clauses in which v_i appears positive (respectively negatively). Additionally we define the sums $s_i^+ = \sum_{k=1}^i a_k^+$ and $s_i^- = \sum_{k=1}^i a_k^-$.

In the following we describe the placement of boxes, which is depicted in Figure 3. The number of rows and columns needed for the different gadgets is indicated in the figure. A box (x, y, w, ℓ) designates the axis-aligned rectangle with unit height and width w whose bottom left corner has coordinates $(x, y) \in \mathbb{R}^2$ with label ℓ . The labels are later used in the proof of correctness.

Variable gadget. For each variable v_i , we place two boxes $(i, i, 1, \neg v_i)$ and $(i, i+n+s_n^+, 1, v_i)$, and no other boxes are placed over the interval $(i, i+1)$ of the bottom boundary. That way, in order to cover said interval, at least one of those two boxes has to be chosen.

Split gadget. The split gadget ensures that we can propagate the assignment of a variable onto all clauses the variable takes part in. We build the splits used for the positive occurrences of the variables first. For each variable v_i , we place the box $(1+n+s_{i-1}^+, i, a_i^+, v_i)$ and the boxes $(n+s_{i-1}^+ + j, n+s_{i-1}^+ + j, 1, \neg v_i)$, for $j \in \{1, \dots, a_i^+\}$. For negated occurrences of $v_i \in V$ we place the box $(1+n+s_n^+ + s_{i-1}^-, n+s_n^+ + i, a_i^-, \neg v_i)$ and the boxes $(n+s_n^+ + s_{i-1}^- + j, 2n+s_n^+ + s_{i-1}^- + j, 1, v_i)$, for $j \in \{1, \dots, a_i^-\}$.

Clause gadget. We assign to each clause c_i the unit interval on the bottom boundary of B starting at $I(c_i) = n+s_n^+ + s_n^- + i$. For each literal of a clause c_i we place a box labeled with the respective literal above the unit interval $[I(c_i), I(c_i) + 1]$. To be precise, for each $v_i \in V$ we place the boxes $(I(c_h), n+s_{i-1}^+ + j, 1, v_i)$, for $j \in \{1, \dots, a_i^+\}$ and $h \in \{1, \dots, m\}$ where $c_h = c_{i,j}^+$, and $(I(c_h), 2n+s_n^+ + s_{i-1}^- + j, 1, \neg v_i)$, for $j \in \{1, \dots, a_i^-\}$, $h \in \{1, \dots, m\}$ where this time $c_h = c_{i,j}^-$.



■ **Figure 3** Construction of the box problem instance and propagation of assignment.

Overall, we have $4n + 2(m_1 + 2m_2 + 3m_3)$ boxes, where m_i is the number of clauses with i variables (and therefore $m_1 + m_2 + m_3 = m$). Each unit interval $(i, i + 1)$ with $i \in \{1, \dots, 2n + s_n^+ + s_n^-\}$ on the left boundary of B can be covered by exactly two different boxes. The same holds for every unit interval $(i, i + 1)$ with $i \in \{1, \dots, n + s_n^+ + s_n^-\}$ on the bottom boundary. Note that for all these unit intervals, one of the boxes is labeled with a variable and the other one is labeled with the negated version of that variable, i.e., one box is labeled v and the other one $\neg v$. Each Interval $I(c)$ on the bottom boundary can be covered by as many boxes as the clause c contains literals. The labels of these boxes correspond to the variables contained within this clause. We set the bounding box B as the axis-aligned rectangle spanned by the points $(1, 1)$ and $(1 + n + s_n^+ + s_n^- + m, 1 + 2n + s_n^+ + s_n^-)$ and we set $k = 2n + m_1 + 2m_2 + 3m_3$ so only half the boxes can be chosen. For a given boolean formula, the set of boxes defined above can be determined in polynomial time.

► **Theorem 1.** *The box problem is NP-hard.*

Proof. First we prove that the box problem as constructed above has a solution if and only if the input 3-SAT formula has a variable assignment such that it evaluates to **true**.

“ \Leftarrow ”. Let $f : V \rightarrow \{\mathbf{true}, \mathbf{false}\}$ be an assignment of the variables that satisfies the 3-SAT formula. We set $S = \{\text{boxes } (x, y, w, v) \mid f(v) = \mathbf{true}\} \cup \{\text{boxes } (x, y, w, \neg v) \mid f(v) = \mathbf{false}\}$. The set S projects surjectively onto the bottom and left boundary of the bounding box B because each unit interval on the left boundary is covered by exactly one box. For most of the bottom boundary we also have that each interval is uniquely covered, but for the clauses columns we allow that more than one box per unit interval is chosen (i.e., more than one corresponding literal is set to **true**).

“ \Rightarrow ”. Let S be a minimal set of boxes that covers the boundaries of the bounding box B with $|S| = k$. This means that each unit interval on the left boundary of B is covered by exactly one box. Due to the position of the boxes, this means that for each variable v either all boxes labeled v or all boxes labeled $\neg v$ have been chosen. This induces an assignment of the variable v , i.e., v is set to **true** if the boxes labeled v have been chosen and else v is set to **false**. Note that the selection S covers the box B . Therefore, for each clause c one of the boxes that can cover $I(c)$ is an element of the selection S . It follows that the assignment of variables induced by S fulfills the formula.

Above we showed the NP-hardness of the box problem. The box problem is in fact even NP-complete since for a given subset S of boxes one can test if the bounding box B is covered by simply marking the covered intervals, which can be done in polynomial time. ◀

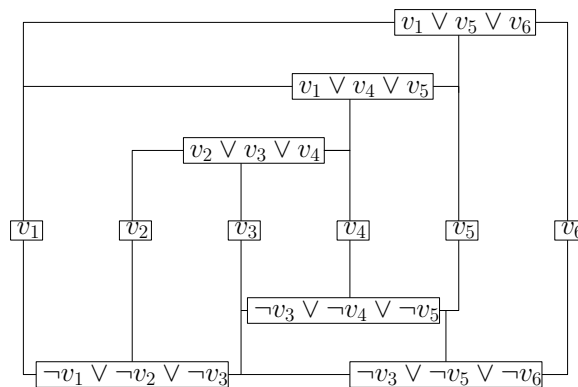
We can interpret the box problem as the problem of finding a selection of components in the free space that cover the parameter spaces. The small boxes can be seen as bounding boxes of actual components (for the projection there is no difference) and the bottom and left boundary of the large box B correspond to the respective parameter spaces. The above hardness proof, especially the construction of the boxes, provides us with the key ideas to prove hardness of the k -Fréchet distance. Next, we construct actual curves where certain intervals on the parameter spaces of the free space diagram each have two components that could cover them. As with the box problem, the choice we make for one of those intervals determines the choices for other intervals as we still need to ensure that the selection size is minimal in the end. The propagation of choices works in the same manner for the box problem as for the k -Fréchet distance problem.

3.2 Reduction for the k -Fréchet distance

We use the following variant of the 3-SAT problem in this subsection.

Rectilinear monotone planar 3-SAT:

INPUT: a 3-SAT formula with only all positive or all negated variables per clause, embedded as a graph with rectilinear, non-crossing edges; variables are drawn as vertices on a horizontal line, positive clauses are vertices drawn above this line and negative clauses are drawn below; OUTPUT: “Yes” if there exists a satisfying assignment for the variables, “No” otherwise.



■ **Figure 4** Instance of rectilinear monotone planar 3-SAT.

Note that we assume that each variable appears in at least one positive and one negative clause. Otherwise, we could simply define the occurring literal to be **true** (or **false**, respectively) and omit the clauses the literal appears in.

We can draw any graph corresponding to such a 3-SAT formula on a grid, see, e.g., Figure 4, which is useful when constructing and analyzing our curves. Since rectilinear monotone planar 3-SAT is NP-hard [13], we prove hardness of the k -Fréchet distance problem by reducing from it.

k -Fréchet distance problem:

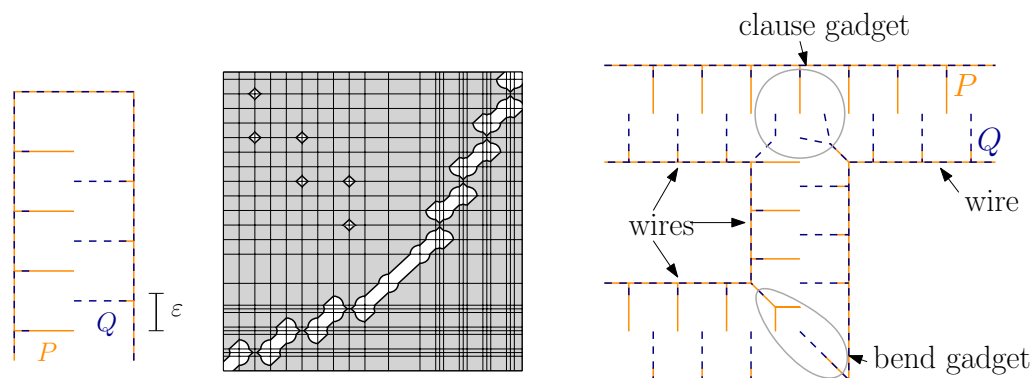
INPUT: Two polygonal curves P and Q , a distance ε and a natural number k ;

OUTPUT: “Yes” if there exists a selection of at most k components in the free space diagram F_ε such that their union projects surjectively onto both parameter spaces, “No” otherwise.

Our goal is to construct two curves P (yellow) and Q (blue) that mimic any input instance of a rectilinear monotone planar 3-SAT graph and show that in the free space resulting from these curves we can find a covering selection of size k if and only if there exists a satisfying assignment for the formula. The detailed construction can be found in the full version of this paper [1].

Overall we create wire and clause gadgets to represent variables and clauses, where wires correspond to the edges of the input graph. Wire gadgets allow a boolean choice that is propagated consistently throughout the wire. Clause gadgets test whether at least one incoming wire carries an appropriate choice.

Figure 5 shows a wire gadget and how it is used. Both curves consist of two long parallels, which we call *base parts* of the curves, and *spikes*, which are the horizontal segments in the figure. The spikes are formed by taking a 90 degree turn from the base part and traversing the spike segment back and forth. The base parts are not particularly relevant for the analysis because the segments forming them can only be covered by larger components that are always part of any covering selection. The value ε is chosen such that two adjacent spikes are just within distance ε . It follows that the spikes induce components that are similar to the boxes of Subsection 3.1. We say that a spike s is *covered* by an adjacent spike t of the other curve if the component of the free space diagram that covers the two intervals induced by these spikes is chosen for the covering selection. In the end, we choose k such that each blue spike in any gadget can only be covered by one single adjacent yellow spike. The choice for blue spikes must be consistent along the wire to preserve minimality of k , and it encodes the assignment of the corresponding variable.



■ **Figure 5** (Left and middle) The wire gadget and its corresponding free space diagram. Note that we connected the curves to give a small example, but the horizontal segment on top is not part of the wire itself. (Right) A part of the construction where wires connect other gadgets.

As displayed in Figure 5, the clause gadget features one yellow spike that can be covered by either one of the three blue spikes within its ε -neighborhood. Which one of their neighboring

yellow spikes the blue ones cover is determined by the variable assignment and propagated throughout the wire, so if at least one of the variables is set to `true`, the yellow spike at the center of the clause is covered.

Next, we need a number of other gadgets, too. As mentioned, the wires correspond to edges in the rectilinear monotone planar 3-SAT instance. To draw them coherently we need to make sure we can make 90 degree turns (so-called *bends*, see the right-hand side of Figure 5) and do T-crossings, i.e., *split* a wire into two.

We need to treat remaining difficulties: first of all, there is a *connection gadget* that enables us to connect the opposite base parts of P and Q , respectively. The resulting curves are closed, which we solve by applying the *scissor gadget*. Finally, we may need to change which of the curves has spikes on a specific side to draw the other gadgets consistently, so we also built a *color gadget* to “switch” the color pattern of the spikes.

At last, we want to connect all gadgets such that the resulting curves follow the embedding of the input graph G . Recall that the input is a grid embedding. We first scale the grid by a factor of 2^{10} to place all gadgets consistently. Note, that we have to deal with 2-clauses and take into account that our split gadget is directed, so we need to have some space for workarounds. Afterwards we can draw the curves’ vertices on grid points only. Consider the input graph G . We want to traverse all edges of G twice, once per inner, once per outer base parts. To do so, we have to “walk around” each face of G . To switch between faces we use connection gadgets. We obtain a traversal order of the faces by computing a minimum spanning tree of the dual graph, see [1] for a detailed description.

Finally, it remains to prove that our construction works in the sense that the curves have k -Fréchet distance ε if and only if the specific 3-SAT instance is satisfiable.

First, we note that the complexity of our curves is polynomial in the size of our input instance: the numbers of variables and clauses, but also the number of splits and the length of the edges determine the number of spikes and therefore also the number of components in the free space diagram. A spike induces either two or three components, depending on whether it is part of a specific gadget, i.e., a clause, or not. In addition, the gadgets induce a number of components, called *clutter*, that are always part of a covering selection. Some gadgets also induce a constant number of unnecessary components that are never chosen.

Our goal is to cover the parameter spaces with k components. We definitely need to select all clutter components and we need to cover all spikes, therefore we need to select (at least) one component per spike. We set k to be the number of clutter components plus the number of blue spikes (spikes of Q). It follows that each blue spike can only be covered once, which ensures that choices are propagated.

► **Theorem 2.** *It is NP-hard to decide whether $\delta_{kF}(P, Q) \leq \varepsilon$ for given polygonal curves P and Q , integer k , and $\varepsilon > 0$ where δ_{kF} denotes the k -Fréchet distance.*

For the full proof, we refer to the arXiv-Version of this paper [1]. For the reduction constructed above, the following holds: given a satisfying assignment for the input formula, we know which components to select: apart from all clutter components, we have to decide how to cover the blue spikes. This choice is implied by the assignment and propagated throughout the gadgets.

Given a selection of components, we need to backtrack our choices throughout the wires and other gadgets to determine how the blue spikes are covered. Depending on this choice, we know whether the corresponding variable has to be set to `true` or to `false`. Thus we derive our assignment for the 3-SAT formula and complete the proof of NP-hardness.

We can test in polynomial time whether the union of a selection of components covers the parameter spaces. Thus the problem of deciding the k -Fréchet distance lies in NP.

4 Algorithms

In this chapter, we begin by presenting a preprocessing algorithm, which applies to the following algorithmic approaches: First, we can find a covering selection of at most size k in exponential time and describe how to approximate k by factor 2 (Section 4.2). Then, we describe an FPT-algorithm for finding an optimal covering selection in Section 4.3. Note that the XP-algorithm as well as the FPT-algorithm are designed to solve the decision problem, but we can also optimize k by repeating the decision problem solving algorithm for different values of k by performing a parametric search on the reasonable values for k , similar to the algorithm for the Fréchet distance for polygonal curves by Alt and Godau [3].

4.1 Preprocessing

First, we observe two preprocessing strategies, which can be applied before entering any of our algorithms. In any case we start by computing the free space diagram, which takes quadratic time. In the free space diagram it is easy to identify all *necessary* components: any component that covers an interval of one of the parameter spaces uniquely (i.e., there is no other component covering the exact same interval) is necessarily chosen for an output selection. Such components can be found in $\mathcal{O}(n \log n)$ time using a scan. Furthermore, it is possible to rule out all *redundant* components. A component is called redundant if and only if it is completely contained in the bounding box of a different component (but there could be more than one such component with a sufficiently large bounding box). This case can also be detected via scans. Thus our preprocessing needs quadratic time. However, it does not improve the size of the input (being the complexity of the free space) nor the resulting runtime of any of the presented algorithms asymptotically.

4.2 XP-algorithm and approximation

We start by giving the more straight-forward approaches. First, we present an XP-algorithm.

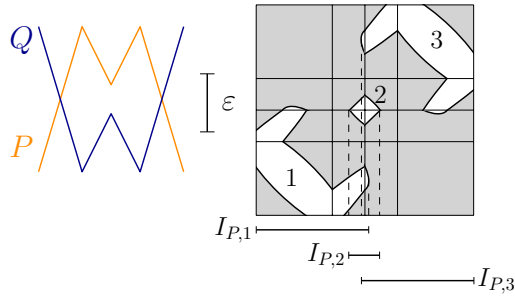
► **Remark 3.** The k -Fréchet distance can be decided in $\mathcal{O}(k \cdot n^{2k})$ time for constant k .

The brute force approach simply checks for all selections of k components of the free space whether their joint projections cover both parameter spaces surjectively. That means we have to check at most $\binom{n^2}{k}$ possible combinations of components resulting in a runtime of $\mathcal{O}(k \cdot n^{2k})$ for fixed k , which is of course only feasible for very small k . Therefore we can compute the answer to the decision problem for the cover distance with $k = 2$ in $\mathcal{O}(n^4)$. Since $\binom{m}{k} \leq 2^m$ holds for any $m > k$, our runtime is upper-bounded by $\mathcal{O}(n \cdot 2^{n^2})$ for general k . We can also approximate the size of an optimal solution.

The main idea of our algorithm is to find minimal covering selections for each parameter space individually and combine those selections into an overall solution in the end. We can find both selections covering only a single parameter space by applying a greedy technique.

Given the free space diagram, we first project all components onto the parameter spaces. We get two sets of intervals, one covering the first parameter space (we store these intervals in the list L_P , see Figure 6) and one for the second parameter space (stored in L_Q). So one component projects onto two intervals, one on each parameter space (and thus one per list). We store the information on which two intervals stem from the same component accordingly.

Now we simply want to select a minimum number of intervals whose union equals the unit interval, i.e., the parameter space. We deal with each parameter space on its own as follows: we sort the lists L_P (and L_Q) by left endpoint. Now, per list, starting at 0, we make a greedy choice and select the interval (among the intervals starting at 0) with the



■ **Figure 6** The projection onto the first parameter space and the resulting elements of L_P .

rightmost endpoint, say r_1 . Here we recurse, i.e., we take r_1 as new start point and again search among the intervals covering r_1 (i.e., intervals starting at or to the left of r_1) for the one with the rightmost endpoint. As soon as we select an interval with 1 as endpoint we have found a minimal covering selection. To see that our greedy strategy is optimal, observe that the algorithm proceeds from left to right maintaining the following invariant: at any time we selected a minimum number of intervals to cover the parameter space from its left boundary to the current position.

As output we have two selections of intervals, S_P and S_Q . The intervals correspond to components. We build the union of both lists, taking into account that an interval in S_P may belong to the same component as an interval of S_Q , and output the selection of components S that contributed at least one of the chosen intervals.

The worst case that might occur is the following: all of the intervals we selected during the greedy procedures correspond to different components in the free space, so that the union of our selections is of size $|S_P| + |S_Q|$. A different selection of size $|\hat{S}| = \max(|S_P|, |S_Q|)$ might cover both parameter spaces but is not detected by the greedy scan. Schäfer proves that the approximation factor 2 is indeed tight [19].

Finally, we consider the runtime: computing the free space takes quadratic time. Sorting the lists adds another logarithmic factor while the greedy selection routine takes linear time in the number of intervals. Hence we get an overall runtime of $\mathcal{O}(n^2 \log n)$.

► **Theorem 4.** *The algorithm described above runs in $\mathcal{O}(n^2 \log n)$ time and finds a selection of components that covers both parameter spaces if and only if one exists. A found selection contains at most twice the minimum number of components needed.*

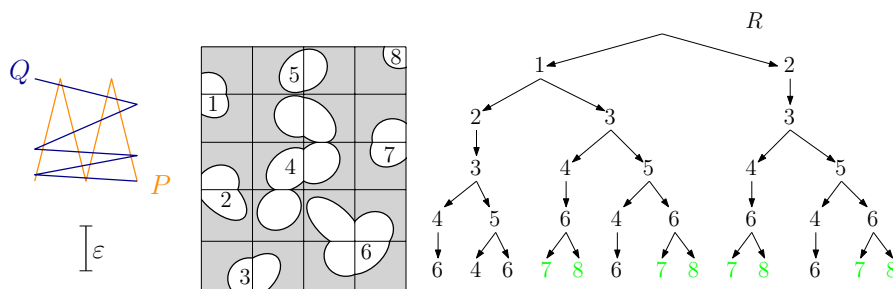
4.3 Fixed-parameter tractability

Lastly, we present an algorithm for deciding whether $\delta_{kF}(P, Q) \leq \epsilon$ for given ϵ and k . The runtime of our algorithm is polynomial in the complexity of our curves P and Q , but exponential in the two parameters k (the selection size) and z (the neighborhood complexity).

We define the *neighborhood complexity* as the maximum number of segments of one curve that intersect with the ϵ -neighborhood of any point of the other curve. In the free space diagram we get that each horizontal or vertical line intersects at most z components. Note that this requirement is similar but not directly related to c -packedness. The first counts the number of segments of one curve within a fixed distance of any point of the other curve. In contrast, c -packedness bounds the length of one curve within a ball of arbitrary radius.

The idea is the following: we build two directed bounded search trees (as described in Chapter 3 of [12]) to create selections of components of size at most k . Each search tree represents the projection of the free space onto one parameter space, see Figure 7 below. A

node corresponds to a component in the free space (or rather the interval on the respective parameter space it covers) and a path encodes a selection that covers the interval $[l, r]$. By l we denote the left boundary point of the interval corresponding to the root of the path and r is the right boundary point of the interval of the bottommost node (e.g. a leaf). We call a selection or a path *feasible* if the union of the (at most k) components it encodes covers the respective parameter space. From the first tree, T_P , we are able to extract all feasible selections which cover the parameter space corresponding to curve P , feasible selections of the second tree, T_Q , cover the other parameter space. In the end, we compare and/or combine a feasible selection of T_P with a feasible selection of T_Q to get a selection S that contains no more than k components, so that its union covers both parameter spaces.



■ **Figure 7** Curves P, Q , their free space diagram and the resulting bounded search tree T_P . Leaves of feasible paths are marked in green.

More formally, we build two trees of depth k and branching factor z . Consider the tree T_P . The root is labeled by the left boundary point of the parameter space of P (we assume w.l.o.g. that the bottom boundary of the free space diagram corresponds to P). Now we use a sweep line initialized at the left boundary of the free space diagram. We assign a node in the tree to all components intersecting the sweep line, i.e., the root has as many children as there are components touching the left boundary of the free space diagram. The sweep line moves to the right. Whenever the sweep line is tangent to a component, one of two cases occur: if it touches the leftmost point of a component it becomes *active*, i.e., the sweep line continues to intersect this component when moving further to the right; if the line touches the rightmost point of the component, it becomes *inactive* (so the sweep line just stops to intersect it). In the first case, nothing immediate happens to the tree, in the latter case, if the tangent component already has a node in the tree, we insert new nodes: each node corresponding to the tangent component gets assigned as many children as there are other currently active components. By definition, a node can never have more than z children. Note that some (small) components may not get assigned any nodes in one tree. Also, with every node we store its depth (the root has depth 0) and stop assigning children at depth k - or as soon as a component touches the right boundary of the free space diagram. If a leaf v_l corresponds to a component touching the right boundary, the path from the root to v_l encodes a feasible selection of components for T_P . Other selections are called *non-feasible*. The second tree T_Q is built analogously by sweeping from bottom to top.

We store the feasible selections obtained from T_P and T_Q in sorted lists L_P^T and L_Q^T . For each pair of selections $S_{P,i}, S_{Q,j}$, where $1 \leq i, j \leq z^k$, we test whether $|S_{P,i} \cup S_{Q,j}| \leq k$ and output this union if the answer is positive.

► **Theorem 5.** *The algorithm described above returns a selection S of k components in the free space that covers both parameter spaces if and only if such a selection exists. Hence it solves the decision problem for the k -Fréchet distance in time $\mathcal{O}(nz + kz^{2k})$.*

Proof. Our algorithm treats all possible selections of size at most k per parameter space and combines all these, hence it necessarily finds a valid solution if and only if one exists.

For the first step, we compute the free space, which takes $\mathcal{O}(n \cdot z)$ time. Building the trees takes $\mathcal{O}(z^k)$ time since we are limited to depth k and insert at most z children per node. Note that any operation in the free space diagram, such as detecting components that cover a boundary or projecting them onto the boundaries to find the intervals they cover, can be done in $\mathcal{O}(nz)$ time. We obtain at most z^k selections per search tree. Each selection is stored as the sorted set $S_{P,i}$, respectively $S_{Q,i}$, by encoding each component as an integer. We then sort both lists of selections lexicographically. During the sorting we might detect duplicates, which we discard immediately. We then compare each selection of the first list L_P^T to each selection of L_Q^T , taking time k per comparison. For any selection smaller than k we can test whether its union with a selection of the other list is still a solution, i.e., whether the unified selection does not have more than k integers. All in all, we have a runtime of $\mathcal{O}(zn + z^k + z^k \cdot k \log k + k(z^k)^2) = \mathcal{O}(zn + kz^{2k})$. ◀

By combining a greedy approach with building an interval tree, Schäfer improved the runtime of our FPT algorithm to $\mathcal{O}(nz + k \cdot (\log n + z) \cdot z^k)$ in [19].

5 Conclusion

We present a novel variant of the Fréchet distance for polygonal chains that allows to compare objects of rearranged pieces. We ask for k (possibly overlapping) subcurves per input curve that have pairwise small weak Fréchet distance. Thus, the k -Fréchet distance provides a transition between weak Fréchet ($k = 1$) and Hausdorff distance (k sufficiently large).

But as we prove, deciding the k -Fréchet distance of two polygonal curves is NP-hard. However, we were able to tackle the computational challenge from different angles: we give an XP-algorithm depending on k , approximate k by factor 2 and present an FPT-algorithm.

As mentioned in the introduction, there is a second variant of defining the k -Fréchet distance we call the “cut version”: instead of allowing overlapping subcurves, we cut the curves into pieces and search for a matching between these pieces. The NP-hardness proof presented at EuroCG2018 [11] holds for this variant. However, the algorithmic approaches only work for the variant we discuss in this paper (we call it the “cover variant”). Finding algorithmic approaches for the cut version of the k -Fréchet distance is work in progress.

References

- 1 Hugo A. Akitaya, Maïke Buchin, Leonie Ryvkin, and Jérôme Urhausen. The k -Fréchet distance. *CoRR*, 2019. [arXiv:1903.02353](https://arxiv.org/abs/1903.02353).
- 2 Helmut Alt, Peter Braß, Michael Godau, Christian Knauer, and Carola Wenk. Computing the Hausdorff distance of geometric patterns and shapes. In *Discrete and computational geometry*, volume 25 of *Algorithms Combin.*, pages 65–76. Springer, 2003.
- 3 Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *Internat. J. Comput. Geom. Appl.*, 5:75–91, 1995.
- 4 Helmut Alt, Christian Knauer, and Carola Wenk. Comparison of distance measures for planar curves. *Algorithmica*, 38(1):45–58, 2004.
- 5 Karl Bringmann. Why Walking the Dog Takes Time: Fréchet Distance Has No Strongly Subquadratic Algorithms Unless SETH Fails. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 661–670, 2014.

- 6 Kevin Buchin, Maike Buchin, Christian Knauer, Günther Rote, and Carola Wenk. How Difficult is it to Walk the Dog? In *23rd European Workshop on Computational Geometry (EuroCG)*, pages 170–173, 2007.
- 7 Kevin Buchin, Maike Buchin, Wouter Meulemans, and Wolfgang Mulzer. Four Soviets Walk the Dog: Improved Bounds for Computing the Fréchet Distance. *Discrete and Computational Geometry*, 58:180–216, 2017.
- 8 Kevin Buchin, Maike Buchin, and Yusu Wang. Exact algorithms for partial curve matching via the Fréchet distance. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '09*, pages 645–654, 2009.
- 9 Maike Buchin. *On the Computability of the Fréchet Distance Between Triangulated Surfaces*. PhD thesis, Free University Berlin, Institute of Computer Science, 2007. URL: http://www.diss.fu-berlin.de/diss/receive/FUDISS_thesis_000000002618.
- 10 Maike Buchin, Anne Driemel, and Bettina Speckmann. Computing the Fréchet distance with shortcuts is NP-hard. In *Proceedings of the Thirtieth Annual Symposium on Computational Geometry, SOCG'14*, pages 367–376. ACM, 2014. doi:10.1145/2582112.2582144.
- 11 Maike Buchin and Leonie Ryvkin. The k-Fréchet distance of polygonal curves. In *34th European Workshop on Computational Geometry (EuroCG)*, 2018. URL: conference.imp.fu-berlin.de/eurocg18/.
- 12 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*. Springer, 2015.
- 13 Mark de Berg and Amirali Khosravi. Optimal binary space partitions for segments in the plane. *Internat. J. Comput. Geom. Appl.*, 22:187–205, 2012.
- 14 Anne Driemel and Sariel Har-Peled. Jaywalking your Dog - Computing the Fréchet Distance with Shortcuts. *CoRR*, 2011. arXiv:1107.1720.
- 15 Anne Driemel, Sariel Har-Peled, and Carola Wenk. Approximating the Fréchet distance for realistic curves in near linear time. *Discrete and Computational Geometry*, 48:94–127, 2012.
- 16 Michael R. Garey and David S. Johnson. *Computers and intractability*. W. H. Freeman and Co., San Francisco, Calif., 1979.
- 17 Amin Gheibi, Anil Maheshwari, Jörg-Rüdiger Sack, and Christian Scheffer. Minimum backward Fréchet distance. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL '14*, pages 381–388. ACM, 2014. doi:10.1145/2666310.2666418.
- 18 Jacob E. Goodman, János Pach, and Chee-K. Yap. Mountain climbing, ladder moving, and the ring-width of a polygon. *Amer. Math. Monthly*, 96:494–510, 1989.
- 19 Peter Schäfer. *Untersuchungen zu Varianten des Fréchet-Abstands*. Master's thesis, Fernuniversität Hagen, 2019.
- 20 Christian Scheffer. More Flexible Curve Matching via the Partial Fréchet Similarity. *Int. J. Comput. Geometry Appl.*, 26:33–52, 2016.

New Applications of Nearest-Neighbor Chains: Euclidean TSP and Motorcycle Graphs

Nil Mamano^a 

Department of Computer Science,
University of California, Irvine, USA
nmamano@uci.edu

^a Corresponding author

David Eppstein

Department of Computer Science,
University of California, Irvine, USA
eppstein@uci.edu

Michael T. Goodrich 


Department of Computer Science,
University of California, Irvine, USA
goodrich@uci.edu

Pedro Matias 

Department of Computer Science,
University of California, Irvine, USA
pmatias@uci.edu

Alon Efrat

Department of Computer Science,
University of Arizona, Tucson, USA
alon@cs.arizona.edu

Daniel Frishberg 

Department of Computer Science,
University of California, Irvine, USA
dfrishbe@uci.edu

Stephen Kobourov 

Department of Computer Science,
University of Arizona, Tucson, USA
kobourov@cs.arizona.edu

Valentin Polishchuk

Communications and Transport Systems,
ITN, Linköping University, Sweden
valentin.polishchuk@liu.se

Abstract

We show new applications of the nearest-neighbor chain algorithm, a technique that originated in agglomerative hierarchical clustering. We use it to construct the greedy multi-fragment tour for Euclidean TSP in $O(n \log n)$ time in any fixed dimension and for Steiner TSP in planar graphs in $O(n\sqrt{n} \log n)$ time; we compute motorcycle graphs, a central step in straight skeleton algorithms, in $O(n^{4/3+\varepsilon})$ time for any $\varepsilon > 0$.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Computational geometry

Keywords and phrases Nearest-neighbors, Nearest-neighbor chain, motorcycle graph, straight skeleton, multi-fragment algorithm, Euclidean TSP, Steiner TSP

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.51

Related Version A full version of the paper is available on <https://arxiv.org/abs/1902.06875>.

Funding *David Eppstein*: Supported in part by NSF grants CCF-1618301 and CCF-1616248.

Michael T. Goodrich: Supported in part by NSF grant 1815073.

Stephen Kobourov: Supported in part by NSF grants CCF-1740858, CCF-1712119, DMS-1839274, and DMS-1839307.

1 Introduction

The *nearest-neighbor chain* (NNC) technique is used for agglomerative hierarchical clustering [36, 35], and has only seen one other use besides it, in stable matching [22]. In this paper, we use it to speed up an algorithm for Euclidean TSP called multi-fragment heuristic. We also use it to speed up the computation of motorcycle graphs, which is a central step in algorithms for computing straight skeletons. In the full version of the paper [32], we follow this approach for two additional problems: a special case of stable matching and a



© Nil Mamano, Alon Efrat, David Eppstein, Daniel Frishberg, Michael T. Goodrich, Stephen Kobourov, Pedro Matias, and Valentin Polishchuk;
licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 51; pp. 51:1–51:21



Leibniz International Proceedings in Informatics

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

geometric coverage problem. All the mentioned problems share a property with agglomerative hierarchical clustering, which we call global-local equivalence, and which is the key to using the NNC algorithm. First, we review the NNC algorithm in its original context.

1.1 Prior work: NNC in hierarchical clustering

Given a set of points, the *agglomerative hierarchical clustering* problem is defined procedurally as follows: each point starts as a base cluster, and the two closest clusters are repeatedly merged until there is only one cluster left. This creates a *hierarchy*, where any two clusters are either nested or disjoint. A key component of hierarchical clustering is the function used to measure distances between clusters. Popular metrics include minimum distance (or single-linkage), maximum distance (or complete-linkage), and centroid distance.

We call two clusters *mutually nearest neighbors* (MNNs) if they are the nearest neighbor of each other. Consider this alternative, non-deterministic procedure: instead of repeatedly merging the two overall closest clusters, merge any pair of MNNs. The non-determinism comes from the fact that there can be multiple pairs of MNNs, and thus several valid choices. Clearly, this may merge clusters in a different order. Nonetheless, if the cluster-distance metric satisfies a property called *reducibility*, this procedure results in the same hierarchy [9, 10, 35]. A cluster-distance metric $d(\cdot, \cdot)$ is reducible if for any clusters A, B, C : if A and B are MNNs, then

$$d(A \cup B, C) \geq \min(d(A, C), d(B, C)). \quad (1)$$

In words, the new cluster $A \cup B$ resulting from merging A and B is *not* closer to other clusters than both A and B were. The relevance of this property is that, if, say, C and D are MNNs, merging A and B does not break that relationship. The net effect is that MNNs can be merged in any order and produce the same result. Many commonly used metrics are reducible, including minimum-, maximum-, and average-distance, but others such as centroid and median distance are not.

The NNC algorithm exploits this reducibility property, which was originally observed by Bruynooghe [10]. We briefly review the algorithm for hierarchical clustering, since we discuss it in detail later in the context of the new problems. For extra background on NNC for hierarchical clustering, see [36, 35]. The basic idea is to maintain a stack (called *chain*) of clusters. The first cluster is arbitrary. The chain is always extended with the nearest neighbor (NN) of the current cluster at the top of the chain. Note that the distance between clusters in the chain keeps decreasing, so (with an appropriate tie breaking rule) no repeated clusters or “cycles” occur, and the chain inevitably reaches a pair of MNNs. At this point, the MNNs are merged and removed from the chain. Crucially, after a merge happens, the rest of the chain is not discarded. Due to reducibility, every cluster in the chain still points to its NN, so the chain is still valid. The process continues from the new top of the chain.

The algorithm is efficient because each cluster is added to the chain only once, since it stays there until it is merged with another cluster. As we will see in detail for our problems, this bounds the number of iterations to be linear in the input size, with the cost of each iteration dominated by a NN computation.

Recently, the NNC algorithm was used for the first time outside of the domain of hierarchical clustering [23]. It was used in a stable matching problem where the two sets to be matched are point sets in a metric space, and each agent in one set ranks the agents in the other set by distance, with closer points being preferred. In this setting, there is an analogous phenomenon: the stable matching is unique, and it can be obtained in two ways; by repeatedly matching the closest pair (from different sets), or by repeatedly matching MNNs. They used the NNC algorithm to solve the problem efficiently.

1.2 Our Contributions

Our new observation is that this equivalence between merging closest pairs and MNNs is not unique to hierarchical clustering and stable matching: it also applies to the problems in this paper. We coin the term *global-local equivalence* for it. The main thesis of this paper is that NNC is an efficient algorithm for problems with global-local equivalence, which may include even more problems than those discussed here and in the full version of the paper.

Section 3 speeds up the multi-fragment heuristic for Euclidean TSP with a variant of NNC that uses a new data structure that we describe in Section 2, the *soft nearest-neighbor data structure*. We extend this result to Steiner TSP in Section 3.3. Section 4 is on motorcycle graphs. Each section contains background on the corresponding problems.

2 The Soft Nearest-Neighbor Data Structure

Throughout this section, we consider points in \mathbb{R}^δ , for some fixed dimension δ , and distances measured under any L_p metric $d(\cdot, \cdot)$. We begin with a formal definition of the structure and the main result of this section.

► **Definition 2.1** (Dynamic soft nearest-neighbor data structure). *Maintain a dynamic set of points, P , subject to insertions, deletions, and soft nearest-neighbor queries: given a query point q , return either of the following:*

- *The nearest neighbor of q in P : $p^* = \arg \min_{p \in P} d(q, p)$.*
- *A pair of points p, p' in P satisfying $d(p, p') < d(q, p^*)$.*

We make a general position assumption: the distances between q and the points in P are all unique. If this is not the case, the query point q can be perturbed slightly.

► **Theorem 2.2.** *In any fixed dimension, and for any L_p metric, there is a dynamic soft nearest-neighbor data structure that maintains a set of n points with $O(n \log n)$ preprocessing time and $O(\log n)$ time per operation (queries and updates).*

We label the two types of answers to soft nearest-neighbor (SNN) queries as *hard* or *soft*. A “standard” NN data structure is a special case of a SNN structure that always gives hard answers. However, in light of Theorem 2.2, a standard NN structure would not be as efficient as a SNN structure. For comparison, the best dynamic NN structure in \mathbb{R}^2 requires $O(\log^4 n)$ time per operation [12].

Given a point set P and a point q , let p_i^* denote the i -th closest point to q in P . In our implementation, we use the following data structure.

► **Definition 2.3** (Dynamic ε -approximate k nearest-neighbor (k -ANN) data structure). *Maintain a dynamic set of points, P , subject to insertions, deletions, and ε -approximate k nearest-neighbor queries: given a query point q and an integer k with $1 \leq k \leq |P|$, return k points $p_1, \dots, p_k \in P$ such that, for each p_i , $d(q, p_i) \leq (1 + \varepsilon)d(q, p_i^*)$, where $\varepsilon > 0$ is a constant known at construction time¹. We assume that p_1, \dots, p_k are sorted by non-decreasing distance from q .*

We reduce each SNN query to a single k -ANN query with constant ε and k . Once we show this reduction, Theorem 2.2 will follow from the following result by Arya et al. [4]:

¹ Some approximate nearest-neighbor data structures [4] do not need to know ε at construction time, and, in fact, allow ε to be part of the query and to be different for each query. Clearly, such data structures are also valid for our needs.

► **Lemma 2.4** ([4]). *In any fixed dimension, for any L_p metric, and for any constant $\varepsilon > 0$, there is a dynamic ε -approximate k nearest-neighbor data structure with $O(n \log n)$ preprocessing time and $O(\log n)$ time per operation (query and updates) for constant k .*

2.1 Soft nearest-neighbor implementation

We maintain the point set P in a dynamic k -ANN structure initialized with an approximation factor ε that will be determined later. The chosen ε will depend only on the metric space. In what follows, q denotes an arbitrary query point and p_i^* the i -th closest point to q in P . Using this notation, $p^* = p_1^*$. Recall the assumption that the points returned by a k -ANN structure are sorted by distance from the query point, so only the first one can be p^* . Queries rely on the following lemma.

► **Lemma 2.5.** *Let p_1, \dots, p_k be the answer given by a k -ANN structure, initialized with approximation factor ε , to a query (q, k) . If $p_1 \neq p^*$, then, for each p_i in p_1, \dots, p_k , $d(q, p_i) \leq (1 + \varepsilon)^{i-1} d(q, p_1)$.*

Proof. For $i = 1$, the claim is trivial. For $i = 2, \dots, k$, note that $d(q, p_i^*) \leq d(q, p_{i-1})$. This is because there are at least i points within distance $d(q, p_{i-1})$ of q : p^*, p_1, \dots, p_{i-1} . Thus, $d(q, p_i) \leq (1 + \varepsilon) d(q, p_i^*) \leq (1 + \varepsilon) d(q, p_{i-1})$. The claim follows by induction on i . ◀

Let $S(q, r_1, r_2)$ denote a closed shell centered at q with inner radius r_1 and outer radius r_2 (i.e., $S(q, r_1, r_2)$ is the difference between two balls centered at q , the bigger one of radius r_2 and the smaller one of radius r_1).

We call a pair (ε, k) *valid SNN parameters* (for a given metric space) if any set of k points inside a shell with inner radius 1 and outer radius $(1 + \varepsilon)^{k-1}$ contains two points, p and p' , satisfying $d(p, p') < 1/(1 + \varepsilon)$.

Suppose that (ε^*, k^*) are valid parameters. Initially, we construct the k -ANN structure using ε^* for the approximation factor. Then we answer queries as in Algorithm 1.

■ **Algorithm 1** Soft nearest-neighbor query.

```

Ask query  $(q, k^*)$  to the  $k$ -ANN structure initialized with  $\varepsilon^*$ .
Measure the distance between each pair of the  $k^*$  returned points,  $p_1, \dots, p_{k^*}$ .
if any pair  $(p_i, p_j)$  satisfies  $d(p_i, p_j) < d(q, p_1)/(1 + \varepsilon^*)$  then
    return  $p_i, p_j$ .
else
    return  $p_1$ .

```

► **Lemma 2.6.** *If (ε^*, k^*) are valid SNN parameters, Algorithm 1 is correct.*

Proof. The algorithm considers two cases. First, if a pair p_i, p_j of points returned by the k -ANN structure satisfies $d(p_i, p_j) < d(q, p_1)/(1 + \varepsilon^*)$, p_i and p_j are a valid soft answer to the query, as $d(q, p_1)/(1 + \varepsilon^*) \leq d(q, p^*)$.

In the alternative case, no pair among the returned points is at distance $< d(q, p_1)/(1 + \varepsilon^*)$. Consider the shell $S(q, d(q, p_1), (1 + \varepsilon^*)^{k^*-1} d(q, p_1))$. If we scale distances so that $d(q, p_1) = 1$, then this shell has inner radius 1 and outer radius $(1 + \varepsilon^*)^{k^*-1}$. Given that (ε^*, k^*) are valid SNN parameters, if all k^* of the returned points were inside this shell, at least two of them would be at a distance smaller than $1/(1 + \varepsilon^*)$. However, without the scaling, this distance

would be smaller than $d(q, p_1)/(1 + \varepsilon^*)$, which corresponds to the first case. Thus, at least one of the returned points lies outside the shell. In particular, the farthest point from q , p_{k^*} , satisfies $d(q, p_{k^*}) > (1 + \varepsilon^*)^{k^* - 1} d(q, p_1)$. By the contrapositive of Lemma 2.5, we have that $p^* = p_1$. ◀

As a side note, a SNN structure always returns a hard answer when queried from a point that is part of the closest pair of the set of points it maintains, as there is no *closer* pair. In this way, a SNN structure can be used to find the closest pair in (\mathbb{R}^δ, L_p) , for constant δ , in $O(n \log n)$ time by querying from every point. This matches the known runtimes in the literature [7].

We left open the issue of finding valid parameters (ε^*, k^*) , which we defer to Appendix A. In particular, it is not hard to see that they exist in any metric space (\mathbb{R}^δ, L_p) (see Lemma A.1).

3 Multi-Fragment Euclidean TSP

Given an undirected, complete graph G with uniquely and positively weighted edges, the *traveling salesperson problem* asks to find a cycle passing through all the nodes of minimum weight. In this section, we consider a classic greedy heuristic for constructing TSP tours, the *multi-fragment* algorithm.

Given two disjoint paths, p and p' , in G , we define the cost of connecting them, $cost(p, p')$, as the weight of the cheapest edge between an endpoint of p and an endpoint of p' . We use $p \cup p'$ to denote the path resulting from connecting p and p' into a single path along that edge.

The multi-fragment algorithm works as follows. A path generally has two endpoints, but, in this algorithm, each node starts as a single-node path. While there is more than one path, we connect the two paths such that the cost of connecting them is minimum. We call this the closest pair. Once there is a single path left, we connect its endpoints to complete the tour. We call the tour resulting from this process the *multi-fragment tour*, and the problem of constructing this tour *multi-fragment TSP*.

Euclidean TSP is the variant of TSP where the nodes are points in the plane, and the weights of the edges are the Euclidean distance between the points. Thus, here the goal is to find a closed polygonal chain, called a *tour*, through all the points of shortest length. The problem is NP-hard even in this geometric setting [27], but a polynomial-time approximation scheme is known [3].

The multi-fragment algorithm was proposed by Bentley [5] specifically in the geometric setting. Its approximation ratio is $O(\log n)$ [37, 8]. Nonetheless, it is used in practice due to its simplicity and empirical support that it generally performs better than other heuristics [19, 30, 33, 34, 6].

We are interested in the complexity of computing the multi-fragment tour in the geometric setting. A straightforward implementation of the multi-fragment algorithm is similar to Kruskal's minimum spanning tree algorithm: sort the $\binom{n}{2}$ pairs of points by increasing distances and process them in order: for each pair, if the two points are endpoints of separate paths, connect them. The runtime of this algorithm is $O(n^2 \log n)$. Eppstein [21] uses dynamic closest pair data structures to compute the multi-fragment tour in $O(n^2)$ time (for arbitrary distance matrices). Bentley [5] gives a K - d tree-based implementation and says that it appears to run in $O(n \log n)$ time on uniformly distributed points in the plane. We give a NNC-type algorithm that compute the multi-fragment tour for Euclidean TSP in $O(n \log n)$ in any fixed dimensions. We do not know of any prior worst-case subquadratic algorithm.

3.1 Global-local equivalence for multi-fragment TSP

In this section, we prove global–local equivalence for multi-fragment TSP in general graphs. That is, this result is not restricted to the Euclidean setting.

We say two paths are mutually nearest neighbors if the cost of connecting them is lower than the cost of connecting either with a third path. We consider an alternative algorithm to the multi-fragment algorithm, which connects MNNs paths instead of the closest pair. We use CP-MF to refer to the multi-fragment algorithm, and MNN-MF to refer to this new algorithm.

Clearly, CP-MF is a special case of MNN-MF. We show that any run of MNN-MF outputs the same solution as CP-MF.

Note the similarity between multi-fragment TSP and hierarchical clustering. Instead of merging clusters, we merge paths. The difference is that, in a cluster, it does not matter in which order the points were added when determining the distance to another cluster. In contrast, in a path, it is important which points are the endpoints.

We can see that in multi-fragment TSP we have a notion equivalent to reducibility in agglomerative hierarchical clustering (Equation 1).

► **Lemma 3.1** (Reducibility in multi-fragment TSP). *Let a, b , and c be paths in an undirected, complete graph G with positively weighted edges. Then, $\text{cost}(a \cup b, c) \geq \min(\text{cost}(a, c), \text{cost}(b, c))$.*

Proof. The cost of connecting paths is defined as the minimum weight among the edges connecting their endpoints. The claim is clear given that two endpoints of $a \cup b$ are a subset of the four endpoints of a and b . ◀

Given that we have reducibility, we can adapt the proof of global-local equivalence for agglomerative hierarchical clustering presented by Müllner [35] to multi-fragment TSP. The proof is in Appendix B.

► **Lemma 3.2** (Global-local equivalence in multi-fragment TSP). *Let G be an undirected, complete graph with uniquely and positively weighted edges. Then, CP-MF and MNN-MF output the same solution.*

3.2 Soft nearest-neighbor chain for multi-fragment Euclidean TSP

In general graphs, computing the multi-fragment tour with CP-MF requires $O(n^2 \log n)$ time, where the bottleneck is to sort the $\Theta(n^2)$ edges from cheapest to most expensive. Given that we have global-local equivalence (Lemma 3.2), this can be improved to $O(n^2)$ by implementing MNN-MF via a straightforward adaptation of the NNC algorithm. NNC builds a chain of paths in order to find MNN paths. We only need to spell out how to find the “nearest neighbor” of a path. We can find it in $O(n)$ time by scanning through the adjacency list of each endpoint. Thus, we can finish the $O(n)$ iterations of the algorithm in $O(n^2)$ time.

Similarly, in the geometric setting, for points in \mathbb{R}^2 , we can pair NNC with the dynamic NN data structure from [12], which runs in $O(\log^4 n)$ amortized time per operation. Thus, we can compute the multi-fragment tour in $O(n \log^4 n)$ time. However, we do not go into the details of these results and jump directly to our main result, which further improves the runtime in the geometric setting:

► **Theorem 3.3.** *The multi-fragment tour of a set of n points in any fixed dimension, and under any L_p metric, can be computed in $O(n \log n)$ time.*

We use a variation of the NNC algorithm that uses a SNN structure instead of the usual NN structure, which we call the *soft nearest-neighbor chain* algorithm (SNNC). For this, we need a SNN structure for paths in the plane (polygonal chains) instead of points. That is, a structure that maintains a set of (possibly single-point) paths, and, given a query path q , returns the closest path to q or two paths in the set which are closer to each other than q to its closest path in the set. The distance between paths is measured as the minimum distance between an endpoint of one and an endpoint of the other, so, for the purposes of this data structure, only the coordinates of the endpoints are important.

A soft nearest-neighbor structure for paths

We simulate a SNN structure for paths with a SNN structure for points. Given a set of paths, we maintain the set of path endpoints in the SNN structure for points. Updates are straightforward: we add or remove both endpoints of the path. Given a query path q with endpoints $\{q_1, q_2\}$, we do a SNN query from each endpoint of the path. If both answers are hard (assuming that the path has two distinct endpoints, otherwise, just the one), then we find the true NN of the path, and we can return it. However, there is a complication with soft answers: the two points returned could be the endpoints of the same path. Thus, it could be the case that we find two closer points, but not two closer paths, as we need. The solution is to modify the specification of the SNN structure for points so that soft answers, instead of returning two points closer to each other than the query point to its NN, return three pairwise closer points. We call this a *three-way* SNN structure. In the context of using the structure for paths, this guarantees that even if two of the three endpoints belong to the same path, at least two different paths are involved.

Lemma 3.4 shows how to obtain a three-way SNN structure for points, Algorithm 2 shows the full algorithm for answering SNN queries about paths using a three-way SNN structure for points, and Lemma 3.5 shows its correctness.

► **Lemma 3.4.** *In any fixed dimension and for any L_p metric, there is a three-way SNN structure with $O(n \log n)$ preprocessing time and $O(\log n)$ operation time (queries and updates).*

Proof. Recall the implementation of the SNN structure from Section 2. To obtain a three-way SNN structure, we need to change the values of ε and k to make the shell smaller and k bigger, so that if there are k points in a shell of inner radius 1 and outer radius $(1 + \varepsilon)^k$, then there must be at least three points at pairwise distance less than 1. The method described in Section A for finding valid parameters in (R^δ, L_p) also works here. It only needs to be modified so that the area (or surface) of the shell is accounted for twice. Since k and ε are still constant, this does not affect the asymptotic runtimes in Theorem 2.2. ◀

► **Lemma 3.5.** *In any fixed dimension, and for any L_p metric, we can maintain a set of n paths in a SNN structure for paths with $O(n \log n)$ preprocessing time and $O(\log n)$ operation time (queries and updates).*

Proof. All the runtimes follow from Lemma 3.4: we maintain the set of path endpoints in a three-way SNN structure S . The structure S can be initialized in $O(n \log n)$ time. Updates require two insertions or deletions to S , taking $O(\log n)$ time each. Algorithm 2 for queries clearly runs in $O(\log n)$ time. We argue that it returns a valid answer. Let q be a query path with endpoints $\{q_1, q_2\}$, and consider the three possible cases:

Algorithm 2 Soft-nearest-neighbor query for paths.

Let q_1 and q_2 be the endpoints of the query path, q .
 Let S be a three-way SNN structure containing the set of path endpoints.
 Query S with q_1 and q_2 .
if both answers are hard **then**
 Let p_1 and p_2 be the respective answers.
 return the closest path to the query path among the paths with endpoints p_1 and p_2 .
else if one answer is hard and the other is soft **then**
 Let p be the hard answer to q_1 and (a, b, c) the soft answer to q_2 (wlog). Let P and P'
 be the two closest paths among the paths with endpoints a, b , and c .
 if $d(p, q) < d(P, P')$ **then**
 return the path with endpoint p .
 else
 return (P, P') .
else (both answers are soft)
 Let (a_1, b_1, c_1) and (a_2, b_2, c_2) be the answers to q_1 and q_2 .
 return the closest pair of paths among the paths with endpoints $a_1, b_1, c_1, a_2, b_2, c_2$.

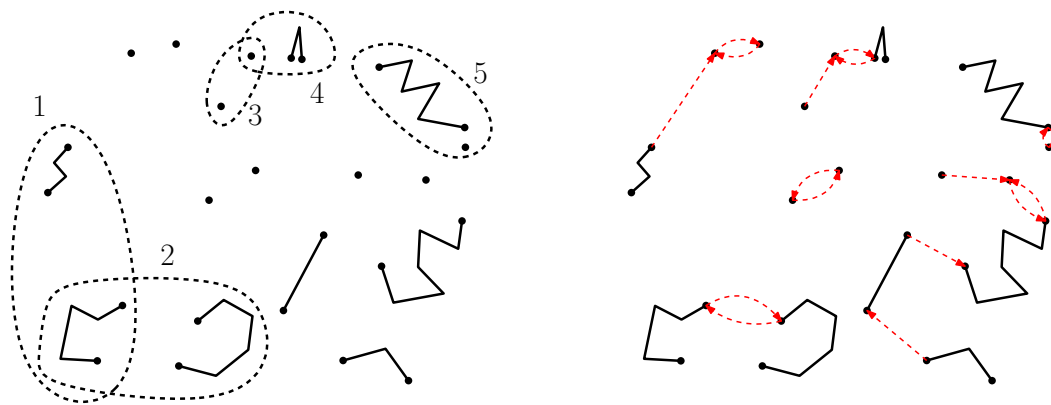
- Both answers are hard. In this case, we find the closest path to each endpoint, and, by definition, the closest of the two is the NN of q .
- One answer is soft and the other is hard. Let p be the hard answer to q_1 and (a, b, c) the soft answer to q_2 (wlog). Let P and P' be the two closest paths among the paths with endpoints a, b , and c . If $d(p, q) < d(P, P')$, then, the path with p as endpoint must be the NN of q , because there is no endpoint closer than $d(P, P')$ to q_2 . Otherwise, P, P' is a valid soft answer, as they are closer to each other than either endpoint of q to their closest endpoints.
- Both answers are soft. Assume (wlog) that the NN of q is closer to q_1 than q_2 . Then, the soft answer to q_1 gives us two paths closer to each other than q to its NN, so we return a valid soft answer. ◀

The soft nearest-neighbor chain algorithm

We use a SNN for paths (Lemma 3.5). In the context of this algorithm, let us think of a SNN answer, hard or soft, as being a set of two paths. If the answer is hard, then one of the paths returned in the answer is the query path itself, and the remaining path is its NN. Now, we can establish a comparison relationship between SNN answers (independently of their type): given two SNN answers $\{a, b\}$ and $\{c, d\}$, we say that $\{a, b\}$ is *better* than $\{c, d\}$ if and only if $d(a, b) < d(c, d)$.

The algorithm is given in Algorithm 3. The input is a set of points, which are interpreted as single-point paths and added to the SNN structure for paths. We assume unique distances between the points. The algorithm maintains a stack (the chain) of *nodes*, where each node consists of a pair of paths. In particular, each node in the chain is the best SNN answer among two queries for the two paths in the predecessor node (when querying from a path, we remove it from the structure temporarily, so that the answer is not the path itself).

Figure 1 shows a snapshot of the algorithm. Nodes 3 and 5 are soft answers, whereas nodes 2 and 4 are hard answers. The pair of paths in the fifth node are the overall closest pair, so the SNN structures will return that pair when queried from each of them. The algorithm will connect them, remove the fifth node from the chain, and continue from the fourth node.



■ **Figure 1** **Left:** a set of paths, including some single-point paths, and a possible chain, where the nodes are denoted by dashed lines and appear in the chain according to the numbering. **Right:** Nearest-neighbor graph of the set of paths. For each path, a dashed/red arrow points to its NN. Further, the arrows start and end at the endpoints determining the minimum distance between the paths.

■ **Algorithm 3** Soft nearest-neighbor chain algorithm for multi-fragment Euclidean TSP.

Initialize an empty stack (the chain).

Initialize a SNN structure S for paths (Lemma 3.5) with the set of input points as single-point paths.

while there is more than one path in S **do**

if the chain is empty **then**

 add a node containing an arbitrary pair of paths from S to it.

 Let $U = \{u, v\}$ be the node at the top of the chain.

 Remove u from S , query S with u , and re-add u to S .

 Remove v from S , query S with v , and re-add v to S .

 Let A be the best answer.

if $A \neq U$ **then**

 Add A to the chain.

else

 Remove u and v from S and add $u \cup v$.

 Remove U from the chain.

if the chain is not empty and the new last node, V , contains u or v **then**

 Remove V from the chain.

 Connect the two endpoints of the remaining path in S .

Correctness and runtime analysis

► **Lemma 3.6.** *The following invariants hold at the beginning of each iteration of Algorithm 3:*

1. *Each input point is an endpoint or a vertex of exactly one path in S .*
2. *If node R appears after node $\{s, t\}$ in the chain, then R is better than $\{s, t\}$.*
3. *Every path in S appears in at most two nodes in the chain, in which case they are consecutive nodes.*
4. *The chain only contains paths in S .*

Proof.

1. The claim holds initially. Each time two paths u, v are replaced by $u \cup v$, one endpoint of each becomes a vertex in the new path $u \cup v$, and the other endpoints become endpoints of $u \cup v$.
2. We show it for the specific case where R is immediately after $\{s, t\}$ in the chain, which suffices. Note that $R \neq \{s, t\}$, or it would not have been added to the chain. We distinguish between two cases:
 - s and t were MNNs when R was added. Then, R had to be a soft answer from s or t , which would have to be better than $\{s, t\}$.
 - s and t were not MNNs when R was added. Then, s had a closer path than t (wlog). Thus, whether the answer for s was soft or hard, the answer had to be better than $\{s, t\}$.
3. Assume, for a contradiction, that a path p appears in two non-consecutive nodes, $X = \{p, x\}$ and $Z = \{p, z\}$ (this covers the case where p appears more than twice). Let Y be the successor of X . By Invariant 2, Z is better than Y . It is easy to see that if z_1 and z_2 are the two endpoints of path z , then z_1 and z_2 were endpoints of paths since the beginning of the algorithm. Thus, the answer for p when X was at the top of the chain had to be a pair at distance at most $\min(d(p, z_1), d(p, z_2))$. However, $\min(d(p, z_1), d(p, z_2)) = d(p, z)$, contradicting that Z is better than Y .
4. Clearly, each node in the chain contains paths that were present in S at the time the node was added. Therefore, the invariant could only break when removing paths from S . In the algorithm, paths are removed from S when merging the paths in the top node. Thus, if a path p is removed from S , it means that p is in the top node. By Invariant 3, besides the top node, p can only occur in the second-from-top node. In the algorithm, when we merge the paths in the top node, we remove the top node from the chain, as well as its predecessor if has a path in common with the top node. ◀

► **Lemma 3.7.** *Paths connected in Algorithm 3 are MNNs in the set of paths in the SNN structure.*

Proof. Let $\{u, v\}$ be the node at the top of the chain at some iteration of Algorithm 3. Let A the best SNN answer among the queries from u and v . In the algorithm, u and v are connected when $A = \{u, v\}$. Thus, we need to show that if A is $\{u, v\}$, then u and v are MNNs. We show the contrapositive: if u and v are *not* MNNs, then A is not $\{u, v\}$. If u and v are not MNN, at least one of them, u (wlog), has a closer path than the other, v , so the answer for u is *better* than $\{u, v\}$. ◀

Proof of Theorem 3.3. We show that Algorithm 3 computes the multi-fragment tour in $O(n \log n)$ time.

For its correctness, note that the output is a single cycle that visits every vertex (Invariant 1). This cycle is constructed by only merging pairs of paths that are MNNs (Lemma 3.7). Thus, Algorithm 3 implements MNN-MF. By global-local equivalence (Lemma 3.2), this produces the multi-fragment tour.

For the runtime, note that the chain is acyclic in the sense that each node contains a path from the current set of paths in S (Invariant 4) not found in previous nodes (Invariant 3). Thus, the chain cannot grow indefinitely, so, eventually, paths get connected. The main loop does not halt until there is a single path.

If there are n points at the beginning, there are $n - 1$ connections between different paths in total, and $2n - 1$ different paths throughout the algorithm. This is because each connection removes two paths and adds one new path. At each iteration, either two paths are connected or one node is added to the chain. There are $n - 1$ iterations of the first kind,

each of which triggers the removal of one or two nodes in the chain. Thus, the total number of nodes removed from the chain is at most $2n - 2$. Since every node added is removed, the number of nodes added to the chain is also bounded by $2n - 2$. Thus, the total number of iterations of the second kind is also at most $2n - 2$, and the total number of iterations is at most $3n - 3$. Therefore, the total running time is $O(P(n) + nT(n))$, where $P(n)$ and $T(n)$ are the preprocessing and operation time of a SNN structure for paths. By Lemma 3.5, this can be done in $O(n \log n)$ time. ◀

3.3 Steiner TSP

In the traditional, non-Euclidean setting, a TSP instance consists of a complete graph with arbitrary distances. We remark that global-local equivalence (Lemma 3.2) still holds in this general setting. In this context, the nearest neighbor of a path can be found in $O(n)$ time by iterating through the adjacency lists of both endpoints, where n is the number of nodes. Using this linear search, we can easily compute the multi-fragment tour in $O(n^2)$ time with a NNC-based algorithm. It is a simpler version of Algorithm 3 that only has to handle hard answers and does not need any sophisticated data structures. This improves upon the natural way to implement the multi-fragment heuristic, which is to sort the $\Theta(n^2)$ edges by weight. Sorting requires $\Theta(n^2 \log n)$ time.

This is the first use of NNC in a graph-theoretical setting, but the fact of the matter is that the NNC algorithm can be used in any setting where we can find nearest neighbors efficiently. Consider the related Steiner TSP problem [16]: given a weighted, undirected graph and a set of k node sites $P \subseteq V$, find a minimum-weight tour (repeated vertices and edges allowed) in G that goes at least once through every site in P . Nodes not in P do not need to be visited. For instance, G could represent a road network, and the sites could represent the daily drop-off locations of a delivery truck.

Recently, Eppstein et al. [24] gave a NN structure for graphs from graph families with sublinear separators, which is the same as the class of graphs with polynomial expansion [18]. For instance, planar graphs have $O(\sqrt{n})$ -size separators². This data structure maintains a subset of nodes P of a graph G , and, given a query node q in G , returns the node in P closest to q . It allows insertions and deletions to and from the set P . We cite their result in Lemma 3.8.

▶ **Lemma 3.8** ([24]). *Given an n -node weighted graph from a graph family with separators of size $S(n) = n^c$, with $0 < c < 1$, which can be constructed in $O(n)$ time, there is a dynamic³ nearest-neighbor data structure requiring $O(nS(n))$ space and preprocessing time and that answers queries in $O(S(n))$ time and updates in $O(S(n) \log n)$ time.*

As mentioned, one way to implement the multi-fragment heuristic is to sort the $\binom{k}{2}$ pairs of sites by increasing distances, and process them in order: for each pair, if the two sites are endpoints of separate paths, connect them. The bottleneck is computing the distances. Running Dijkstra's algorithm from each site in a sparse graph, this takes $O(k(n \log n))$ (or $O(kn)$ in planar graphs [28]). When k is $\Theta(n)$, this becomes $O(n^2 \log n)$. We do not know of any prior faster algorithm to compute the multi-fragment tour for Steiner TSP.

² Other important families of sparse graphs with sublinear separators include k -planar graphs [17], minor-closed graph families [31], and graphs that model road networks (better than, e.g., k -planar graphs) [25].

³ They ([24]) use the term *reactive* for the data structure instead of dynamic, to distinguish from other types of updates, e.g., edge insertions and deletions.

Since we have global-local equivalence, we can use the NNC algorithm to construct the multi-fragment tour in $O(P(n) + kT(n))$ time, where $P(n)$ and $T(n)$ are the preprocessing and operation time of a nearest-neighbor structure. Thus, using the structure from [24], we get:

► **Theorem 3.9.** *The multi-fragment tour for the steiner TSP problem can be computed in $O(nS(n) + kS(n) \log n)$ -time in weighted graphs from a graph family with separators of size $S(n) = n^c$, with $0 < c < 1$.*

Finally, in graphs of bounded treewidth, which have separators of size $O(1)$, the data structure from [24] achieves $P(n) = O(n \log n)$ and $T(n) = O(\log^2 n)$, so we can construct a multi-fragment tour in $O(n \log n + k \log^2 n)$.

4 Motorcycle Graphs

An important concept in geometric computing is the straight skeleton [2]. It is a tree-like structure similar to the medial axis of a polygon, but which consists of straight segments only. Given a polygon, consider a shrinking process where each edge moves inward, at the same speed, in a direction perpendicular to itself. The straight skeleton of the polygon is the trace of the vertices through this process. Some of its applications include computing offset polygons [20] and medical imaging [15]. It is a standard tool in geometric computing software [11].

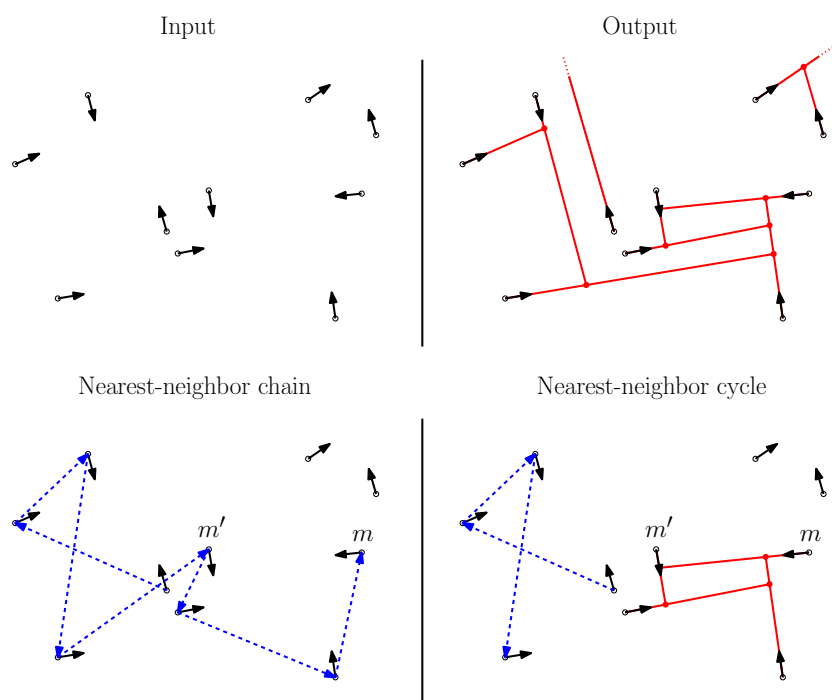
The fastest algorithms for computing straight skeletons consist of two phases [14, 29], neither of which dominates the other in computation time [13]. We focus on the first phase, which is a motorcycle graph computation induced by the reflex vertices of the polygon.

The motorcycle graph problem can be described as follows (see Figure 2, top) [20]. The input consists of n points in the plane, with associated directions and speeds (the motorcycles). Consider the process where all the motorcycles start moving at the same time, in their respective directions and speeds. Motorcycles leave a trace behind that acts as a “wall” such that other motorcycles crash and stop if they reach it. Some motorcycles escape to infinity while others crash against the traces of other motorcycles. The motorcycle graph is the set of traces.

Most existing algorithms rely on three-dimensional ray-shooting queries. This is because if time is seen as the third dimension, the position of a motorcycle starting to move from (x, y) , at speed s , in the direction (u, v) , forms a ray (if it escapes) or a segment (if it crashes) in three dimensions, starting at $(x, y, 0)$ in the direction $(u, v, 1/s)$. In particular, the impassable traces left behind by the motorcycles correspond to infinite vertical “curtains” – wedges or trapezoidal slabs, depending on whether they are bounded below by a ray or a segment.

Thus, ray-shooting queries help determine which trace a motorcycle would reach first, if any. Of course, the complication is that as motorcycles crash, their potential traces change. Early algorithms handle this issue by computing the crashes in chronological order [20, 14]. The best previously known algorithm, by Vigneron and Yan [39], is the first that computes the crashes in non-chronological order. It runs in $O(P(n) + n(T(n) + \log n) \log n)$ time, where $P(n)$ and $T(n)$ are the preprocessing time and operation time (maximum between query and update) of a dynamic ray-shooting data structure for curtains in \mathbb{R}^3 .

We propose a NNC-based algorithm which improves the number of ray-shooting operations needed from $O(n \log n)$ to $3n$. Besides the ray-shooting structure, Vigneron and Yan also use range searching data structures, which do not increase the asymptotic runtime but make the algorithm more complex. Our algorithm only uses a ray-shooting data structure.



■ **Figure 2 Top:** an instance input with uniform velocities and its corresponding motorcycle graph. **Bottom:** snapshots of the NNC algorithm before and after determining all the motorcycles in a NN cycle found by the chain: the NN of the motorcycle at the top, m , is m' , which is already in the chain. Note that some motorcycles in the chain have as NNs motorcycles against the traces of which they do not crash in the final output. That is expected, because these motorcycles are still undetermined (e.g., as a result of clipping the curtain of m' , the NN of its predecessor in the chain changes).

Agarwal and Matoušek [1] give a ray-shooting data structure for curtains in \mathbb{R}^3 which achieves $P(n) = O(n^{4/3+\varepsilon})$ and $T(n) = O(n^{1/3+\varepsilon})$ for any $\varepsilon > 0$. Using this structure, both our algorithm and the algorithm of Vigneron and Yan [39] run in $O(n^{4/3+\varepsilon})$ time for any $\varepsilon > 0$. However, if both algorithms use the same ε in the ray-shooting data structure, then our algorithm is asymptotically faster by a logarithmic factor.

4.1 Algorithm description

Initially, we label all motorcycles as *undetermined*, meaning that their final location is still unknown. They change to *determined* during the algorithm. We use a dynamic three-dimensional ray-shooting data structure to maintain a set of curtains, one for each motorcycle. Determined motorcycles have wedges or slabs as curtains, corresponding to their final trajectory. Undetermined motorcycles have wedge curtains as if they were to escape. When a motorcycle goes from undetermined to determined, the curtain is “clipped” from a wedge to a slab at the point where it crashes.

For an undetermined motorcycle m , we define its nearest neighbor to be the motorcycle, determined or not, against which m would crash next according to the set of curtains in the data structure. We can find the NN of a motorcycle m with one ray-shooting query. Motorcycles that escape may have no NN. Note that m may actually not crash against the trace of its NN, say m' , if m' is undetermined and happens to crash early.

Our main structure is a chain (a stack) of undetermined motorcycles. It starts (and restarts, if it becomes empty) with an arbitrary motorcycle, and it is repeatedly extended with the NN of the motorcycle m at the top of the chain, until one of the following cases is reached: (a) m has no NN (it escapes), (b) the NN of m is determined, or (c) the NN of m is already in the chain. In Case (a), we label m as determined and remove it from the chain. In Case (b), we clip m 's curtain, mark it as determined, and remove it and the previous motorcycle (if any) from the chain, as m may not be its NN anymore after the clipping.

In contrast to typical applications of the NNC algorithm, here “proximity” is not symmetric: there may be no “mutually nearest-neighbors”. In fact, the only case where two motorcycles are MNNs is the degenerate case where two motorcycles reach the same point simultaneously. That said, mutually nearest neighbors have an appropriate analogue in the asymmetric setting: *nearest-neighbor cycles*, $m_1 \rightarrow m_2 \rightarrow \dots \rightarrow m_k \rightarrow m_1$. Case (c) corresponds to finding such a cycle. If we find a nearest-neighbor cycle of undetermined motorcycles, then each motorcycle in the cycle crashes into the next motorcycle's trace. It is easy to see from our definition of nearest neighbors that no motorcycle outside the cycle would “interrupt” the cycle by making one of them crash early. Thus, in Case (c), we can determine all the motorcycles in the cycle at once (this can be seen as a type of chronological global-local equivalence; See Figure 2, bottom). We clip their curtains appropriately and we remove them and the prior motorcycle (if any) from the chain.

The process continues until all motorcycles are determined. Note that we only modify the curtain of the newly determined motorcycles. Thus, if at some step we determine the motorcycle (or motorcycles) at the top of the chain, only the NN of the would-be top motorcycle in the chain may have changed. This is why the would-be top motorcycle is removed from the chain in Cases (b) and (c). The rest of the chain remains consistent.

4.2 Analysis

Clearly, every motorcycle eventually becomes determined, and we have already argued in the algorithm description that irrespective of whether it becomes determined through Case (a), (b), or (c), its final position is correct. Thus, we move on to the complexity analysis. Each “clipping” update can be seen as an update to the ray-shooting data structure: we remove the wedge and add the slab.

► **Theorem 4.1.** *The NNC algorithm computes the motorcycle graph in time $O(P(n)+nT(n))$, where $P(n)$ and $T(n)$ are the preprocessing time and operation time (maximum between query and update) of a dynamic, three-dimensional ray-shooting data structure.*

Proof. Each step of the algorithm requires one ray-shooting query from the motorcycle at the top of the chain. At each iteration, either a motorcycle is added to the chain, or at least one motorcycle is determined.

Motorcycles begin as undetermined and, once they become determined, they remain so. This bounds the number of Cases (a–c) to n . In Cases (b) and (c), one undetermined motorcycle may be removed from the chain. Thus, the number of undetermined motorcycles removed from the chain is at most n . It follows that there are at most $2n$ iterations where a motorcycle is added to the chain.

Overall, the algorithm takes at most $3n$ iterations, so it needs no more than $3n$ ray-shooting queries and at most n “clipping” updates where we change a triangular curtain into a slab. It follows that the runtime is $O(P(n) + nT(n))$. ◀

See Appendix C for additional results in special cases and some remarks.

5 Conclusions

Before this paper, NNC had been used only in agglomerative hierarchical clustering and stable matching problems based on proximity. This paper adds: its first use without a NN structure (multi-fragment TSP, which uses our new soft NN structure); its first use in a graph-theoretical framework (Steiner TSP, Subsection 3.3); and its first use in problems without symmetric distances (motorcycle graphs, where it finds nearest-neighbor cycles).

The full version of the paper considers two additional problems [32]. The NNC technique is versatile enough that it seems likely that it will find more uses.

References

- 1 Pankaj K. Agarwal and Jiří Matoušek. Ray Shooting and Parametric Search. *SIAM Journal on Computing*, 22(4):794–806, 1993.
- 2 Oswin Aichholzer, Franz Aurenhammer, David Albers, and Bernd Gärtner. A novel type of skeleton for polygons. In *J. UCS The Journal of Universal Computer Science*, pages 752–761. Springer, 1996.
- 3 Sanjeev Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the ACM (JACM)*, 45(5):753–782, 1998.
- 4 Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM (JACM)*, 45(6):891–923, 1998.
- 5 Jon Jouis Bentley. Fast Algorithms for Geometric Traveling Salesman Problems. *ORSA Journal on Computing*, 4(4):387–411, 1992.
- 6 Jon Louis Bentley. Experiments on Traveling Salesman Heuristics. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '90*, pages 91–99, Philadelphia, PA, USA, 1990. Society for Industrial and Applied Mathematics.
- 7 Sergei N. Bespamyatnikh. An Optimal Algorithm for Closest-Pair Maintenance. *Discrete & Computational Geometry*, 19(2):175–195, February 1998.
- 8 Judith Brecklinghaus and Stefan Hougardy. The approximation ratio of the greedy algorithm for the metric traveling salesman problem. *Operations Research Letters*, 43(3):259–261, 2015.
- 9 Michel Bruynooghe. New methods in automatic classification of numerous taxonomic data. *Statistics and data analysis*, 2(3):24–42, 1977.
- 10 Michel Bruynooghe. Classification ascendante hiérarchique des grands ensembles de données: un algorithme rapide fondé sur la construction des voisinages réductibles. *Les cahiers de l'analyse de données*, 3:7–33, 1978.
- 11 Fernando Cacciola. A CGAL implementation of the Straight Skeleton of a Simple 2D Polygon with Holes. *2nd CGAL User Workshop*, January 2004. URL: http://www.cgal.org/UserWorkshop/2004/straight_skeleton.pdf.
- 12 Timothy M. Chan. Dynamic Geometric Data Structures via Shallow Cuttings. 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 24:1–24:13, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.SocG.2019.24.
- 13 Siu-Wing Cheng, Liam Mencil, and Antoine Vigneron. A Faster Algorithm for Computing Straight Skeletons. *ACM Trans. Algorithms*, 12(3):44:1–44:21, April 2016.
- 14 Siu-Wing Cheng and Antoine Vigneron. Motorcycle Graphs and Straight Skeletons. *Algorithmica*, 47(2):159–182, February 2007.
- 15 F. Cloppet, J. M. Oliva, and G. Stamon. Angular bisector network, a simplified generalized Voronoi diagram: application to processing complex intersections in biomedical images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):120–128, January 2000.

- 16 Gérard Cornuéjols, Jean Fonlupt, and Denis Naddef. The traveling salesman problem on a graph and some related integer polyhedra. *Mathematical Programming*, 33(1):1–27, September 1985.
- 17 Vida Dujmović, David Eppstein, and David R. Wood. Structure of graphs with locally restricted crossings. *SIAM J. Discrete Mathematics*, 31(2):805–824, 2017.
- 18 Zdeněk Dvořák and Sergey Norin. Strongly sublinear separators and polynomial expansion. *SIAM Journal on Discrete Mathematics*, 30(2):1095–1101, 2016.
- 19 Mehdi El Krari, Belaïd Ahiod, and Bouazza El Benani. An Empirical Study of the Multi-fragment Tour Construction Algorithm for the Travelling Salesman Problem. In Ajith Abraham, Abdelkrim Haqiq, Adel M. Alimi, Ghita Mezzour, Nizar Rokbani, and Azah Kamilah Muda, editors, *Proceedings of the 16th International Conference on Hybrid Intelligent Systems (HIS 2016)*, pages 278–287, Cham, 2017. Springer International Publishing.
- 20 D. Eppstein and J. Erickson. Raising Roofs, Crashing Cycles, and Playing Pool: Applications of a Data Structure for Finding Pairwise Interactions. *Discrete & Computational Geometry*, 22(4):569–592, December 1999.
- 21 David Eppstein. Fast hierarchical clustering and other applications of dynamic closest pairs. *Journal of Experimental Algorithmics (JEA)*, 5:1, 2000.
- 22 David Eppstein, Michael T. Goodrich, Doruk Korkmaz, and Nil Mamano. Defining equitable geographic districts in road networks via stable matching. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 52. ACM, 2017.
- 23 David Eppstein, Michael T. Goodrich, and Nil Mamano. Algorithms for stable matching and clustering in a grid. In *International Workshop on Combinatorial Image Analysis*, pages 117–131. Springer, 2017.
- 24 David Eppstein, Michael T. Goodrich, and Nil Mamano. Reactive Proximity Data Structures for Graphs. In Michael A. Bender, Martín Farach-Colton, and Miguel A. Mosteiro, editors, *LATIN 2018: Theoretical Informatics*, pages 777–789, Cham, 2018. Springer International Publishing.
- 25 David Eppstein and Siddharth Gupta. Crossing patterns in nonplanar road networks. In *25th ACM SIGSPATIAL Int. Conf. on Advances in Geographic Information Systems*, September 2017.
- 26 Michael T. Goodrich and Roberto Tamassia. Dynamic Ray Shooting and Shortest Paths via Balanced Geodesic Triangulations. In *Proceedings of the Ninth Annual Symposium on Computational Geometry, SCG '93*, pages 318–327, New York, NY, USA, 1993. ACM.
- 27 Christos H. Papadimitriou. The Euclidean traveling salesman problem is NP-complete. *Theoretical Computer Science*, 4:237–244, June 1977.
- 28 Monika R. Henzinger, Philip Klein, Satish Rao, and Sairam Subramanian. Faster shortest-path algorithms for planar graphs. *Journal of Computer and System Sciences*, 55(1):3–23, 1997.
- 29 Stefan Huber and Martin Held. A fast straight-skeleton algorithm based on generalized motorcycle graphs. *International Journal of Computational Geometry & Applications*, 22(05):471–498, 2012.
- 30 David S. Johnson and Lyle A. McGeoch. The traveling salesman problem: A case study in local optimization. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. John Wiley and Sons, Chichester, United Kingdom, 1997.
- 31 Ken-ichi Kawarabayashi and Bruce Reed. A separator theorem in minor-closed classes. In *51st IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 153–162, 2010.
- 32 Nil Mamano, Alon Efrat, David Eppstein, Daniel Frishberg, Michael Goodrich, Stephen Kobourov, Pedro Matias, and Valentin Polishchuk. Euclidean TSP, Motorcycle Graphs, and Other New Applications of Nearest-Neighbor Chains, 2019. [arXiv:1902.06875](https://arxiv.org/abs/1902.06875).
- 33 Alfonsas Misevicius and Andrius Blazinskas. Combining 2-OPT, 3-OPT and 4-OPT with K-Swap-Kick perturbations for the traveling salesman problem. *17th International Conference on Information and Software Technologies*, 2011.

- 34 Pablo Moscato and Michael G. Norman. On the Performance of Heuristics on Finite and Infinite Fractal Instances of the Euclidean Traveling Salesman Problem. *INFORMS Journal on Computing*, 10(2):121–132, 1998. doi:10.1287/ijoc.10.2.121.
- 35 Daniel Müllner. Modern hierarchical, agglomerative clustering algorithms. *arXiv e-prints*, September 2011. arXiv:1109.2378.
- 36 Fionn Murtagh. A survey of recent advances in hierarchical clustering algorithms. *The Computer Journal*, 26(4):354–359, 1983.
- 37 Hoon Liong Ong and J. B. Moore. Worst-case analysis of two travelling salesman heuristics. *Operations Research Letters*, 2(6):273–277, 1984.
- 38 Lloyd Shapley and Herbert Scarf. On cores and indivisibility. *Journal of mathematical economics*, 1(1):23–37, 1974.
- 39 Antoine Vigneron and Lie Yan. A Faster Algorithm for Computing Motorcycle Graphs. *Discrete Comput. Geom.*, 52(3):492–514, October 2014.

A Choice of Parameters

We left open the issue of finding valid SNN parameters. Recall that (ε, k) are valid if any set of k points inside a shell with inner radius 1 and outer radius $(1 + \varepsilon)^{k-1}$ contains two points, p and p' , satisfying $d(p, p') < 1/(1 + \varepsilon)$. To simplify the question, we can scale distances by $(1 + \varepsilon)$, so that the inner radius is $1 + \varepsilon$, the outer radius $(1 + \varepsilon)^k$, and the required distance between the two points is < 1 . As a further simplification, we can reduce the inner radius to be 1. This makes the shell grow, and thus, if (ε, k) are valid parameters with this change, then they are also valid under the original statement. Hence, to clarify, the goal of this section is to show how to find, for any metric space (\mathbb{R}^δ, L_p) , a pair of parameters (ε, k) such that any set of k points inside a shell with inner radius 1 and outer radius $(1 + \varepsilon)^k$ contains two points, p and p' , satisfying $d(p, p') < 1$.

This question is related to the *kissing number* of the metric space, which is the maximum number of points that can be on the surface of a unit sphere all at pairwise distance ≥ 1 . For instance, it is well known that the kissing number is 6 in (\mathbb{R}^2, L_2) and 12 in (\mathbb{R}^3, L_2) . It follows that, in (\mathbb{R}^2, L_2) , $(\varepsilon^* = 0, k^* = 7)$ are valid SNN parameters. Of course, we are interested in $\varepsilon^* > 0$. Thus, our question is more general in the sense that our points are not constrained to lie on a ball, but in a shell (and, to complicate things, the width of the shell, $(1 + \varepsilon)^k - 1$, depends on the number of points).

► **Lemma A.1.** *There are valid SNN parameters in any metric space (\mathbb{R}^δ, L_p) .*

Proof. Consider a shell with inner radius 1 and outer radius $1 + c$, for some constant $c > 0$. A set of points in the shell at pairwise distance ≥ 1 corresponds to a set of disjoint balls of radius $1/2$ centered inside the shell. Consider the volume of the intersection of the shell with such a ball. This volume is lower bounded by some constant, v , corresponding to the case where the ball is centered along the exterior boundary. Since the volume of the shell, v_s , is itself constant, the maximum number of disjoint balls of radius $1/2$ that fit in the shell is constant smaller than v_s/v . This is because no matter where the balls are placed, at least v volume of the shell is inside any one of them, so, if there are more than v_s/v balls, there must be some region in the shell inside at least two of them. This corresponds to two points at distance < 1 .

Set k to be v_s/v , and ε to be the constant such that $(1 + \varepsilon)^k = 1 + c$. Then, (ε, k) are valid parameters for (\mathbb{R}^δ, L_p) . ◀

51:18 New Applications of Nearest-Neighbor Chains

The dependency of k -ANN structures on $1/\varepsilon$ is typically severe. Thus, for practical purposes, one would like to find a valid pair of parameters with ε as big as possible. The dependency on k is usually negligible in comparison, and, in any case, k cannot be too large because the shell's width grows exponentially in k . Thus, we narrow the question to optimizing ε : what is the largest ε that is part of a pair of valid parameters?

We first address the case of (\mathbb{R}^2, L_2) , where we derive the optimal value for ε analytically. We then give a heuristic, numerical algorithm for general (\mathbb{R}^δ, L_p) spaces.

Parameters in (\mathbb{R}^2, L_2)

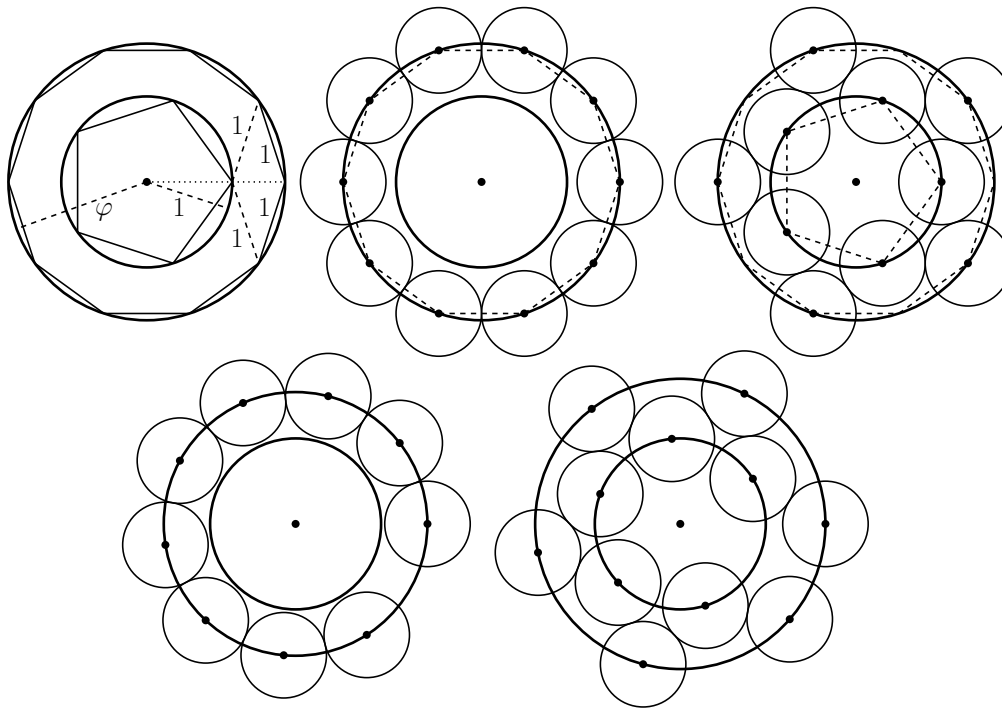
Let $\varepsilon_\varphi \approx 0.0492$ be the number such that $(1 + \varepsilon_\varphi)^{10} = \varphi$, where $\varphi = \frac{1+\sqrt{5}}{2}$ is the golden ratio. The valid SNN parameters with largest ε for (\mathbb{R}^2, L_2) are $(\varepsilon^* < \varepsilon_\varphi, k^* = 10)$ (ε^* can be arbitrarily close to ε_φ , but must be smaller). This follows from the following observations.

- The kissing number is 6, so there are no valid parameters with $k < 6$.
- The thinnest annulus (i.e., 2D shell) with inner radius 1 such that 10 points can be placed inside at pairwise distance ≥ 1 has outer radius $\varphi = (1 + \varepsilon_\varphi)^{10}$. Figure 3, top, illustrates this fact. In other words, if the outer radius is any smaller than φ , two of the 10 points would be at distance < 1 . Thus, any valid pair with $k = 10$ requires ε to be smaller than ε_φ , but any value smaller than ε_φ forms a valid pair with $k = 10$.
- For $6 \leq k < 10$ and for $k > 10$, it is possible to place k points at pairwise distance > 1 in an annulus of inner radius 1 and outer radius $(1 + \varepsilon_\varphi)^k$, and they are not packed “tightly”, in the sense that k points at pairwise distance > 1 can lie in a thinner annulus. This can be observed easily; Figure 3 (bottom) shows the cases for $k = 9$ and $k = 11$. Cases with $k < 9$ can be checked one by one; in cases with $k > 11$, the annulus grows at an increasingly faster rate, so placing k points at pairwise distance > 1 of each other becomes increasingly “easier”. Thus, for any $k \neq 10$, any valid pair with that specific k would require an ε smaller than ε_φ .

Parameters in (\mathbb{R}^δ, L_p)

For other L_p spaces, we suggest a numerical approach. We can do a binary search on the values of ε to find one close to optimal. For a given value of ε , we want to know if there is any k such that (ε, k) are valid. We can search for such a k iteratively, trying $k = 1, 2, \dots$ (the answer will certainly be “no” for any k smaller than the kissing number). Note that, for a fixed k , the shell has constant volume. As in Lemma A.1, let v be the volume of the intersection between the shell and a ball of radius $1/2$ centered on the exterior boundary of the shell. As argued before, if kv is bigger than the shell's volume, then (ε, k) are valid parameters. For the termination condition, note that if in the iterative search for k , k reaches a value where the volume of the shell grows more than v in a single iteration, no valid k for that ε will be found, as the shell grows faster than the new points cover it.

Besides the volume check, one should also consider a lower bound on how much of the shell's surface (both inner and outer) is contained inside an arbitrary ball. We can then see if, for a given k , the amount of surface contained inside the k balls is bigger than the total surface of the shell, at which point two balls surely intersect. This check finds better valid parameters than the volume one for relatively thin shells, where the balls “poke” out of the shell on both sides.



■ **Figure 3 Top:** The first figure shows two concentric circles of radius 1 and φ with an inscribed pentagon and decagon, respectively, and some proportions of these shapes. The other figures show two different ways to place 10 points at pairwise distance ≥ 1 inside an annulus of inner radius 1 and outer radius $(1 + \varepsilon_\varphi)^{10} = \varphi$. Disks of radius $1/2$ around each point are shown to be non-overlapping. In one case, the points are placed on the vertices of the decagon. In the other, they alternate between vertices of the decagon and the pentagon. In both cases, the distance between adjacent disks is 0. Thus, these packings are “tight”, i.e., if the annulus were any thinner, there would be two of the 10 points at distance < 1 . **Bottom:** 9 and 11 points at pairwise distance ≥ 1 inside annuli of radius $(1 + \varepsilon_\varphi)^9$ and $(1 + \varepsilon_\varphi)^{11}$, respectively. These packings are not tight, meaning that, for $k = 9$ and $k = 11$, a valid value of ε would have to be smaller than ε_φ .

B Proof of Global-Local Equivalence in Multi-Fragment TSP

In this section, we show the proof for Lemma 3.2. First we introduce another lemma.

► **Lemma B.1.** *Let G be an undirected, complete graph with uniquely and positively weighted edges. Then, any pair of paths p, p' that is or becomes MNNs during CP-MF remains MNNs until they are connected.*

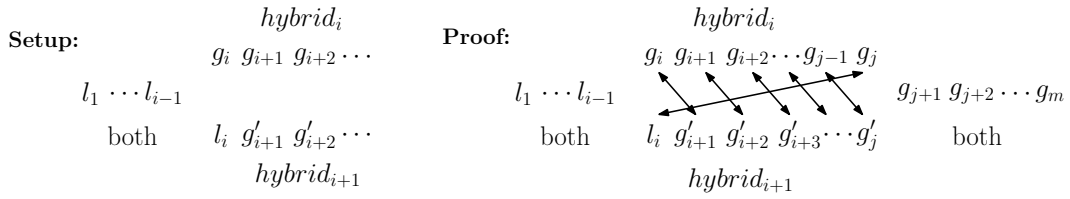
Proof. By definition of MNNs, it is cheaper to connect p and p' to each other than to any third path. By reducibility (Lemma 3.1), this does not change if CP-MF connects other paths, as the resulting path will be further from p and from p' than at least one of the preexisting paths before the merge. Eventually, p and p' become the overall cheapest pair to connect, and CP-MF connects them. ◀

Proof of Lemma 3.2. Let $L = l_1, l_2, \dots, l_m$ be the sequence of pairs of paths merged by a specific run of MNN-MF. Now, consider an algorithm that is a hybrid between MNN-MF and CP-MF. Let *hybrid_j* be an algorithm that starts merging the same pairs as in L up to and including l_{j-1} , and then switches to CP-MF, that is, it switches to merging closest pairs.

The main claim is that, no matter at which iteration the switch happens, the final solution is the same. In the extremes, if the switch happens before the first pick, the method is simply CP-MF. If the switch happens after the last pick, when no more elements can be picked, the solution is L . If we can prove the main claim, it follows that the solution of CP-MF is L .

Let $i \in \{1, \dots, m\}$. We show that the hybrid that switches before iteration i ($hybrid_i$) finds the same solution as the greedy that switches after iteration i ($hybrid_{i+1}$). The main claim follows by induction on i .

We label the pairs merged by $hybrid_i$ starting at iteration i with g_i, g_{i+1} , and so on. We label the pairs merged by $hybrid_{i+1}$ starting at iteration $i + 1$ with g'_{i+1}, g'_{i+2} , and so on. Figure 4, Left, shows this setup. Figure 4, Right, shows what actually happens with the paths merged by the two hybrid methods.



■ **Figure 4 Left:** the sequence of path pairs merged by $hybrid_i$ and $hybrid_{i+1}$. They agree up to and including iteration $i - 1$, and then $hybrid_i$ switches to CP-MF an iteration early. **Right:** what we prove about the pairs merged by the two hybrid methods. The arrows indicate that these pairs are actually the same. They coincide up to and including iteration $i - 1$ and after iteration j .

First, note that $hybrid_i$ eventually merges l_i . That is, $l_i = g_j$ for some $j \geq i$. This follows from Lemma B.1.

Further, $g'_{k+1} = g_k$ for all $k \in \{i, \dots, j - 1\}$. This can be shown by induction on k . By Lemma B.1 again, the path resulting from merging the paths in l_i is not closer to any of the paths involved in any of the pairs g'_{i+1}, \dots, g'_j than one of the two paths in l_i . It follows that “hoisting” the merge of l_i before them does not change the fact that g'_k is the closest pair at iteration k .

After iteration j , both $hybrid_i$ and $hybrid_{i+1}$ have merged exactly the same pairs, so they have the same partial solutions, and both are now the same algorithm, CP-MF. Thus, from that point on, they coincide in their merges, and finish with the same solution. ◀

C Motorcycle Graphs: Special Cases and Remarks

Consider the case where all motorcycles start from the boundary of a simple polygon with $O(n)$ vertices, move through the inside of the polygon, and also crash against the edges of the polygon. In this setting, the motorcycle trajectories form a connected planar subdivision. There are dynamic ray-shooting queries for connected planar subdivisions that achieve $T(n) = O(\log^2 n)$ [26]. Vigneron and Yan used this data structure in their algorithm to get a $O(n \log^3 n)$ -time algorithm for this case [39]. Our algorithm brings this down to $O(n \log^2 n)$. Furthermore, their other data structures require that coordinates have $O(\log n)$ bits, while we do not have this requirement.

Vigneron and Yan also consider the case where motorcycles can only go in C different directions. They show how to reduce $T(n)$ to $\min(O(C \log^2 n), C^2 \log n)$, leading to a $O(n \log^2 n C \min(\log n, C))$ algorithm for motorcycle graphs in this setting. Using the same data structures, the NNC algorithm improves the runtime to $O(n \log n C \min(\log n, C))$.

A remark on the use of our algorithm for computing straight skeletons: degenerate polygons where two shrinking reflex vertices collide gives rise to a motorcycle graph problem where two motorcycles collide head on. To compute the straight skeleton, a new motorcycle should emerge from the collision. Our algorithm does not work if new motorcycles are added dynamically (such a motorcycle could, e.g., disrupt a NN cycle already determined), so it cannot be used in the computation of straight skeletons of degenerate polygons.

As a side note, the NNC algorithm for motorcycle graphs is reminiscent of Gale's top trading cycle algorithm [38] from the field of economics. That algorithm also works by finding "first-choice" cycles. We are not aware of whether they use a NNC-type algorithm to find such cycles; if they do not, they certainly can; if they do, then at least our use is new in the context of motorcycle graphs.

Efficient Circuit Simulation in MapReduce

Fabian Frei

Department of Computer Science, ETH Zürich, Universitätstrasse 6, CH-8006 Zürich, Switzerland
fabian.frei@inf.ethz.ch

Koichi Wada

Department of Applied Informatics, Hosei University, 3-7-2 Kajino, 184-8584 Tokyo, Japan
wada@hosei.ac.jp

Abstract

The MapReduce framework has firmly established itself as one of the most widely used parallel computing platforms for processing big data on tera- and peta-byte scale. Approaching it from a theoretical standpoint has proved to be notoriously difficult, however. In continuation of Goodrich et al.'s early efforts, explicitly espousing the goal of putting the MapReduce framework on footing equal to that of long-established models such as the PRAM, we investigate the obvious complexity question of how the computational power of MapReduce algorithms compares to that of combinational Boolean circuits commonly used for parallel computations. Relying on the standard MapReduce model introduced by Karloff et al. a decade ago, we develop an intricate simulation technique to show that any problem in \mathcal{NC} (i.e., a problem solved by a logspace-uniform family of Boolean circuits of polynomial size and a depth polylogarithmic in the input size) can be solved by a MapReduce computation in $O(T(n)/\log n)$ rounds, where n is the input size and $T(n)$ is the depth of the witnessing circuit family. Thus, we are able to closely relate the standard, uniform \mathcal{NC} hierarchy modeling parallel computations to the deterministic MapReduce hierarchy \mathcal{DMRC} by proving that $\mathcal{NC}^{i+1} \subseteq \mathcal{DMRC}^i$ for all $i \in \mathbb{N}$. Besides the theoretical significance, this result has important applied aspects as well. In particular, we show for all problems in \mathcal{NC}^1 – many practically relevant ones, such as integer multiplication and division and the parity function, being among these – how to solve them in a constant number of deterministic MapReduce rounds.

2012 ACM Subject Classification Theory of computation → Complexity classes; Computing methodologies → MapReduce algorithms; Theory of computation → Circuit complexity; Theory of computation → MapReduce algorithms; Software and its engineering → Ultra-large-scale systems

Keywords and phrases MapReduce, Circuit Complexity, Parallel Algorithms, Nick's Class \mathcal{NC}

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.52

Related Version The full version of this paper including all figures and proofs is freely available at <http://arxiv.org/abs/1907.01624>.

Funding *Koichi Wada*: Research done in part during a supported visit at ETH Zürich and partly supported by JSPS KAKENHI No. 17K00019 and by the Japan Science and Technology Agency (JST) SICORP (Grant#JPMJSC1806).

Acknowledgements We thank the anonymous reviewers for their helpful comments.

1 Introduction

Despite the overwhelming success of the MapReduce framework in the big data industry and the great attention it has garnered ever since its inception over a decade ago, theoretical results about it have remained scarce in the literature. In particular, it is very natural to ask how powerful exactly MapReduce computations are in comparison to the traditional models of parallel computations based on circuits; a question that has practical implications as well. The answers have proved to be very elusive, however. In this paper, we show how MapReduce programs can efficiently simulate circuits used for parallel computations, thus tying these two worlds together more tightly.



© Fabian Frei and Koichi Wada;

licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 52; pp. 52:1–52:21

Leibniz International Proceedings in Informatics



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

In this section we first provide an introduction to the concept of MapReduce, then present the related work, and finally describe our contribution. In Section 2, we will formally define the traditional models of parallel computing and the MapReduce model. In Section 3, we then derive our main results. Section 4 concludes the paper with a short summary and a discussion of our findings, outlining opportunities for future research.

1.1 Background and Motivation

In recent years the amount of data available and demanding analysis has experienced an astonishing growth. The amount of memory in commercially available servers has also grown at a remarkable pace over the past decade; it is now exceeding tera- and even peta-bytes. Despite the considerable advances in the availability of computational power, traditional approaches remain insufficient to cope with such huge amounts of data. A new form of parallel computing has become necessary to deal with these enormous quantities of available data. The MapReduce framework has been attracting great interest due to its suitability for processing massive data-sets. This framework was originally developed by Google [5], but an open source implementation called Hadoop has recently been developed and is currently used by over a hundred companies, including Yahoo!, Facebook, Adobe, and IBM [19].

MapReduce differs substantially from previous models of parallel computation in that it combines aspects of both parallel and sequential computation. Informally, a MapReduce computation can be described as follows.

The input is a multiset of *key-value pairs* $\langle k; v \rangle$. In a first step, the *map step*, each of these key-value pairs is separately and independently transformed into an entire multiset of key-value pairs by a *map function* μ . In the next step, the *shuffle step*, we collect all key-value pairs from the multisets that have been produced in the previous step, group them by their keys, and collapse each group $\{\langle k; v_1 \rangle, \langle k; v_2 \rangle, \dots\}$ of pairs containing the same key into a single key-value pair $\langle k; \{v_1, v_2, \dots\} \rangle$ consisting of said key and a list of the associated values. In a third step, the *reduce step*, a *reduce function* ρ transforms the list of values in each key-value pair $\langle k; \{v_1, v_2, \dots\} \rangle$ into a new list $\{v'_1, v'_2, \dots\}$. Again, this is done separately and independently for each pair. The final output consists of the pairs $\{\langle k; v'_1 \rangle, \langle k; v'_2 \rangle, \dots\}$ for each key k . The different instances that implement the reduce function for the different groups of pairs are called reducers. Analogously, mappers are instances of the map function.

The three steps described above constitute one *round* of the MapReduce computation and transform the input multiset into a new multiset of key-value pairs. A complete MapReduce computation consists of any given number of rounds and acts just as the composition of the single rounds. The shuffle step works the same way every time; the map and reduce functions, however, may change from round to round. A MapReduce computation with R rounds is therefore completely described by a list $\mu_1, \rho_1, \mu_2, \rho_2, \dots, \mu_R, \rho_R$ of map and reduce functions. In both the map step and the reduce step, the input pairs can be processed in parallel since the map and reduce functions act independently on the pairs and groups of pairs, respectively. These steps therefore capture the parallel aspect of a MapReduce computation, whereas the shuffle step enforces a partial sequentiality since the shuffled pairs can be output only once the previous map step is completed in its entirety.

The MapReduce paradigm has been introduced in [5] in the context of algorithm design and analysis. A treatment as a formal computational model, however, was missing in the beginning. Later on, a number of models have emerged to deal more rigorously with algorithmic issues [7, 10, 12, 14, 15]. In this paper, our interest lies in studying the MapReduce framework from a standpoint of parallel algorithmic power by comparing it to standard models of parallel computation such as Boolean circuits and parallel random access machines

(PRAMs). A PRAM can be classified by how far simultaneous access by processors to its memory is restricted; it can be CRCW, EREW, CREW, or ERCW, where R, W, C, and E stand for Read, Write, Concurrent, and Exclusive, respectively [4]. If concurrent writing is allowed, we need to further specify how parallel writes by multiple processors to a single memory cell are handled. The most natural choice is arguably that every memory cell contains after each time step the total of all numbers assigned to it by different processors during that step. In fact, all constructions in this paper work with this treatment of simultaneous writes; we thus generally assume this model. If the context warrants it, we speak of a Sum-CRCW to make this assumption explicit.

1.2 Related Work

We briefly present and discuss the following known results on the comparative power of the MapReduce framework and PRAM models.

1. A T -time EREW-PRAM algorithm can be simulated by an $O(T)$ -round MapReduce algorithm, where each reducer uses memory of constant size and an aggregate memory proportional to the amount of shared memory required by the PRAM algorithm [10, 12].
2. A P -processor, M -memory, T -time EREW-PRAM algorithm can be simulated by an $O(T)$ -round, $(P + M)$ -key MUD algorithm with a communication complexity of $O(\log(P + M))$ bits per key, where a MUD (massive, unordered, distributed) algorithm is a data-streaming MapReduce algorithm in the following sense: The reducers do not receive the entire list of values associated with a given key at once, but rather as a stream to be processed in one pass, using only a small working memory determining the communication complexity [7].
3. When using MapReduce computations to simulate a CRCW-PRAM instead, again with P processors and M memory, we incur an $O(\log_m(P + M))$ slowdown compared to the simulations above, where m is an upper bound on each reducer's input and output [10].

These results imply that any problem solved by a PRAM with a polynomial number of processors and in polylogarithmic time T can be simulated by a MapReduce computation with an amount of memory equal to the number of PRAM processors, and in a number of rounds equal to the computation time of even the powerful CRCW-PRAM. Since the class of problems solved by CRCW-PRAMs in time $T \in O(\log^i n)$ is equal to the class of problems solved by families of polynomial-sized combinational circuits consisting of gates with unbounded fan-in and fan-out and depth $T \in O(\log^i n)$ (often denoted \mathcal{AC}^i) [1], these circuits can be simulated in a MapReduce computation with a number of rounds equal to the time required by these circuits.

Since the publication of the seminal paper by Karloff et al. [12], extensive effort has been spent on developing efficient algorithms in MapReduce-like frameworks [3, 6, 13, 11, 17]. Only few relationships between the theoretical MapReduce model by [12] and classical complexity classes have been established, however; for example, any problem in $SPACE(o(\log n))$ can be solved by a MapReduce computation with a constant number of rounds [8].

Recently, Roughgarden et al. [16, Theorem 6.1] described a short and simple way of simulating \mathcal{NC}^1 circuits with a certain class of models of parallel computation. The constraints of these models, namely the number of machines and the memory restrictions, are exactly tailored to allow for this general simulation method, however. In particular, it crucially relies on the fact that all models of this class are more powerful than the MapReduce model in that they all grant us a number of machines that is polynomial in the input size; this makes it possible to just dedicate one machine to each of the circuit gates. Such a simple simulation is impossible with MapReduce computations since the standard model due to Karloff only allows for a sublinear number of machines with sublinear memory.

1.3 Contribution

We prove that $\mathcal{NC}^{i+1} \subseteq \mathcal{DMRC}^i$ for all $i \in \{0, 1, 2, \dots\}$, where \mathcal{DMRC}^i is the set of problems solvable by a deterministic MapReduce computation in $O(\log^i n)$ rounds. In the case of $\mathcal{NC}^1 \subseteq \mathcal{DMRC}^0$, which already opens up a plethora of applications on its own, the result holds for every possible choice of ε , that is, for $0 < \varepsilon \leq 1/2$. The higher levels of the hierarchy require an entirely different proof method, which yields the result for $0 < \varepsilon < 1/2$.

This is a substantial improvement over the previous results that only imply, as outlined above, the far weaker claim $\mathcal{AC}^i \subseteq \mathcal{MRC}^i$. The case $i = 1$ is of particular practical interest since $\mathcal{NC}^1 \setminus \mathcal{AC}^0$ contains plenty of relevant problems such as integer multiplication and division, the parity function, and the recognition of Dyck languages; see [1]. Our results show how to solve all of these problems with a deterministic MapReduce program in a constant number of rounds.

2 Preliminaries

We denote by $\mathbb{N} = \{0, 1, 2, \dots\}$ the natural numbers including zero and let $\mathbb{N}_+ = \mathbb{N} \setminus \{0\}$. Moreover, we let $[i] = \{0, 1, \dots, i-1\}$ denote the i first natural numbers for any $i \in \mathbb{N}_+$.

2.1 Models of Parallel Computation

In this section, we define the common complexity classes capturing the power of parallel computation; most prominently the \mathcal{NC} hierarchy.

A finite set $\mathcal{B} = \{f_0, \dots, f_{|\mathcal{B}|-1}\}$ of Boolean functions $f_i : \{0, 1\}^{n_i} \rightarrow \{0, 1\}$ with $n_i \in \mathbb{N}$ for every $i \in [|\mathcal{B}|]$ is called a *basis*. For every $n, m \in \mathbb{N}_+$, a (*Boolean*) *circuit* C over the basis \mathcal{B} with n inputs and m outputs is a directed acyclic graph that contains n *sources* (nodes with no incoming edges), called the *input nodes*, and m *sinks* (nodes with no outgoing edges). The *fan-in* of a node is the number of incoming edges, the *fan-out* is the number of outgoing edges. Nodes that are neither sources nor sinks are called *gates*. Each gate is labeled with a function $f_i \in \mathcal{B}$ and has fan-in n_i . It computes f_i on the input given by the incoming edges and outputs the result (either 0 or 1) to each of the outgoing edges. A basis \mathcal{B} is said to be *complete* if for every Boolean function f , we can construct over the basis \mathcal{B} a circuit of the described form that computes f . In the following, we use the complete basis $\mathcal{B} = \{\vee, \wedge, \neg\}$.

The *size* of a circuit C , denoted by $\text{size}(C)$, is the total number of edges it contains. The *level* of a node v in a circuit C , denoted $\text{level}(v)$, is defined recursively: The level of a sink is 0, and the level of a node v with nonzero fan-out is one greater than the maximum of the levels of the outgoing neighbors of v . The *depth* of C , denoted $\text{depth}(C)$, is the maximum level across all nodes in C . A function $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ is *implicitly logspace computable* if the two mappings $(x, i) \mapsto \chi_{i \leq |f(x)|}$, where χ denotes the characteristic function, and $(x, i) \mapsto (f(x))_i$ are computable using logarithmic space. A circuit family $\{C_n\}_{n=0}^\infty$ is *logspace-uniform* if there is an implicitly logspace computable function mapping 1^n to the description of the circuit C_n . It is known that the class of languages that have logspace-uniform circuits of polynomial size equals \mathcal{P} [1, Thm. 6.15].

For any $i \in \mathbb{N}$, the complexity class \mathcal{NC}^i contains a language L exactly if there is a constant c and a logspace-uniform family of circuits $\{C_n\}_{n=0}^\infty$ recognizing L such that C_n has size $O(n^c)$, depth $O(\log^i n)$, and all nodes have fan-in at most 2. The union is Nick's class $\mathcal{NC} = \bigcup_{i=0}^\infty \mathcal{NC}^i$. We mention that there is an analogous definition of classes Nonuniform- \mathcal{NC}^i that do not require logspace uniformity from the circuits; they constitute a different hierarchy.

The complexity classes \mathcal{AC}^i and $\mathcal{AC} = \bigcup_{i=0}^{\infty} \mathcal{AC}^i$ are defined exactly as \mathcal{NC}^i and \mathcal{NC} , except that the restriction of the maximal fan-in to at most 2 is omitted. Nevertheless, the restriction on the circuit size imply that the fan-in of a node is bounded by a polynomial in n . The OR gates and AND gates in such a circuit can therefore be replaced by trees of gates of fan-in at most 2 with a depth in $O(\log n)$. It follows that $\mathcal{AC}^i \subseteq \mathcal{NC}^{i+1}$ for all $i \in \mathbb{N}$ and thus $\mathcal{NC} = \mathcal{AC}$. (Analogously, we see why Nick's class can also be defined, as it often is, by upper-bounding the fan-in by an arbitrary constant greater than 2.) The inclusion $\mathcal{NC}^i \subseteq \mathcal{AC}^i$ for every $i \in \mathbb{N}$ is immediate from the definition. The first two inclusions of the resulting chain are known to be strict – namely, we have $\mathcal{NC}^0 \subsetneq \mathcal{AC}^0 \subsetneq \mathcal{NC}^1$; see [1].

Finally, we summarize the known results on how the classes of languages recognized by different PRAMs fit into the two hierarchies of \mathcal{NC} and \mathcal{AC} . Let \mathcal{EREW}^i , \mathcal{CREW}^i and \mathcal{CRCW}^i denote the sets of problems of size n computed by EREW-PRAMs, CREW-PRAMs, and CRCW-PRAMs, respectively, with a polynomial number of processors in $O(\log^i n)$ time. For every $i \in \mathbb{N}$, we have $\mathcal{NC}^i \subseteq \mathcal{EREW}^i \subseteq \mathcal{CREW}^i \subseteq \mathcal{CRCW}^i = \mathcal{AC}^i \subseteq \mathcal{NC}^{i+1}$; see [1].

2.2 The MapReduce Model

In this section we describe the standard MapReduce model as proposed by [12]. It defines the notions of *map functions* and *reduce functions*, which are summarized under the term *primitives*. Roughly speaking, a MapReduce computing system executes primitives, interleaved with so-called *shuffle* operations. The basic data unit in these computations is an ordered pair $\langle \text{key}; \text{value} \rangle$, called *key-value pair*. In general, keys and values are just binary strings, allowing us to encode all the usual entities.

A map function is a (possibly randomized) function that takes as input a single key-value pair and outputs a finite multiset of new key-value pairs. A reduce function (again, possibly randomized) takes instead an entire set of key-value pairs $\{\langle k; v_{k,1} \rangle, \langle k; v_{k,2} \rangle, \dots\}$, where all the keys are identical, and outputs a single key-value pair $\langle k; v' \rangle$ with that same key.

A MapReduce program is nothing else than a sequence $\mu_1, \rho_1, \mu_2, \rho_2, \dots, \mu_R, \rho_R$ of map functions μ_r and reduce functions ρ_r . The input of this program is a multiset U_0 of key-value pairs. For each $r \in \{1, \dots, R\}$, a map step, a shuffle step and a reduce step are successively executed as follows:

1. **Map step:** Each pair $\langle k; v \rangle$ in U_{r-1} is given as input to an arbitrary instance of the map function μ_r , which then produces a finite sequence of pairs. The multiset of all produced pairs is denoted by V_r .
2. **Shuffle step:** For each key k , let $V_{k,r}$ be the multiset of all values v_i such that $\langle k, v_i \rangle$. The MapReduce system automatically constructs the multiset $V_{k,r}$ from V_r in the background.
3. **Reduce step:** For each key k , a reducer (i.e., an instance calculating the reduce function ρ_r) receives k and the elements of $V_{k,r}$ in arbitrary order. We usually write such an input as a set of key-value pairs that all have key k . The reducer calculates, for each key k independently, from $V_{k,r}$ a set $U_{k,r}$ of key-value pairs. The output will then consist of all key-value pairs computed in this reduce step; that is, U_r is the union over all sets $U_{k,r}$.

Fix any ε with $0 < \varepsilon \leq 1/2$ and denote the size of the MapReduce program's input by N . For every $i \in \mathbb{N}$, a problem is in \mathcal{MRC}^i if and only if there is a MapReduce program $\mu_1, \rho_1, \mu_2, \rho_2, \dots, \mu_R, \rho_R$ satisfying the following properties:

1. It outputs a correct answer to the problem with probability at least $3/4$.
2. The number of rounds of the MapReduce program, R , is in $O(\log^i N)$.
3. The potentially randomized primitives (i.e., all map and reduce functions) are computable by a RAM with $O(\log N)$ -bit words using $O(N^{1-\varepsilon})$ space and time polynomial in N .
4. The pairs produced by the map functions can be stored in $O(N^{2(1-\varepsilon)})$ space.

A MapReduce program satisfying these conditions is called an MRC^i -algorithm. Note that due to the last condition it is impossible to even store the input unless $2(1 - \varepsilon) \geq 1$, which explains the restriction $0 < \varepsilon \leq 1/2$. As with \mathcal{NC} , we define the union class $MRC = \bigcup_{i=0}^{\infty} MRC^i$. Requiring all primitives to be deterministic yields the analogous hierarchy of $DMRC = \bigcup_{i=0}^{\infty} DMRC^i$. Note that we obviously have $DMRC^i \subseteq MRC^i$ for all $i \in \mathbb{N}$. We will often refer to the single rounds of such MapReduce algorithms as MRC -rounds and $DMRC$ -rounds, respectively.

3 Simulating Parallel Computations in MapReduce

We are now going to prove our two main results $\mathcal{NC}^1 \subseteq DMRC^0$ for $0 < \varepsilon \leq 1/2$ and $\mathcal{NC}^{i+1} \subseteq DMRC^i$ for all $i \in \mathbb{N}_+$ and $0 < \varepsilon < 1/2$ in Sections 3.2 and 3.3, respectively. In both cases, we will be making use of a technical tool derived in Section 3.1 and obtain the results by showing how to use MapReduce computations for two different, delicate simulations. For the inclusion $\mathcal{NC}^1 \subseteq DMRC^0$, we simulate width-bounded branching programs that are equivalent to the respective circuits by Barrington's classical theorem [2], whereas for the higher levels of the hierarchy, we directly simulate the combinational circuits themselves.

3.1 A Technical Tool

Goodrich et al. [10] parametrize MapReduce algorithms, on the one hand, by the memory limit m for the input/output buffer of the reducers and, on the other hand, by the *communication complexity* K_r of round r , that is, the total size of inputs and outputs for all mappers and reducers in round r . We state a useful result from [10].

► **Theorem 1.** *Any CRCW-PRAM algorithm using M total memory, P processors and T time can be simulated in $O(T \log_m P)$ deterministic MapReduce-rounds with communication complexity $K_r \in O((M + P) \log_m(M + P))$.*

We denote by N the size of the smallest circuit representation of the CRCW-PRAM algorithm (i.e., its number of edges) plus the size of its input. Taking into account our requirements $m \in O(N^{1-\varepsilon})$ and $K_r \in O(N^{2(1-\varepsilon)})$, we obtain the following a technical tool, which will prove to be useful in our endeavor.

► **Corollary 2.** *Any CRCW-PRAM algorithm using M total memory, P processors and T time can be simulated in $O(T \log_{N^{1-\varepsilon}} P)$ $DMRC$ -rounds if $(M + P) \log_{N^{1-\varepsilon}}(M + P) \in O(N^{2(1-\varepsilon)})$.*

3.2 Simulating \mathcal{NC}^1

It is known that Nonuniform- \mathcal{NC}^1 is equal to the class of languages recognized by nonuniform width-bounded branching programs. A careful inspection of the proof due to Barrington [2] – crucially relying on the non-solvability of the permutation group on 5 elements – reveals that it naturally translates to the uniform analogue: Our uniform class \mathcal{NC}^1 is identical with the class of languages recognized by uniform width-bounded branching programs. In order to prove $\mathcal{NC}^1 \subseteq DMRC^0$, it therefore suffices to show how to simulate such branching programs by appropriate MapReduce computations with a constant number of rounds.

We first define what a width-bounded branching program is. Let $n, w \in \mathbb{N}_+$. The input to the program is an assignment α to n Boolean variables $\mathcal{X} = \{x_0, \dots, x_{n-1}\}$. An *instruction* or *line* of the program is a triple (x_i, f, g) , where i is the *index* of an input variable $x_i \in \mathcal{X}$ and f and g are endomorphisms of $[w]$. An instruction (x_i, f, g) *evaluates* to f if $\alpha(x_i) = 1$ and to g if $\alpha(x_i) = 0$. A *width- w branching program* of length t is a sequence of instructions

(x_{i_j}, f_j, g_j) for $j \in [t]$. We also refer to the t instructions as the lines of the program. Given an assignment α to \mathcal{X} , a branching program B yields a function $B(\alpha)$ that is the composition of the functions to which the instructions evaluate.

To recognize a language $L \subseteq \{0, 1\}^*$, we need a family $(B_n)_{n=0}^\infty$ of width- w branching programs with B_n taking n Boolean inputs. We say that L is recognized by B_n if there is, for each $n \in \mathbb{N}$, a set F_n of endomorphisms of $[w]$ such that for all $\alpha \in \{0, 1\}^n$, $\alpha \in L$ if and only if $B_n(\alpha) \in F_n$. If f_i and g_i are automorphisms, that is, permutations of $[w]$ for all $i \in [t]$, then B_n is called a *width- w permutation branching program*, or *w-PBP* for short.

► **Theorem 3** ([2]). *If $L \in \mathcal{NC}^1$, then L is recognized by a logspace-uniform 5-PBP family.*

Due to Theorem 3 it is sufficient for our purposes to simulate the w -PBPs with constant w instead of the circuit families provided by the definition of \mathcal{NC}^1 . In order to do this, we need to encode the given w -PBP and the possible assignments in the right form, namely we express them as sets of key-value pairs. A w -PBP of length t can be described as the set $\{\langle p; (x_{i_p}, f_p, g_p) \rangle \mid p \in [t]\}$, where we call p the *line number* of line (x_{i_p}, f_p, g_p) . Similarly, an assignment $\alpha: \mathcal{X} \rightarrow \{0, 1\}, x_i \mapsto v_i$ to the input variables $\mathcal{X} = \{x_0, x_1, \dots, x_{n-1}\}$ is described by the set of key-value pairs $\{\langle i; (x_i, v_i) \rangle \mid i \in [n]\}$, letting the mappers divide the information by the indices of the input variables. Let N_O and N_I be the total size of the encodings of the w -PBP and the input assignment α , respectively. Let $N = N_O + N_I$ and let $d = \lceil N_O^{1-\epsilon} \rceil$ and $\ell = \lceil N_O^\epsilon \rceil$. We denote by \div the integer division. For every $q \in [t \div d]$, let w -PBP $_q$ be the q th of the subprogram blocks of w -PBP of length d , that is $\{\langle p; (x_{i_p}, f_p, g_p) \rangle \mid qd \leq p \leq (q+1)d - 1\}$. For ease of readability, we assume from now on without loss of generality that $d\ell = t$, so that w -PBP can be partitioned into exactly ℓ such subprograms.

For every $q \in [\ell]$, we denote by \mathcal{X}_q the subset of variables from \mathcal{X} appearing in the instructions of subprogram w -PBP $_q$. An assignment $\alpha_q: \mathcal{X}_q \rightarrow \{0, 1\}$ to these variables is represented as a set of key-value pairs in the following way. Recall that the subprogram w -PBP $_q$ is a list of lines, each of which requires the assignment of a value, either 0 or 1, for exactly one variable. Let $x_{q,j}$ be the j th variable to which a value is assigned in w -PBP $_q$, let $p_{q,j}$ denote the number of the line in which this assignment occurs for the first time in w -PBP $_q$, and let $v_{q,j}$ denote the value that is assigned to $x_{q,j}$ in this line. Now, we represent α_q by $\{\langle q; (p_{q,j}, x_{q,j}, v_{q,j}) \rangle \mid j \in [|\mathcal{X}_q|]\}$. Note that despite the dependence of \mathcal{X}_q on q , we always have $|\mathcal{X}_q| \leq d$. Having seen how to express w -PBP, α , and both w -PBP $_q$ and α_q for all $q \in [\ell]$ as a set of key-value pairs, we are ready to state and prove the following lemma.

► **Lemma 4** (Proof in Appendix A [9]). *Let L be a w -PBP-recognized language. If the representations of w -PBP and, for every $q \in [\ell]$, α_q are given, then we can decide in a 2-round \mathcal{DMRC} -computation whether $\alpha \in L$ or not.*

In the following four lemmas, we show that α_q can be computed in a constant number of rounds from w -PBP and α for every $q \in [\ell]$. The challenge lies in designing an interface between the different reducers to bridge the gap between the ℓ program blocks w -PBP $_q$ and the given assignments, initially cut into ℓ block based solely on the indices of the input variables, without exceeding the memory limits. We begin with a brief overview of the four steps.

1. For each x_i , where $i \in [n]$, we compute the number of subprograms in which x_i appears, and denote this number by $\#S(x_i)$. Note that $\#S(x_i) \leq \ell$ and that $\#S(x_i)$ is the number of all those reducers for which the value assignment of x_i is generally required to compute the resulting permutations in the corresponding subprograms.

2. We compute the prefix sums of $\#S(x_i)$. For $i \in [n]$, let $y_i = \sum_{j=0}^i \#S(x_j)$. Note that y_i is the number of assignment triples $(p_{q,j}, x_{q,j}, v_{q,j})$ with $0 < j \leq i$ needed to compute the action of the first i subprograms and that $y_{n-1} = \sum_{q=0}^{\ell-1} |\alpha_q|$.
3. Based on the prefix sums, we will compute a *separation* of the input variables into ℓ contiguous blocks such that, for each $q \in [\ell]$, it is feasible for reducer $_q$ to produce from the q th block the input value assignments that it needs to contribute for the next step. This is nontrivial since the number of input assignments must not exceed $O(d)$ due to the memory limitation of reducer $_q$. A *separation* of the input variables $\{x_0, \dots, x_{n-1}\}$ is a list of $\ell - 1$ *split values* $\sigma_1, \dots, \sigma_{\ell-1}$ such that we have ℓ ordered, contiguous blocks $\{x_0, \dots, x_{\sigma_1}\}, \{x_{\sigma_1+1}, \dots, x_{\sigma_2}\}, \dots, \{x_{\sigma_{\ell-1}+1}, \dots, x_{n-1}\}$. For notational convenience, we let $\sigma_0 = -1$ and $\sigma_\ell = n - 1$. Let $\sigma_q = \max\{j \in [n] \mid y_j \leq qd\}$ for $q \in \{1, \dots, \ell - 1\}$. Using these split values each reducer $_q$ can provide all value assignments needed for the computation of all subprograms in the next step without violating the memory limitations.
4. We compute α_q for $q \in [\ell]$ by using w -PBP, the input assignment α , and the split values.
 - **Lemma 5** (Proof in Appendix A [9]). *Calculating $\#S(x_i)$ is in \mathcal{DMRC}^0 . That is, for each $i \in [n]$, $\#S(x_i)$ is computable from w -PBP in a constant number of \mathcal{DMRC} -rounds.*
 - **Lemma 6** (Proof in Appendix A [9]). *Computing the prefix-sums of $\#S(x_i)$ is in \mathcal{DMRC}^0 .*
 - **Lemma 7** (Proof in Appendix A [9]). *Each of the split values $\sigma_1, \dots, \sigma_{\ell-1}$ can be computed in one reducer with the required prefix-sums being made available in one more \mathcal{DMRC} -round.*
 - **Lemma 8** (Proof in Appendix A [9]). *Given w -PBP, α , and the split values $\sigma_0, \dots, \sigma_\ell$, we can, for each $q \in [\ell]$, compute α_q in a constant number of \mathcal{DMRC} -rounds.*

We finally obtain the desired inclusion by applying Theorem 3 and Lemmas 4 through 8.

- **Theorem 9.** *We have $\mathcal{NC}^1 \subseteq \mathcal{DMRC}^0$.*

3.3 Simulating \mathcal{NC}^i For All $i \geq 2$

For the higher levels in the hierarchy of Nick's class, we show how to simulate the involved circuits directly. We begin with a short outline of the proof.

Let $C_n = (V_n, E_n)$ be a \mathcal{NC}^{i+1} circuit with an input of size n , given as a set of nodes and a set of directed edges, together with an input assignment α . The total size of C_n in bits is N_O , the total size of the input assignment in bits is N_I , and $N = N_O + N_I$. Note that $\text{size}(C_n)$ is polynomial in n and $\text{depth}(C_n) \in O(\log^i n)$. We will take the following steps to simulate the circuit C_n with deterministic MapReduce computations:

1. We compute the level of each node in C_n .
2. The nodes and edges are sorted by their level.
3. Both the circuit C_n and the input assignment α are divided equally among the reducers.
4. We split the circuit into subcircuits computable in a constant number of rounds.
5. A custom communication scheme collects and constructs the complete subcircuits.
6. The entire circuit is evaluated via evaluation of the subcircuits.

Note that equal division of C_n in the third step is very different from the split in the fourth one, where the parts may differ radically in size. Great care must be taken so as to not violate any of the memory and time restrictions, necessitating the two unlike partitions. The subsequent steps then need to mediate between these dissimilar divisions. We will show that the steps (1) to (6) can be computed in $O(\log n)$, $O(1)$, $O(1)$, $O(1)$, $O(\log n)$, and $O(\text{depth}(C_n)/\log n)$ rounds, respectively, yielding the desired theorem.

- **Theorem 10.** *We have $\mathcal{NC}^{i+1} \subseteq \mathcal{DMRC}^i$ for all $i \in \mathbb{N}_+$ and all $0 < \varepsilon < 1/2$.*

3.3.1 Computing The Levels

We begin by showing how to compute the level of each node in the circuit in $O(\log n)$ \mathcal{DMRC} -rounds by simulating a CRCW-PRAM algorithm. (We mention in passing that this step requires more than a constant number of rounds, which prevents us from obtaining the result for $\mathcal{NC}^1 \subseteq \mathcal{DMRC}^0$ by simulating the circuits directly; the separate approach from Subsection 3.2 via Barrington's theorem is thus required for this case.)

In [18], an algorithm is presented that computes the levels of all nodes in a directed acyclic graph and can be computed on a CREW-PRAM with $O(n + m)$ processors in $O(\log^2 m)$ time, where n and m are the numbers of nodes and edges in the graph, respectively. The first stage of this algorithm relies partly on the computation of prefix-sums, which can be computed much more efficiently when switching to a CRCW-PRAM, as we will show below. A straightforward adaptation of the analysis in [18], taking into account the maximum in-degree and out-degree and separating out the computation of prefix-sums, yields the following result.

► **Lemma 11.** *Let $G = (V, E)$ be a directed acyclic graph with n nodes, m edges, maximum in-degree d_{in} , and maximum out-degree d_{out} . The level of each node in G can then be computed on a CRCW-PRAM with $P \in O(m + P_{\text{P-Sum}}(O(m)))$ processors in time $T \in O((\log m) \cdot (T_{\text{P-Sum}}(O(m)) + \log \max\{d_{\text{in}}, d_{\text{out}}\}))$, where $P_{\text{P-Sum}}(q)$ and $T_{\text{P-Sum}}(q)$ denote, respectively, the number of processors and the computation time to compute the prefix-sums of q numbers on a CRCW-PRAM.*

In the following lemma, we aim to lower the time and memory requirements for computing prefix-sums on a CRCW-PRAM as far as possible.

► **Lemma 12** (Proof in Appendix A [9]). *The prefix-sums of q numbers can be computed on a CRCW-PRAM with $P \in O(q \log q)$ processors and memory $M \in O(q)$ in constant time.*

We plug in the result of Lemma 12 into Lemma 11 and then apply it to the graph C_n . Since its in-degrees and out-degrees are bounded by a constant Δ , we have $m \leq \Delta n/2 \in O(n)$. Hence we can compute the levels of the nodes of C_n on a CRCW-PRAM with $P \in O(N \log N)$ processors in time $T \in O(\log n)$. By Corollary 2, we obtain the following result.

► **Lemma 13** (Proof in Appendix A [9]). *Computing the levels of all nodes in C_n is in \mathcal{DMRC}^1 .*

3.3.2 Sorting By Levels

Once the levels of all nodes are computed, each node in the circuit can be represented as $(\text{level}(x_i), x_i)$. Recall that the depth of C_n is just the maximum level. Since $\text{depth}(C_n) \in O(\log^k n)$ for some $k \in \mathbb{N}_+$ and the number of nodes is bounded by the number of edges, which is $\text{size}(C_n) \in O(N)$, we can encode each pair $(\text{level}(x_i), x_i)$ by appending to a bit string of length $\log(c_1 \log^k n)$ another one of length $\log(c_2 N)$, for appropriate constants c_1 and c_2 , which results in a bit string of length $\log(cN \log^k n)$ for $c = c_1 c_2 \in \mathbb{N}$. This enables us to identify each pair $(\text{level}(x_i), x_i)$ with a different bit string, which can be interpreted as an integer bounded by $cN \log^k n$. We call this integer the *sorting index* of node x_i . Crucially, we chose the bit string to start with the encoding of the level. Sorting the sorting indices thus means to sort the nodes of C_n by their level. The following lemma shows how prefix-sums can be used to perform such a sort so efficiently on a CRCW-PRAM that we can apply Corollary 2 to simulate it in a constant number of \mathcal{DMRC} -rounds.

► **Lemma 14** (Proof in Appendix A [9]). *A CRCW-PRAM with $P \in O(D \log D)$ processors and memory $M \in O(D)$ can sort any subset $I \subseteq \{1, \dots, D\}$ of integers in constant time.*

Combining Lemma 14 and Corollary 2 we obtain, by a careful analysis using $\varepsilon \neq 1/2$, the promised result.

► **Corollary 15** (Proof in Appendix A [9]). *Let $c \in \mathbb{N}$ and $0 < \varepsilon < 1/2$. Any set of distinct integers from $\{1, \dots, \lceil cN \log^k n \rceil\}$ can be sorted in a constant number of \mathcal{DMRC} -rounds.*

Once all the nodes are sorted by their sorting index (and therefore implicitly by their level), we can enumerate them in ascending order using the sorting index j ; that is, we represent each node as the key-value pair $\langle j; (\text{level}(v), v) \rangle$. Clearly, we obtain an analogous representation of the edges of the form $\langle i; ((j, (\text{level}(v), v), (j', (\text{level}(v'), v')))) \rangle$, which will prove useful later on.

3.3.3 Division of Circuit And Assignment Among Reducers

As we have already seen when discussing the branching programs, an assignment α to input variables $\mathcal{X} = \{x_0, x_1, \dots, x_{n-1}\}$ can be represented as a set $\{\langle i; (x_i, v_i) \rangle \mid i \in [n]\}$ of key-value pairs, where $\alpha(x_i) = v_i \in \{0, 1\}$.

The circuit C_n is now divided into $\ell = N_O^\varepsilon$ subsets of edges according to the sorting indices and input values that are assigned to each subset as in the case of branching programs. For every $q \in [\ell]$, let $C_n^q = \{((j, \text{level}(v), v), (j', \text{level}(v'), v')) \mid qd \leq j \leq (q+1)d - 1\}$, where $d = N_O^{1-\varepsilon}$, be the q th subset. Note that $|C_n^q| \in O(d)$. For every $q \in [\ell]$, the set of variables appearing in C_n^q is denoted as \mathcal{X}_q and the assignment α_q to \mathcal{X}_q is represented as $\{\langle j; x_{q,j}, v_{q,j} \rangle \mid j \in [\alpha_q]\}$, where $x_{q,j}$ is the j th variable that appears as an input in C_n^q , and $v_{q,j}$ is its assignment value. Just as seen in Lemma 8 for the case of a branching program, we can now, for all $q \in [\ell]$, compute α_q from C_n and α , yielding the following lemma.

► **Lemma 16.** *Computing α_q from C_n and α is in \mathcal{DMRC}^0 for every $q \in [\ell]$.*

We can therefore assume that each input node is represented by $\langle j; (\text{level}(x_{j_i}), x_{j_i}, v_{j_i}) \rangle$, a key-value pair that is computed from C_n^q and α_q for $q \in [\ell]$ in a single \mathcal{DMRC} -round.

3.3.4 Division Into Subcircuits By Levels

We divide $C_n = (V_n, E_n)$ into as few subcircuits as possible such that the simulation of each subcircuit is in \mathcal{DMRC}^0 and we can evaluate C_n by evaluating the subcircuits sequentially.

Given $v \in V_n$ and $\delta \in \mathbb{N}$, we define the v -down-circuit $C_\delta^{\text{down}}(v) = (V_\delta^{\text{down}}(v), E_\delta^{\text{down}}(v))$ of depth δ to be the subcircuit of C_n induced by $V_\delta^{\text{down}}(v) = \{u \mid \text{level}(v) \leq \text{level}(u) \leq \text{level}(v) + \delta, u \rightarrow^* v\}$, where $u \rightarrow^* v$ means that there is a directed path of any length (including 0) from u to v in C_n . The v -up-circuit $C_\delta^{\text{up}}(v) = (V_\delta^{\text{up}}(v), E_\delta^{\text{up}}(v))$ of depth δ is analogously the subcircuit of C_n induced by $V_\delta^{\text{up}}(v) = \{u \mid \text{level}(v) - \delta \leq \text{level}(u) \leq \text{level}(v), v \rightarrow^* u\}$.

When dividing C_n into subcircuits we have two conflicting goals. On the one hand, we want as few of them as possible, which implies that they have to be of great depth. On the other hand, we need to simulate them in MapReduce without exceeding the memory bounds. A depth in $\Theta(\log n)$ turns out to be the right choice. Let $s = (\gamma \log n) / \log \Delta$, where $\Delta \geq 2$ is a constant bounding the maximum degree of C_n and γ is an arbitrary constant satisfying $0 < \gamma < 1 - 2\varepsilon$. (Note that such a γ exists exactly if $\varepsilon < 1/2$.) Since a tree of depth s and maximum degree bounded by a constant Δ contains at most $\sum_{i=1}^s \Delta^i$ edges, its size is in $O(\Delta^s) = O(n^\gamma) \subseteq O(N^\gamma)$. Hence each reducer may contain up to $N^{1-\varepsilon}/N^\gamma$ such subcircuits without exceeding the memory constraint of $O(N^{1-\varepsilon})$; see Figure 2 in Appendix B [9]. We denote this number of allowed subcircuits per reducer by $\beta = N^{1-\varepsilon-\gamma}$.

For each $i \in \lceil \lceil \text{depth}(C_n)/s \rceil + 1 \rceil$, we define $L_i = i \cdot s$. For every node v on level L_i – that is, with $\text{level}(v) = L_i$ – we call the v -down-circuit (v -up-circuit, resp.) of depth s an L_i -down-circuit (L_i -up-circuit, resp.). We will construct in each reducer the v -down-circuits and v -up-circuits of depth 1 of all its nodes. From those we then construct all L_i -down-circuits and L_i -up-circuits for every i . Note that we can evaluate all L_i -down-circuits if the values of the nodes of level L_{i+1} are given. The values of the nodes v of level L_{i+1} that are necessary to compute the L_i -up-circuits are then known from the L_{i+1} -down-circuits.

When the circuit C_n is divided into L_i -down-circuits, there may exist edges of C_n that are not contained in any L_i -down-circuit. If an edge $((j_u, \text{level}(u), u), (j_v, \text{level}(v), v))$ satisfies $L_{i_u} \leq \text{level}(u) \leq L_{i_u+1}$ and $L_{i_v} \leq \text{level}(v) \leq L_{i_v+1}$ for $i_u \neq i_v$, then this edge is not included in any L_{i_u} -down-circuit nor any L_{i_v} -down-circuit. We call such edges *level-jumping edges*; see Figure 3 in Appendix B [9] for an example. We would like to replace every level-jumping edge (u, v) by a path from u to v that consists only of edges that will be part of the respective L_i -down-circuits and L_i -up-circuits in the resulting, *augmented* circuit. The following lemma states that this is possible without increasing the size by too much.

► **Lemma 17** (Proof in Appendix A [9]). *We can subdivide the jumping edges in C_n in a way that renders the subcircuit-wise evaluation possible without increasing the size beyond $O(N)$.*

3.3.5 Construction of Subcircuits in Reducers

Having described the subcircuits on which the evaluation of the entire circuits will be based, we now need to show how to split and construct them in the ℓ different reducers. In each reducer, we start with the nodes v contained in it that satisfy $\text{level}(v) = L_i$ for any i and the associated v -down-circuits and v -up-circuits of depth 1. We then iteratively increase the depth one by one, until the full L_i -down-circuits and L_i -up-circuits of depth up to s are constructed. Note that the nodes of any level L_i and their corresponding circuits may be scattered across multiple reducers since edges were split equally among them according to their the sorting index and not depending on the level. We therefore need to carefully implement a communication scheme that allows each reducer to encode requests for missing edges required in the construction, which are then delivered to them in multiple rounds, without exceeding any of the memory or time bounds. Taking care of all these details, we obtain the following lemma.

► **Lemma 18** (Proof in Appendix A [9]). *Given C_n , all L_i -down-circuits and L_i -up-circuits can be constructed in $O(\log n)$ DMRC-rounds whenever $0 < \varepsilon < 1/2$.*

3.3.6 Evaluation Via Subcircuits

The main idea in the proof of the following lemma is to compute the evaluation values subcircuit-wise, starting with the deepest ones, and then iteratively moving up the circuit in $\text{depth}(C_n)/s$ rounds, passing on the newly computed values to the right reducers, until the value of the unique output node is known.

► **Lemma 19.** *If all up-circuits and down-circuits are constructed in the proper reducers, C_n can be evaluated in $O(\text{depth}(C_n)/\log n)$ DMRC-rounds.*

4 Conclusion and Research Opportunities

In a substantial improvement over all previously known results, we have shown that $\mathcal{NC}^{i+1} \subseteq \mathcal{DMRC}^i$ for all $i \in \mathbb{N}$. In the case of $\mathcal{NC}^1 \subseteq \mathcal{DMRC}^0$, we have proved this result for every feasible choice of ε in the model, that is, for $0 < \varepsilon \leq 1/2$. For $i > 0$, we have shown the result to hold for all but one value, namely $\varepsilon = 1/2$.

Achieving these two results required a detailed description of two different, delicate simulations within the MapReduce framework. For the case of \mathcal{NC}^1 , which is particularly relevant in practice, we applied Barrington’s theorem and simulated width-bounded branching programs [2], whereas we directly simulated the circuits for the higher levels of the hierarchy. We emphasize that none of the two approaches can replace the other: Barrington’s theorem only gives a characterization for the first level of the \mathcal{NC} hierarchy and the second approach does not even yield $\mathcal{NC}^1 \subseteq \mathcal{MRC}^0$. (Recall that \mathcal{DMRC} is just the deterministic variant of \mathcal{MRC} , so we have $\mathcal{DMRC}^i \subseteq \mathcal{MRC}^i$ for all $i \in \mathbb{N}$.)

We would like to briefly address the small question that immediately arises from our result, namely whether it possible to extend the inclusion $\mathcal{NC}^{i+1} \subseteq \mathcal{DMRC}^i$ of Theorem 10 to the case $\varepsilon = 1/2$. Going through all involved lemmas, we see that the two reasons that our proof does not work in this corner case are the sorting of the nodes using Lemma 15 and the construction of the up-circuits and down-circuits in Lemma 18. Regarding the former, we can avoid the restriction by allowing randomization. For the latter, it is not clear that this can be achieved, however. If there was any way to construct the levels for $\varepsilon = 1/2$ as well, then Theorem 10 would immediately extend to the full range $0 < \varepsilon \leq 1/2$ of feasible choices for ε .

Besides dealing with the small issue mentioned above, the natural next step for future research is to take the complementary approach and address the reverse relationship: Having shown in this paper how to obtain efficient deterministic MapReduce algorithms for parallelizable problems, we now aim to include \mathcal{DMRC}^i into \mathcal{NC}^{i+1} for all $i \in \mathbb{N}$, thus finally settling the long-standing open question of how exactly the MapReduce classes correspond to the classical classes of parallel computation.

References

- 1 S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.
- 2 D.A. Barrington. Bounded-Width Polynomial-Size Branching Programs Recognize Exactly Those Languages in \mathcal{NC}^1 . *J. of Computer and System Sciences*, 38:150–164, 1989.
- 3 C.-T. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. R. Bradski, A. Y. Ng, and K. Olukotum. MapReduce for machine learning on multicore. In *Advances in neural information processing systems (NIPS)*, pages 281–288, 2006.
- 4 T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.
- 5 J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.
- 6 A. K. Farahat, A. Elgohary, A. Ghodsi, and M. S. Kamel. Distributed Column Subset Selection on MapReduce. In *International Conference on Data Mining (ICDM)*, pages 171–180, 2013.
- 7 J. Feldman, S. Muthukrishnan, A. Sidiropoulos, C. Stein, and Z. Svitkina. On Distributing Symmetric Streaming Computations. *ACM Trans. on Algorithms*, 6(4):66:1–66:15, 2010.
- 8 B. Fish, J. Kun, Á. D. Lelker, L. Reyzin, and G. Turán. On the Computational Complexity of MapReduce. In *International Symposium on Distributed Computing (DISC)*, pages 1–15, 2015.
- 9 F. Frei and K. Wada. Efficient Circuit Simulation in MapReduce. *Technical Report arXiv.org*, cs(arXiv:1907.01624):1–20, 2019. arXiv:1907.01624.

- 10 M. Goodrich, N. Sichinava, and Q. Zhang. Sorting, Searching, and Simulation in the MapReduce Framework. In *22nd Int. Symp. on Algorithms and Computation (ISAAC)*, pages 374–383, 2011.
- 11 S. Kamara and M. Raykova. Parallel Homomorphic Encryption. In *Financial Cryptography Workshops*, pages 213–225, 2013.
- 12 H. Karloff, S. Suri, and S. Vassilvitskii. A Model of Computation for MapReduce. In *21st ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 938–948, 2010.
- 13 R. Kumar, B. Moseley, and S. Vassilvitskii. Fast Greedy Algorithms in MapReduce and Streaming. In *ACM Symp. on Parallelism in Algorithms and Architectures (SPAA)*, pages 1–10, 2013.
- 14 M.F. Pace. BSP vs MapReduce. In *12th Int. Conf. on Computational Science (ICCS)*, pages 246–255, 2012.
- 15 A. Pietracaprina, G. Pucci, M. Riondato, F. Silvestri, and E. Upfal. Space-Round Tradeoffs for MapReduce Computations. In *26th ACM Int. Conf. on Supercomputing (ICS)*, pages 235–244, 2012.
- 16 T. Roughgarden, S. Vassilvitskii, and J.R. Wang. Shuffles and Circuits (On Lower Bounds for Modern Parallel Computation). *Journal of the ACM (JACM)*, 65(6):41:1–66:24, 2018.
- 17 A.D. Sarma, F.N. Afrati, S. Salihoglu, and J.D. Ullman. Upper and Lower Bounds on the Cost of a Map-Reduce Computation. In *Proceedings of the VLDB Endowment (PVLDB)*, pages 277–288, 2013.
- 18 A. Tada, M. Migita, and R. Nakamura. Parallel Topological Sorting Algorithm. *J. of the Information Processing Society of Japan (IPSJ)*, 45(4):1102–1111, 2004.
- 19 T. White. *Hadoop: The Definitive Guide, 4th edition*. O’Reilly, 2015.

A

 Deferred Proofs

In this appendix, we provide all proofs that had to be deferred due to the space constraints. For the reader’s convenience, we reprint all statements.

► **Lemma 4** (Reprint of Lemma 4 on page 7). *Let L be a w -PBP-recognized language. If the representations of the w -PBP and, for every $q \in [\ell]$, α_q are given, then we can decide in a 2-round DMRC-computation whether $\alpha \in L$ or not.*

Proof. As already described above, let w -PBP be represented by the set $\{\langle p; (x_{i_p}, f_p, g_p) \rangle \mid p \in [t]\}$ and, for every $q \in [\ell]$, the assignment α_q by $\{\langle q, (p_{q,j}, x_{q,j}, v_{q,j}) \rangle \mid j \in [|\mathcal{X}_q|]\}$. Note that there are ℓ subprograms of length at most d and ℓ partial assignments that each assign values to at most one variable per line of the corresponding partial program. The total size of the input is thus in $O(d\ell) \subseteq O(N_0) \subseteq O(N)$.

We define the first map function μ_1 by

$$\begin{aligned} \mu_1(\langle p; (x_{i_p}, f_p, g_p) \rangle) &= \{\langle p \div d; (p, x_{i_p}, f_p, g_p) \rangle\}, \text{ for each } p \in [t] \text{ and} \\ \mu_1(\langle q; (p_{q,j}, x_{q,j}, v_{q,j}) \rangle) &= \{\langle p_{q,j} \div d; (p_{q,j}, x_{q,j}, v_{q,j}) \rangle\} \text{ for each } q \in [\ell], j \in [k+1]. \end{aligned}$$

For any $q \in [\ell]$, there is one subprogram w -PBP $_q$ and an associated assignment set α_q . We use the map function μ_1 to find the value assignment for each variable appearing in w -PBP $_q$ and store it in a key-value pair. This pair has the key q and is thereby designated to be processed by reducer $_q$, which can calculate ρ_1 , having all pairs with key q available. This function simulates, for each permutation π of $[w]$, the subprogram w -PBP $_q$ on this permutation with the received assignment and stores the resulting permutation π' . This yields a table T_q of size $w! \in O(1)$, describing the action of w -PBP $_q$ for the given assignment on all $w!$ permutations. (We mention in passing that for the first reducer $_0$ it would be sufficient to compute and store

only the permutation that results from applying $w\text{-PBP}_0$ on the given assignment to the identity as the initial permutation, thus saving the time and memory necessary for the rest of the first table.) The output of ρ_1 on the q th reducer is $\langle q; T_q \rangle$.

The map function μ_2 of the second round is simple, it maps $\langle q; T_q \rangle$ to $\langle 0; (q, T_q) \rangle$, thus delivering all pairs (i, T_i) to a single instance of the reduce function ρ_2 . This first reducer has therefore all tables $T_0, \dots, T_{\ell-1}$ at its disposal and knows which one is which. Using T_q as a look-up table for the permutation performed by $w\text{-PBP}_q$, reducer₀ can now compute, starting from the identity permutation id , the permutation $\pi = T_{\ell-1} \circ \dots \circ T_2 \circ T_1 \circ T_0(\text{id})$, and the input is accepted if and only if $\pi \in F_n$, where F_n is the set of accepted permutations that is given to us alongside the program $w\text{-PBP}$. ◀

► **Lemma 5** (Reprint of Lemma 5 on page 8). *Calculating $\#S(x_i)$ is in \mathcal{DMRC}^0 . That is, for each $i \in [n]$, $\#S(x_i)$ is computable from $w\text{-PBP}$ in a constant number of \mathcal{DMRC} -rounds.*

Proof. For each $q \in [\ell]$, the subprogram $w\text{-PBP}_q$ is stored in reducer _{q} . The output of reducer _{q} – which will be the input to compute $\#S(x_i)$ – is $\langle q; (q, 1) \rangle, \dots, \langle q; (q, k_q) \rangle$, with the variables $x_{q,1}, \dots, x_{q,k_q}$ appearing in the subprogram $w\text{-PBP}_q$ and $k_q \in O(d)$. The total number of inputs used to compute $\#S(x_i)$ is therefore at most $d\ell \in O(N)$. We use a Sum-CRCW-PRAM, whose concurrent writes to a single memory register are resolved by summing up all values being written to the same register simultaneously, see [10]. We use at most $d\ell$ processors, $P_{q,1}, \dots, P_{q,k_q}$ for each $q \in [\ell]$, and registers R_0, \dots, R_{n-1} and let all processors $P_{q,j}$ add 1 to R_j concurrently. Thus we see that computing $\#S(x_i)$ is possible in constant time on a Sum-CRCW-PRAM and therefore, by Corollary 2, in \mathcal{DMRC}^0 . ◀

► **Lemma 6** (Reprint of Lemma 6 on page 8). *Computing the prefix-sums of $\#S(x_i)$ is in \mathcal{DMRC}^0 .*

Proof. The input is given as $\langle i; (\#S(x_i), i) \rangle$ for $i \in [n]$. We compute the prefix-sums y_i of $\#S(x_i)$ for all $i \in [n]$ in three rounds that can be summarized as follows:

1. Each reducer _{q} , for $q \in [\ell]$, determines its local prefix-sums; that is, it computes the d prefix-sums $y_{dq}^{\text{local}}, \dots, y_{d(q+1)-1}^{\text{local}}$ of the d numbers $\#S(x_{dq}), \dots, \#S(x_{d(q+1)-1})$.
2. A single reducer computes the prefix-sums $z_0, z_1, \dots, z_{\ell-1}$ of $y_{d-1}^{\text{local}}, y_{2d-1}^{\text{local}}, \dots, y_{\ell d-1}^{\text{local}}$, which are known from the first round. For every $q \in [\ell-1]$, we send z_q to reducer _{$q+1$} .
3. Each reducer _{$q+1$} with $q \in [\ell-1]$ computes $y_{d(q+1)+j} = y_{d(q+1)+j}^{\text{local}} + z_q$ for each $j \in [d]$.

We now describe the three rounds in more detail at the level of the key-value pairs.

1. By defining the map function $\mu_1(\langle i; (\#S(x_i), i) \rangle) = \langle i \div d; (\#S(x_i), i) \rangle$, each reducer _{q} , for $q \in [\ell]$, receives $\#S(x_{dq}), \dots, \#S(x_{d(q+1)-1})$ together with the correct indices. Thus we can compute in reducer _{q} all local prefix-sums $y_{dq}^{\text{local}}, \dots, y_{d(q+1)-1}^{\text{local}}$ of these number. The output of reducer _{q} consists of the local prefix-sums in the format $\langle q; (\text{p-sum}, q, j, y_{q,j}^{\text{local}}) \rangle$ for $j \in [d]$ and the last of each group of local prefix-sums in the format $\langle q; (\text{last}, y_{d(q+1)-1}^{\text{local}}) \rangle$, where $\text{p-sum} = 0$ and $\text{last} = 1$ is a simple binary identifier.
2. By defining the map function $\mu_2(\langle q; (\text{last}, y_{d(q+1)-1}^{\text{local}}) \rangle) = \langle 0; (\text{last}, y_{d(q+1)-1}^{\text{local}}) \rangle$, all last parts of the local prefix-sums can be gathered in reducer₀. Thus, the prefix-sums $z_0, z_1, \dots, z_{\ell-1}$ of $y_{d-1}^{\text{local}}, \dots, y_{\ell d-1}^{\text{local}}$ can be computed in it and the output of the reducer is $\langle 0; (\text{last}, i+1, z_i) \rangle$ for every $i \in [\ell-1]$. All other key-value pairs – that is, those of the form $\langle q; (\text{p-sum}, q, j, y_{q,j}^{\text{local}}) \rangle$ – are passed on unaltered.
3. The input of the third round consists of the output pairs $\langle q; (\text{p-sum}, q, j, y_{q,j}^{\text{local}}) \rangle$ for all $j \in [d]$ and $q \in [\ell]$ passed on from the first round and the pairs $\langle 0; (\text{last}, q+1, z_q) \rangle$ for all $q \in [\ell-1]$ from the second round. Defining the map function as $\mu_3(\langle q; (\text{p-sum}, q, j, y_{q,j}^{\text{local}}) \rangle) =$

$\langle q; (\text{p-sum}, q, j, y_{q,j}^{\text{local}}) \rangle$ and $\mu_3(\langle 0; (\text{last}, q+1, z_q) \rangle) = \langle q+1; (\text{last}, q+1, z_q) \rangle$, we can, for each $j \in [d]$ and each $q \in \{1, \dots, \ell-1\}$, compute $y_{q,j} = y_{q,j}^{\text{local}} + z_j$ in reducer $_q$.

The memory limitations of the mappers and reducers are clearly respected. \blacktriangleleft

► **Lemma 7** (Reprint of Lemma 7 on page 8). *Each of the split values $\sigma_1, \dots, \sigma_{\ell-1}$ can be computed in one reducer with the required prefix-sums being made available in one more DMR C-round.*

Proof. If there is a $k \in [\ell-1]$ such that $y_{n-1} \leq kd$, then it is clear from the definition $\sigma_q = \max\{j \in [n] \mid y_j \leq qd\}$ of the split values that $\sigma_k = \sigma_{k+1} = \dots = \sigma_{\ell-1}$. We can therefore assume that $y_{n-1} > (\ell-1)d$ and characterize, for each $q \in \{1, \dots, \ell-1\}$, the split value σ_q as the unique integer satisfying $(q-1)d < y_{\sigma_q} \leq qd$ and $qd < y_{\sigma_q+1}$; see Figure 1 in Appendix B [9].

This characterization is well defined since $0 < \#S(x_i) \leq \ell < d$ for each $i \in [n]$ and $y_{n-1} \leq d\ell \in O(N_O)$. For each $q \in [\ell]$, in order to determine the split value σ_q , it is therefore sufficient to have available in the respective reducer a sequence of consecutive prefix-sums such that the first one is at most qd and the last one is greater than qd . This condition is satisfied if reducer $_q$ has the $d+2$ consecutive prefix-sums $y_{qd-1}, y_{qd}, \dots, y_{(q+1)d-1}, y_{(q+1)d}$ available. (For the first and the last reducer, the $d+1$ prefix-sums y_0, \dots, y_{d-1}, y_d and $y_{(\ell-1)d-1}, y_{(\ell-1)d}, \dots, y_{\ell d-1}$, respectively, will suffice.) Slightly extending the sequence of available prefix-sums in each reducer by copying the overlapping prefix-sums from another reducer thus enables us to compute all split values in the ℓ reducers. Since for each $q \in [\ell]$, there are the d prefix-sums $y_{qd}, \dots, y_{(q+1)d-1}$ in reducer $_q$, each reducer can have the $d+2$ prefix-sums made available after one more round by having each neighboring reducer copy one more prefix-sum into it. We have $\sigma_0 = -1$ and $\sigma_\ell = n-1$; it is thus immediately verified that, for every $q \in [\ell]$, the total number of subprograms in which input variables between x_{σ_q+1} and $x_{\sigma_{q+1}}$ appear is at most $2d$, showing that all the memory restrictions on the reducers are observed. \blacktriangleleft

► **Lemma 8** (Reprint of Lemma 8 on page 8). *Given w -PBP, α , and the split values $\sigma_0, \dots, \sigma_\ell$, we can, for each $q \in [\ell]$, compute α_q in a constant number of DMR C-rounds.*

Proof. We can assume that, for each $\kappa \in [\ell]$, the reducer $_\kappa$ has the subprogram w -PBP $_\kappa$, the κ th block of input assignments $\{(x_j, v_j) \mid \kappa \cdot d \leq j \leq (\kappa+1)d-1\}$, and the split values $\sigma_0, \dots, \sigma_\ell$ available. The output of reducer $_\kappa$ then consists of the following:

1. $\langle \kappa; (q, p, x_{i_p}, f_p, g_p) \rangle$ for each line (p, x_{i_p}, f_p, g_p) in w -PBP $_\kappa$, where $\sigma_q + 1 \leq i_p \leq \sigma_{q+1}$.
2. $\langle \kappa; (q, x_j, v_j) \rangle$ for each value assignment (x_j, v_j) with $\sigma_q + 1 \leq j \leq \sigma_{q+1}$.

For any $\kappa \in [\ell]$, we need to bound the total number of outputs with key κ from above. From the definition of the split values we see that this number is in $O(d)$ since it is bounded by the number of lines, which is at most $2d$, plus the number of assignments, which is at most d .

Naturally, the map function μ of the next round is defined by

1. $\mu(\langle \kappa; (q, p, x_{j_p}, f_p, g_p) \rangle) = \langle q; (p, x_{j_p}, f_p, g_p) \rangle$ and
2. $\mu(\langle \kappa; (q, x_j, v_j) \rangle) = \langle q; (x_j, v_j) \rangle$.

For any $\kappa \in [\ell]$, the assignment variables α_q can be computed by the subsequent reduce function using the key-value pairs produced above. For each $q \in [\ell]$, the reducer $_q$ has now available the lines of w -PBP and the value assignments for the input variables between x_{σ_q+1} and $x_{\sigma_{q+1}}$. It can therefore go through all the program lines and determine, on the one hand, which value assignments they require and, on the other hand, to which subprogram they belong. The required assignment information is then sent to the respective reducers by outputting $\langle q; (p \div d, p, x_{i_p}, v_{i_p}) \rangle$. \blacktriangleleft

► **Lemma 12** (Reprint of Lemma 12 on page 9). *The prefix-sums of q numbers can be computed on a CRCW-PRAM with $P \in O(q \log q)$ processors and memory $M \in O(q)$ in constant time.*

Proof. We use a Sum-CRCW-PRAM, where concurrent writes to the same memory register are resolved by adding up all simultaneously assigned numbers [10]. Let q numbers x_0, x_1, \dots, x_{q-1} be given as input. Without loss of generality, we assume q to be a power of 2 and calculate $s_i(j) = \sum_{j2^i \leq p < (j+1)2^i} x_p$ for all $i \in [1 + \log q]$ and all $j \in [q/2^i + 1]$; see Figure 4 in Appendix B [9] for an illustrating example.

Since each of the $q/2^i$ elements in s_i is the sum of 2^i elements, we can – by allocating q processors for each $i \in [1 + \log q]$ – compute every $s_i(j)$ in a Sum-CRCW-PRAM with $O(q \log q)$ processors and $O(1)$ time.

We now describe how the prefix-sums $y(0), y(1), \dots, y(q-1)$ are computed from the $s_i(j)$. Assume first that $j+1$ is a power of 2, that is, $j+1 = 2^p$. Then we have $y(j) = s_p(0)$, so the value has already been computed. If $j+1 = 2^p + 1$ for some p , then we have $y(j) = s_p(0) + s_0(2^p)$, so we need to add two summands. In general, $y(j)$ can be calculated as the sum of at most $\log q - 1$ known summands.

Let $a_{\log q}^j a_{(\log q)-1}^j \dots a_0^j$ be the binary representation of $j+1$. Now, we can see that

$$\begin{aligned} y(j) &= s_{\log q}(0) \cdot a_{\log q}^j \\ &\quad + s_{(\log q)-1}((j+1 - 2^{(\log q)-1}) \div 2^{(\log q)-1}) \cdot a_{(\log q)-1}^j \\ &\quad + \dots \\ &\quad + s_1((j+1 - 2^1) \div 2^1) \cdot a_1^j \\ &\quad + s_0((j+1 - 2^0) \div 2^0) \cdot a_0^j; \end{aligned}$$

that is, $y(j)$ can be computed as the sum of all $s_p((j+1 - 2^p) \div 2^p)$ such that $a_p^j = 1$. Thus, it is sufficient to supply a maximum of $(\log q) - 1$ processors for the calculation of each $y(j)$ in a second time step, and the prefix-sums can be computed on a Sum-CRCW-PRAM with $O(q \log q)$ processors in constant time. ◀

► **Lemma 13** (Reprint of Lemma 13 on page 9). *Computing the levels of all nodes in C_n is in \mathcal{DMRC}^1 .*

Proof. From Lemmas 11 and 12 we know that the level of each node in C_n can be computed in $T \in O(\log n)$ time on a Sum-CRCW-PRAM with $P \in O(N + N \log N)$ processors. Now, Corollary 2 yields a MapReduce simulation of this Sum-CRCW-PRAM. We need to check that the conditions of Corollary 2 are indeed all satisfied: From $T \in O(\log n)$, $P \in O(N + N \log N)$, and $M \in O(N)$ follows $M + P \in O(N \log N)$ and $\log_{N^{1-\varepsilon}}(M + P) \in O(1)$, hence we have $(M + P) \log_{N^{1-\varepsilon}}(M + P) \in O(N^{2(1-\varepsilon)})$. Thus, the level of each node in C_n can be computed in $O(\log n)$ \mathcal{DMRC} -rounds. ◀

► **Lemma 14** (Reprint of Lemma 14 on page 10). *A CRCW-PRAM with $P \in O(D \log D)$ processors and memory $M \in O(D)$ can sort any subset $I \subseteq \{1, \dots, D\}$ of integers in constant time.*

Proof. Recall that we use a Sum-CRCW-PRAM that sums up concurrent writes. Assume that the input and output are stored in the arrays $x[0], \dots, x[p-1]$ and $y[0], \dots, y[p-1]$, respectively. We will use two auxiliary arrays $z[0], \dots, z[D]$ and $\hat{z}[0], \dots, \hat{z}[D]$ of size $D+1$. The algorithm works in four steps:

1. Initialize z by using $D + 1 \leq P$ processors to set $z[k] \leftarrow 0$ for all $k \in [D + 1]$.
2. Use $p \leq P$ processors in parallel to set $z[x[k]] \leftarrow 1$ for all $k \in [p]$.
3. Compute the prefix-sums of the array z and save them into \hat{z} .
4. Use D processors to set, for all $k \in \{1, \dots, D\}$ in parallel, $y[\hat{z}[k]] \leftarrow k$ if and only if $\hat{z}[k] \neq \hat{z}[k - 1]$.

Since the prefix-sums of D numbers can be computed by the Sum-CRCW PRAM with $P \in O(D \log D)$ processors and memory $M \in O(D)$ in constant time by Lemma 12, the above algorithm stays within these bounds as well.

We now prove that this algorithm is correct. First we observe that after step 2, for every $k \in \{1, \dots, D\}$, we have $z[k] = 1$ if and only if one of the p integers to be sorted is k . Because \hat{z} contains the prefix-sums of z , the value stored in $\hat{z}[k]$ hence tells us how many of the p integers in x are at most k . (Note that accordingly we always have $z[0] = \hat{z}[0] = 0$.) Thus k is one of the integers in x if and only if $\hat{z}[k] = \hat{z}[k - 1] + 1$; otherwise, we have $\hat{z}[k] = \hat{z}[k - 1]$. As a consequence, the array \hat{z} contains exactly the indices of x , namely $[p]$, as values in non-decreasing order, that is, $0 = \hat{z}[0] \leq \hat{z}[1] \leq \dots \leq \hat{z}[D - 1] \leq \hat{z}[D] = p$. Stepping through \hat{z} from start to end, that is, from $k = 0$ to $k = D$, we therefore observe an increment of 1 from $\hat{z}[k - 1]$ to $\hat{z}[k]$ exactly if k is one of the integers to be sorted. This means that in step 4 the integers contained in x are detected from left to right in ascending order and subsequently stored into y in the same order. \blacktriangleleft

► Corollary 15 (Reprint of Lemma 15 on page 10). *Let $c \in \mathbb{N}$ and $0 < \varepsilon < 1/2$. Any set of distinct integers from $\{1, \dots, \lceil cN \log^k n \rceil\}$ can be sorted in a constant number of \mathcal{DMRC} -rounds.*

Proof. We apply Lemma 14 with $D \in O(N \log^k n)$. We have $D \in O(N \log^k N) \subseteq O(N^{1+\zeta})$ and thus also $D \log D \in O(N^{1+\zeta})$ for any constant $\zeta > 0$. Choose any $\zeta < 1 - 2\varepsilon$, which is possible for $\varepsilon < 1/2$. The sorting is then possible on a CRCW-PRAM with $O(N^{1+\zeta})$ processors and $O(N^{1+\zeta})$ memory in constant time. By Corollary 2, this CRCW-PRAM can be simulated in a constant number of \mathcal{DMRC} -rounds because $\log_{N^{1-\varepsilon}}(N^{1+\zeta}) = (1+\zeta)/(1-\varepsilon) \in O(1)$ and $O(N^{1+\zeta}) \subseteq O(N^{2(1-\varepsilon)})$. \blacktriangleleft

► Lemma 17 (Reprint of Lemma 17 on page 11). *We can subdivide the jumping edges in C_n in a way that renders the subcircuit-wise evaluation possible without increasing the size beyond $O(N)$.*

Proof. Let $((j_u, \text{level}(u), u), (j_v, \text{level}(v), v))$ be a jumping edge, where $L_{i_u} \leq \text{level}(u) \leq L_{i_u+1}$, $L_{i_v} \leq \text{level}(v) \leq L_{i_v+1}$, and $i_u < i_v$. If $i_u = i_v - 1$, then this edge is divided into two edges $((j_u, \text{level}(u), u), \text{dummy})$ and $(\text{dummy}, (j_v, \text{level}(v), v))$, introducing a new node dummy of the id kind with $\text{level}(\text{dummy}) = i_v$. If $i_u \leq i_v - 2$, then this edge is divided into three edges $((j_u, \text{level}(u), u), \text{dummy}_1)$, $(\text{dummy}_1, \text{dummy}_2)$, and $(\text{dummy}_2, (j_v, \text{level}(v), v))$, introducing two new nodes with $\text{level}(\text{dummy}_1) = i_u + 1$, $\text{level}(\text{dummy}_2) = i_v$. Having divided the jumping edges in this way, the newly created edges are all part of some dummy-down-circuit or dummy-up-circuit, except for edges of the form $(\text{dummy}_1, \text{dummy}_2)$. Note that we cannot further subdivide the edges of the form $(\text{dummy}_1, \text{dummy}_2)$ because we would exceed the size limit on the circuit otherwise. The most convenient way to deal with this is to adjust our definition of down-circuits and up-circuits such that every edge of the form $(\text{dummy}_1, \text{dummy}_2)$ is considered to be both a dummy_1 -down-circuit and a dummy_2 -up-circuit on its own. This way, every edge in the augmented circuit is included in

some down-circuit or up-circuit. Note that this augmentation can be performed in a single round and that the size of the augmented circuit is in $O(N)$. In what follows, we consider C_n to be this augmented circuit. ◀

► **Lemma 18** (Reprint of Lemma 18 on page 11). *Given C_n , all L_i -down-circuits and L_i -up-circuits can be constructed in $O(\log n)$ DMR rounds whenever $0 < \varepsilon < 1/2$.*

Proof. In the first round, the map function μ_1 is defined such that each reducer $_q$ is assigned (via the choice of the key) β nodes of the form $\langle j; (\text{level}(v), v) \rangle$ and directed edges adjacent to these nodes. Note that one edge can thus be assigned to two different reducers, once as an outgoing and once as an incoming edge. Specifically, we define

$$\mu_1(\langle j; (\text{level}(v), v) \rangle) = \{ \langle j \div \beta; (j, \text{level}(v), v) \rangle \}$$

for the key-value pairs representing nodes and

$$\begin{aligned} \mu_1(\langle i; ((j, \text{level}(v), v), (j', \text{level}(v'), v')) \rangle) = & \{ \langle j \div \beta; ((j, \text{level}(v), v), (j', \text{level}(v'), v')) \rangle, \\ & \langle j' \div \beta; ((j, \text{level}(v), v), (j', \text{level}(v'), v')) \rangle \} \end{aligned}$$

for the key-value pairs representing edges.

In the subsequent execution of ρ_1 , each reducer can therefore directly construct the v -up-circuits and v -down-circuits of depth 1 for its β assigned nodes. We will now describe how some of these initial circuits, namely those on levels L_i for any $i \in [r]$, can be extended to full L_i -up-circuits and L_i -down-circuits by iteratively increasing the circuit depth one by one in the following way:

Let v be a node with $\text{level}(v) = L_i$ in reducer $_q$ for any $i \in [r]$ and $q \in [\ell]$. We want to extend $C_1^{\text{down}}(v)$ and $C_1^{\text{up}}(v)$ to $C_2^{\text{down}}(v)$ and $C_2^{\text{up}}(v)$, respectively. Let u_{in} (u_{out} , resp.) be any node of in-degree (out-degree, resp.) 0 in it, that is, any node that potentially needs to be extended by one or multiple edges. These extending edges are not necessarily available in reducer $_q$, however. We need to find out which reducer stores them – if there are any – and then request these edges from it in some way. To determine the right reducer, we make use of the sorting index stored alongside each node, even when part of an edge. Any edge (u_{in}, v) that we need to check for possible extensions is in fact represented as $\langle q, ((j_{u_{\text{in}}}, \text{level}(u_{\text{in}}), u_{\text{in}}), (j_v, \text{level}(v), v)) \rangle$ in reducer $_q$. The number of the reducer containing the downward extending edges is now retrieved as $\text{to}(u_{\text{in}}) = j_{u_{\text{in}}} \div \beta$. Analogously, the upward extending edges for an edge (v, u_{out}) are to be found in reducer $_{\text{to}(u_{\text{out}})}$, where $\text{to}(u_{\text{out}}) = j_{u_{\text{out}}} \div \beta$. We now know whom to ask for edges extending the subcircuit beyond node u , namely reducer number $\text{to}(u)$. Let $\text{from}(v) = q$ denote the number of the reducer sending the request, which we encode in form of the key-value pair $\langle q; (u, \text{to}(u), \text{from}(v)) \rangle$.

Each reducer $_q$ does the above for every node with possible extending edges and also passes along to the mapper all v -up-circuits and v -down-circuits constructed so far unaltered. This concludes the first round. In the second round, the map function μ_2 naturally re-assigns $\langle q; (u, \text{to}(u), \text{from}(v)) \rangle$ to reducer $_{\text{to}(u)}$, and returns the v -up-circuits and v -down-circuits to the reducers that sent them. Having received the edge request of the form $\langle \text{to}(u); (u, \text{to}(u), \text{from}(v)) \rangle$ while executing ρ_2 , reducer $_{\text{to}(u)}$ now sends all edges potentially useful to reducer $_{\text{from}(v)}$ – that is, the entire u -up-circuit and the entire u -down-circuit of depth 1 – to the next mapper in the form of a pair $(\text{from}(v), e)$ for every edge containing node u . As before, all other circuits constructed so far get passed along without modification as well.

In the third round, the map function μ_3 routes the requested edges to the requesting reducer by generating the key-value pairs $\langle \text{from}(v); (\text{from}(v), e) \rangle$. In the reducing step, which implements the same reduce function ρ_1 as in the first round, $\text{reducer}_{\text{from}(v)}$ now finally has all v -up-circuits and v -down-circuits fully extended to depth 2.

Since performing the two rounds $\mu_2, \rho_2, \mu_3, \rho_1$ deepens the L_i -up-circuits and L_i -down-circuits by one level in the way just seen, the complete L_i -up-circuits and L_i -down-circuits can be constructed by repeating these two rounds s times.

It is again clear that the memory and I/O requirements of the reducers are all met in every round since the input size and output size are in $O(d)$ for each reducer. Moreover, the total memory for storing the v -up-circuits and v -down-circuits is $\beta \cdot N \in O(N^{1+\gamma})$ because C_n has at most $N_O \in O(N)$ nodes. Since the constant γ was chosen such that $0 < \gamma \leq 1 - 2\varepsilon$, we have $N^{1+\gamma} \in O(N^{2(1-\varepsilon)})$ and thus all up-circuits and down-circuits can be stored in the respective reducers. ◀

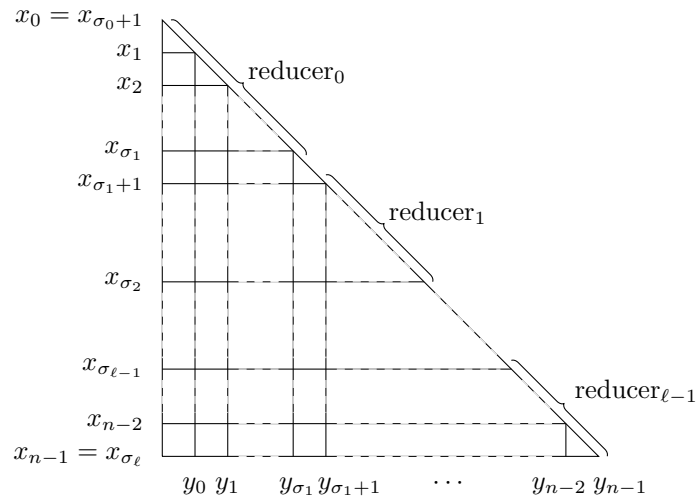
► **Lemma 19** (Reprint of Lemma 19 on page 11). *If all up-circuits and down-circuits are constructed in the proper reducers, C_n can be evaluated in $O(\text{depth}(C_n)/\log n)$ DMRC-rounds.*

Proof. Without loss of generality, let $\text{depth}(C_n)$ be divisible by s and let $r = \text{depth}(C_n)/s$. Once all L_i -down-circuits and L_i -up-circuits for all $i \in \{1, \dots, r\}$ have been constructed, we can evaluate C_n on the given input assignment. We begin by evaluating the L_{r-1} -down-circuits. Since every input node has its value assigned in a v -down-circuit, the L_{r-1} -down-circuits can be computed in the reducers containing these v -down-circuits. With the values of all nodes at level L_{r-1} determined, we can send the necessary values to the L_{r-2} -down-circuits and, in the case of edges that were divided using two dummy nodes, to lower-level down-circuits. Nodes at level L_{r-1} that are necessary to compute L_{r-2} -down-circuits are described in the L_{r-1} -up-circuits. Any node v at level L_{r-1} that is necessary to compute L_{r-2} -down-circuits is described in the v -up-circuit. Therefore, the output of the reducer q is as follows: Let v be at level L_{r-1} and let u_i , for $i \in \{1, \dots, k_v\}$, be the nodes at level L_{r-2} in the v -up-circuit. For each v in reducer q , it outputs $(\text{to}(u_i), v, \text{val}(v))$, where $\text{to}(u_i)$ is the index of the reducer containing the u_i -down-circuit and $\text{val}(v)$ is the value of v determined in the computation of the v -down-circuit. The reducer q also passes on all v -down-circuits and v -up-circuits contained in it.

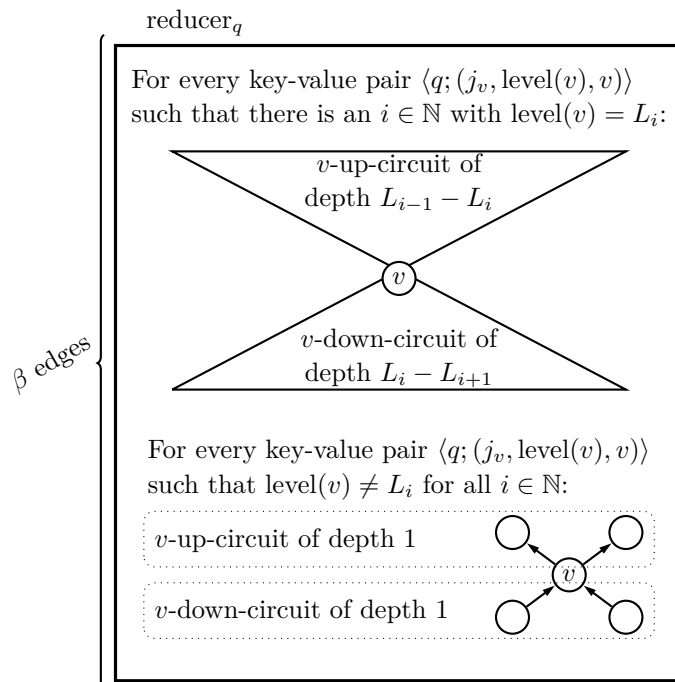
In the next round, the map function sends each $(\text{to}(u_{\text{in}}), v, \text{val}(v))$ to the reducer containing the u_{in} -down-circuit; that is, it generates the key-value pair $\langle \text{to}(u_{\text{in}}); (v, \text{val}(v)) \rangle$. Of course, the map function also passes along all v -down-circuits and v -up-circuits to the proper reducers.

Since now each L_{r-2} -down-circuit is contained completely in a reducer that has gathered all values of nodes at level L_{r-1} necessary to compute this subcircuit, all L_{r-2} -down-circuits can be computed in their reducers. Now we can compute the values of nodes higher and higher up in the circuit, by iterating the last mapping-reducing function pair, until the value is finally known for the unique output node. As before, we clearly stay within the memory and I/O buffer limits of each reducer. ◀

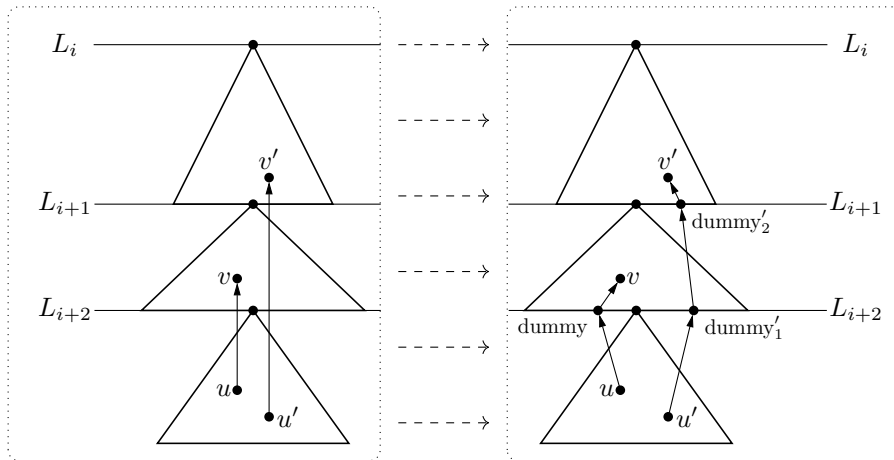
B Illustrating Figures



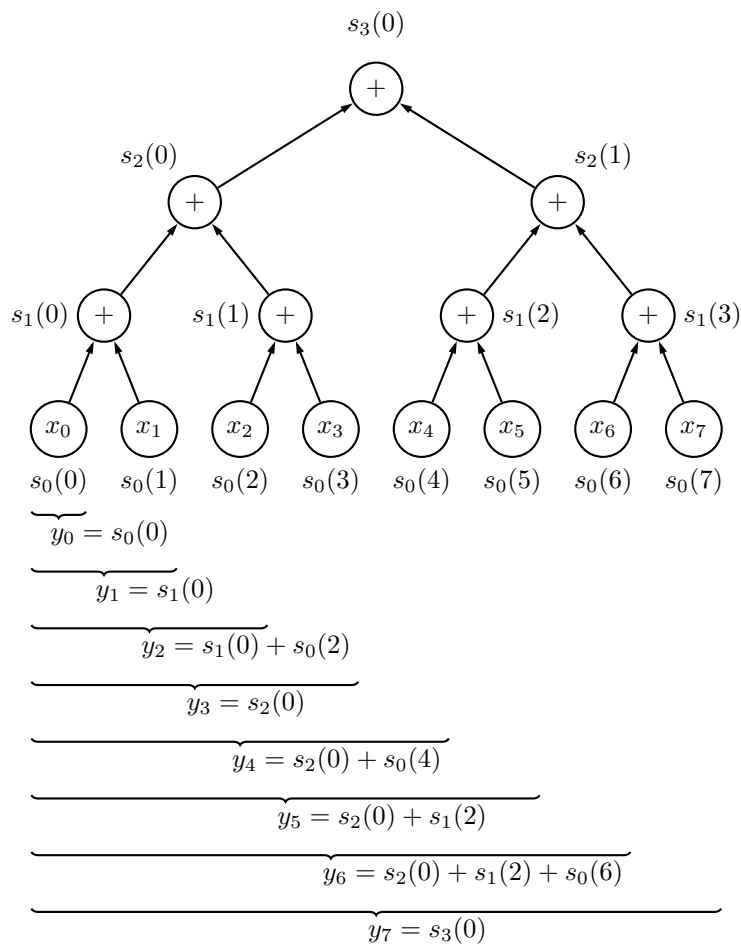
■ **Figure 1** Separation of the input variables x_0, \dots, x_{n-1} into ℓ blocks for the ℓ reducers, in dependence of the values of y_i .



■ **Figure 2** The up-circuits and down-circuits constructed in reducer_q , comprising up to β edges.



■ **Figure 3** Two jumping edges on the left and their resolving division on the right.



■ **Figure 4** Calculation of the prefix-sums $s_i(j) = \sum_{p \in [(j+1)2^i] \setminus [j2^i]} x_p$ for every $i \in [1 + \log q]$ and $j \in [q/2^i]$ for the example of $q = 8$.

Concurrent Distributed Serving with Mobile Servers

Abdolhamid Ghodselahi

Institute of Telematics, Hamburg University of Technology, Germany
abdolhamid.ghodselahi@tuhh.de

Fabian Kuhn

Department of Computer Science, University of Freiburg, Germany
kuhn@cs.uni-freiburg.de

Volker Turau

Institute of Telematics, Hamburg University of Technology, Germany
turau@tuhh.de

Abstract

This paper introduces a new resource allocation problem in distributed computing called *distributed serving with mobile servers (DSMS)*. In DSMS, there are k identical mobile servers residing at the processors of a network. At arbitrary points of time, any subset of processors can invoke one or more requests. To serve a request, one of the servers must move to the processor that invoked the request. Resource allocation is performed in a distributed manner since only the processor that invoked the request initially knows about it. All processors cooperate by passing messages to achieve correct resource allocation. They do this with the goal to minimize the communication cost.

Routing servers in large-scale distributed systems requires a scalable location service. We introduce the distributed protocol GNN that solves the DSMS problem on *overlay trees*. We prove that GNN is starvation-free and correctly integrates locating the servers and synchronizing the concurrent access to servers despite asynchrony, even when the requests are invoked over time. Further, we analyze GNN for “one-shot” executions, i.e., all requests are invoked simultaneously. We prove that when running GNN on top of a special family of tree topologies – known as *hierarchically well-separated trees (HSTs)* – we obtain a randomized distributed protocol with an expected competitive ratio of $O(\log n)$ on general network topologies with n processors. From a technical point of view, our main result is that GNN optimally solves the DSMS problem on HSTs for one-shot executions, even if communication is asynchronous. Further, we present a lower bound of $\Omega(\max\{k, \log n / \log \log n\})$ on the competitive ratio for DSMS. The lower bound even holds when communication is synchronous and requests are invoked sequentially.

2012 ACM Subject Classification Theory of computation → Online algorithms; Theory of computation → Distributed algorithms; Theory of computation → Graph algorithms analysis; Theory of computation → Discrete optimization

Keywords and phrases Distributed online resource allocation, Distributed directory, Asynchronous communication, Amortized analysis, Tree embeddings

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.53

Related Version A full version of this paper is available at <https://arxiv.org/abs/1902.07354> [14].

Funding This work is supported by the Deutsche Forschungsgemeinschaft (DFG), under grant DFG TU 221/6-3.



© Abdolhamid Ghodselahi, Fabian Kuhn, and Volker Turau;
licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 53; pp. 53:1–53:18

Leibniz International Proceedings in Informatics



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

1 Introduction

Consider the following family of online resource allocation problems. We are given a metric space with n points. Initially, a set of $k \geq 1$ ¹ identical mobile servers are residing at different points of the metric space. Requests arrive over time in an online fashion, that is, one or several requests can arrive at any point of time. A request needs to be served by a server at the requesting point sometime after its arrival. The goal is to provide a schedule for serving all requests. This abstract problem lies at the heart of many centralized and distributed online applications in industrial planning, operating systems, content distribution in networks, and scheduling [3, 7, 8, 16, 21]. Each concrete problem of this family is characterized by a cost function. We study this abstract problem in distributed computing and call it the distributed serving with mobile servers (DSMS) problem. A distributed protocol ALG that solves the DSMS problem must compute a schedule for each server consisting of a queue of requests such that consecutive requests are successively served, and all requests are served. The k schedules are distributedly stored at the requesting nodes: each node knows for each of its requests the node which invoked the subsequent request in the schedule so that a server after serving one request can subsequently move to the next node (not necessarily a different node). As long as new requests are invoked the schedule is extended. Therefore, in response to the appearance of a new request at a given processor, ALG must contact a processor that invoked a request but yet has no successor request in the global schedule, to instruct the motion of the corresponding server. This will result in the entry of a server to the requesting processor. Sending a server from a processor to another one is done using an underlying routing scheme that routes most efficiently. The goal is to minimize the ratio between the communication costs of an online and an *optimal offline* protocols that solve DSMS. We assume that an optimal offline DSMS protocol OPT knows the whole sequence of requests in advance. However, OPT still needs to send messages from each request to its predecessor request. The DSMS problem has some interesting applications. We state two of them:

Distributed k -server problem

The k -server problem [5, 18], is arguably one of the most influential research problems in the area of online algorithms and competitive analysis. The distributed k -server was studied in [8] where requests arrive sequentially one by one, but only after the current request is served. The cost function for this problem is defined as the sum of all communication costs and the total movement costs of all servers. A generalization of the k -server problem where requests can arrive over time is called the online service with delay (OSD) problem [4, 9]. The OSD cost function is defined as the sum of the total movement costs of all servers and the total *delay cost*. The delay of a request is the difference between the service and the arrival times.

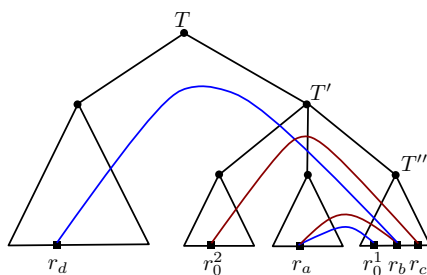
Distributed queuing problem

This problem is an application of DSMS with $k = 1$, i.e., only one server or shared object [10, 15, 16]. The distributed queuing problem is at the core of many distributed problems that schedule concurrent access requests to a shared object. The goal is to minimize the sum of the total communication cost and the total “waiting time”. The waiting time of a request is the difference between the times when the request message reaches the processor of the predecessor request and when the predecessor request is invoked. Note that in this problem,

¹ Table 1 provides an index for the essential notations used throughout the paper.

the processor of a request must only send one message to the processor of the predecessor request in the global schedule. Two well-known applications for this problem are distributed mutual exclusion [19, 21, 22] and distributed transactional memory [24].

Next, we explain why DSMS is also interesting from a theoretical point of view even for one-shot executions, that is, when all requests are simultaneously invoked. Figure 1 shows a rooted tree T , where the lengths of all edges of each level are equal. Further, the length of every edge is shorter than the length of its parent edge by some factor larger than one. A set of six requests arrive at the leaves of T at the same time. Two servers s_0, s_1 are initially located at the points that invoked requests r_0^1 and r_0^2 . Serving the requests r_0^1 and r_0^2 does not require communication, and these two requests are the current tails of the queues of s_0 and s_1 . The requests r_0^1 and r_0^2 are at the heads of the two queues. An optimal solution for serving the remaining requests is that s_0 consecutively serves the requests r_b, r_c , and r_a after serving r_0^1 , while s_1 serves r_d after having served r_0^2 . Next, consider an asynchronous network where, in contrast with a synchronous network where there is a global clock, message latencies are arbitrary and protocols have no control over these latencies. A possible schedule, in this case, is shown in Figure 1: Request r_a is scheduled after r_0^1, r_b after r_a , and r_d after r_b , since the message latency of a request further away can be much less than the latency of a closer request. This can lead to complications with regard to improving the *locality* as it is met in the above optimal solution.



■ **Figure 1** A distributed protocol may lead to complications with regard to improving locality.

GNN protocol

We devise the generalized nearest-neighbor (GNN) protocol that greedily solves the DSMS problem on **overlay trees**. An overlay tree T is a rooted tree that is constructed on top of the underlying network. The processors of the original network are in a one-to-one correspondence with the leaves of T . Hence, only T 's leaves can invoke requests, and the remaining overlay nodes are artificial. The k servers reside at different leaves of T . Initially, all edges of T are oriented such that from each leaf there is a directed path to a leaf, where a server resides. This also implies that every leaf node with a server has a self-loop. Roughly speaking, the main idea of GNN is to update the directions of edges with respect to future addresses of a server. A leaf invoking a request forwards a message along the directed links, the orientations of all these links are inverted. When a message reaches a node and finds several outgoing (upward/downward) links, it is forwarded via an arbitrary downward link to find the current or a future address of a server. We show that in GNN a processor holding a request always sends a message through a direct path to the processor of the predecessor request in the global schedule. We refer to Section 3 for a formal description of GNN.

1.1 Our Contribution

This paper introduces the DSMS problem as a distributed online allocation problem. We devise the greedy protocol GNN that solves the DSMS problem on overlay trees. We prove that even in an asynchronous system GNN operates correctly, that is, it does not suffer from starvation, nor livelocks, or deadlocks. To the best of our knowledge, GNN is the first link-reversal-based protocol that supports navigating more than one server.

► **Theorem 1.** *Suppose the overlay tree T is constructed on top of a distributed network. Consider the DSMS problem on T where a set of $k \geq 1$ identical mobile servers are initially located at different leaves of T . Further, a sequence of requests can be invoked at any time by the leaves of T . Then GNN schedules all requests to be served by some server at the requested points in a finite time despite asynchrony.*

While GNN itself solves any instance of the DSMS problem, we analyze GNN for the particular case that the requests are simultaneously invoked. We consider general distributed networks with n processors. We model such a network by a graph G . A hierarchically well-separated tree (HST) is an overlay tree with parameter $\alpha > 1$, that is, an α -HST is a rooted tree where every edge weight is shorter by a factor of α from its parent edge weight. A tree is an HST if it is an α -HST for some $\alpha > 1$. There is a randomized embedding of any graph into a distribution over HSTs [6, 11]. We sample an HST T according to the distribution defined by the embedding. We consider an instance I of the DSMS problem where the communication is asynchronous, and the requests are simultaneously invoked by the nodes of G . When running GNN on T , we get a randomized distributed protocol on G that solves I with an expected competitive ratio of $O(\log n)$ against oblivious adversaries².

► **Theorem 2.** *Let I denote an instance of the DSMS problem consisting of an asynchronous network with n processors and a set of requests that are simultaneously invoked by processors of the network. There is a randomized distributed protocol that solves I with an expected competitive ratio of $O(\log n)$ against an oblivious adversary.*

Consider an instance I of DSMS that consists of an HST T where communication is asynchronous and a set of requests that are simultaneously invoked by the leaves of T . Analyzing GNN for I turns out to be involved and non-trivial. The fact that the GNN (as any other protocol) has no control on the message latencies bears a superficial resemblance to the case where the requests are invoked over time. Hence, when analyzing GNN for I , one faces the following complications: 1) A server may go back to a subtree of T after having left it. 2) A request in a subtree of T that initially hosts at least one server can be served by a server that is initially outside this subtree. 3) Different servers can serve two requests in a subtree of T that does not initially host any server. Theorem 2 is derived from our main technical result for HSTs.

► **Theorem 3.** *Consider an instance I of DSMS that consists of an HST T where even the communication is asynchronous and a set of requests that are simultaneously invoked by the leaves of T . The GNN protocol optimally solves I .*

One-shot executions of the distributed queuing problem for synchronous communication were already considered in [16]. The following corollary follows from Theorem 3.

► **Corollary 4.** *GNN optimally solves the distributed queuing problem on HSTs for one-shot executions even when the communication is asynchronous.*

² This assumes that the sequence of requests is statistically independent of the randomness used for constructing the given tree.

We provide a simple reduction from the distributed k -server problem to the DSMS problem. Our following lower bound is obtained using this reduction and an existing lower bound [8] on the competitive ratio for the distributed k -server problem.

► **Theorem 5.** *There is a network topology with n processors – for all n – such that there is no online distributed protocol solving DSMS with a competitive ratio of $o(\max\{k, \log n / \log \log n\})$ against adaptive online adversaries where k is the number of servers. This result even holds when requests are invoked one by one by processors in a sequential manner and even when the communication is synchronous.*

2 Model, Problem Statement, and Preliminaries

2.1 Communication Model

We consider a point-to-point communication network that is modeled by a graph $G = (V, E)$, where the n nodes in V represent the processors of the network and the edges in E represent bidirectional communication links between the corresponding processors. We suppose that the edge weights are positive and are normalized such that the weight of each edge will be at least 1. If G is unweighted, then we assume that the weight of an edge is 1. We consider the message passing model [20] where neighboring processors can exchange messages with each other. The communication links can have different latencies. These latencies are not even under control of an optimal offline distributed protocol. We consider both synchronous and asynchronous systems. In a synchronous system, the latency for sending a message over an edge equals the weight of the edge. In an asynchronous system, in contrast, the messages arrive at their destinations after a finite but unbounded amount of time. Messages that take a longer path may arrive earlier, and the receiver of a message can never distinguish whether a message is still in transit or whether it has been sent at all. For our analysis, however, we adhere to the conventional approach where the latencies are scaled such that the latency for sending a message over an edge is upper bounded by the edge weight in the “worst case” (for every legal input and in every execution scenario) (see Section 2.2 in [20] for more information).

2.2 Distributed Serving with Mobile Servers (DSMS) Problem

The input for DSMS problem for a graph G consists of $k \geq 1$ identical mobile servers that are initially located at different nodes of G and a set \mathcal{R} of requests that are invoked at the nodes at any time. A request $r_i \in \mathcal{R}$ is represented by (v_i, t_i) where node v_i invoked request r_i at time $t_i \geq 0$. A distributed protocol ALG that solves the DSMS problem needs to serve each request with one of the k servers at the requested node. Hence, ALG must schedule all requests that access a particular server. Consequently, ALG outputs k global schedules. Let π_{ALG}^z where $z \in \{1, \dots, k\}$ denote the z -th schedule generated by ALG, and s^z be the z -th server. The request sets of these k schedules form a partition of \mathcal{R} , and all requests of the schedule π_{ALG}^z consecutively access the server s^z . We assume that at time 0, when an execution starts, the tail of schedule π_{ALG}^z is at a given node $v_0^z \in V$ that hosts s^z . Formally, this is modeled as a “dummy request” $r_0^z = (v_0^z, 0)$ that has to be scheduled first in the schedule π_{ALG}^z by ALG. Consider two requests r_i and r_j that are consecutively served by s^z where r_i is scheduled after r_j . To schedule request r_i the protocol needs to inform node v_j , the predecessor request r_j in the constructed schedule. As soon as r_j is served by s^z , node v_j sends the server to v_i for serving r_i using an underlying routing facility that efficiently routes messages. The goal is to minimize the total communication cost, i.e., the sum of the latencies of all messages sent during the execution of ALG.

2.3 Preliminaries

Consider a distributed protocol ALG for the DSMS problem when requests can arrive at any time. Let \mathcal{R} denote the set of requests, including the dummy requests. Assume that ALG partitions \mathcal{R} into k sets $\mathcal{R}_{\text{ALG}}^1, \dots, \mathcal{R}_{\text{ALG}}^k$, and that it schedules the requests in set $\mathcal{R}_{\text{ALG}}^z$ according to permutation π_{ALG}^z . Denote the request at position i of π_{ALG}^z by $r_{\pi_{\text{ALG}}^z(i)}$. The dummy request r_0^z of π_{ALG}^z is represented by $r_{\pi_{\text{ALG}}^z(0)}$. Consider a message denoted by μ . Let $\ell_{\text{ALG}}(\mu)$ denote the latency of message μ as routed by ALG. For every $i \in \{1, \dots, |\mathcal{R}| - 1\}$, if r_i belongs to $\mathcal{R}_{\text{ALG}}^z$, the communication cost $c_{\text{ALG}}(r_{\pi_{\text{ALG}}^z(i-1)}, r_{\pi_{\text{ALG}}^z(i)})$ incurred for scheduling $r_{\pi_{\text{ALG}}^z(i)}$ as the successor of $r_{\pi_{\text{ALG}}^z(i-1)}$ is the sum of the latencies of all messages sent by ALG to schedule $r_{\pi_{\text{ALG}}^z(i)}$ immediately after $r_{\pi_{\text{ALG}}^z(i-1)}$. The total communication cost of ALG for scheduling all requests in $\mathcal{R}_{\text{ALG}}^z$ is defined as

$$C_{\text{ALG}}(\pi_{\text{ALG}}^z) := \sum_{i=1}^{|\mathcal{R}_{\text{ALG}}^z|-1} c_{\text{ALG}}(r_{\pi_{\text{ALG}}^z(i-1)}, r_{\pi_{\text{ALG}}^z(i)}). \quad (1)$$

The total communication cost of ALG for scheduling all requests in \mathcal{R} , therefore, is

$$C_{\text{ALG}} := \sum_{z=1}^k C_{\text{ALG}}(\pi_{\text{ALG}}^z). \quad (2)$$

2.4 Hierarchically Well-Separated Trees (HSTs)

Embedding of a metric space into probability distributions over tree metrics have found many important applications in both centralized and distributed settings [4, 5, 13]. The notion of a hierarchically well-separated tree was defined by Bartal in [6].

► **Definition 6 (α -HST).** For $\alpha > 1$ an α -HST of depth h is a rooted tree with the following properties: The children of the root are at a distance α^{h-1} from the root and every subtree of the root is an α -HST of depth $h - 1$. A tree is an HST if it is an α -HST for some $\alpha > 1$.

The definition implies that the nodes two hops away from the root are at a distance α^{h-2} from their parents. The probabilistic tree embedding result of [11] shows that for every metric space (X, d) with minimum distance normalized to 1 and for every constant $\alpha > 1$ there is a randomized construction of an α -HST T with a bijection f between the points in X and the leaves of T such that a) the distances on T are dominating the distances in the metric space (X, d) , i.e., $\forall x, y \in X : d_T(f(x), f(y)) \geq d(x, y)$ and such that b) the expected tree distance is $\mathbb{E}[d_T(f(x), f(y))] = O(\alpha \log |X| / \log \alpha) \cdot d(x, y)$ for every $x, y \in X$. The length of the shortest path between any two leaves u and v of T is denoted by $d_T(u, v)$. An efficient distributed construction of the probabilistic tree embedding of [11] has been given in [12].

3 The Distributed GNN Protocol

In this section the GNN protocol is introduced.

3.1 Description of GNN

GNN runs on overlay trees and outputs a feasible solution for the DSMS problem. Consider a rooted tree $T = (V_T, E_T)$ whose leaves correspond to the nodes of the underlying graph $G = (V, E)$, i.e., $V \subseteq V_T$. Let $n = |V|$. The $k \geq 1$ identical mobile servers are initially at different leaves of T . Further, there is a dummy request at every leaf that initially hosts a

server. The leaves of T can invoke requests at any time. A leaf node can invoke a request while it is hosting a server and a leaf can also invoke a request while its previous requests have not been served yet. Initially, a directed version of T is constructed and denoted by H , the directed edges of H are called *links*. During an execution of GNN, GNN changes the directions of the links. Denote by $v.links$ the set of neighbors of v that are pointed by v . After a leaf u has invoked a request it sends a find-predecessor message denoted by $\mu(u)$ along the links to inform the node of the predecessor request in the global schedule. The routing of $\mu(u)$ is explained below. At the beginning before any message is sent and for any server, all the nodes on the direct path from the root of T to the leaf that hosts the server, point to the server. Further, the host points to itself and creates a self-loop. Hence, we have k directed paths with downward links from the root of T to the points of the current tails of the schedules. Any other node points to its parent with an upward link. Therefore, the sets $v.links$ for all $v \in V_T$ are non-empty at the beginning of the executing the protocol. Figure 2a shows the directed HST at the beginning as an example.

■ **Algorithm 1** GNN Protocol.

Input : The rooted tree T , k identical mobile servers that are initially at distinct leaves of T , and a set of requests that are invoked over time

Output : k schedules for serving all requests

Upon requesting a service: Algorithm 2

Upon receiving a find-predecessor message: Algorithm 3

Upon u invoking a new request

Consider the leaf node u when it invokes a new request r . If u has a self-loop, then r is scheduled immediately after the last request that has been invoked at u . Otherwise, the leaf u atomically sends $\mu(u)$ to its parent through an upward link, u points to itself, and the link from u to its parent is removed. We suppose that messages are reliably delivered. The details of this part of the protocol are given by Algorithm 2. See Figure 2b as an example.

■ **Algorithm 2** Upon u invoking a new request r .

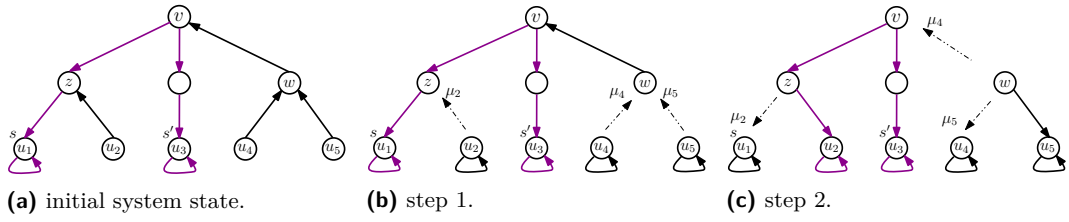
```

1 do atomically
  /* suppose  $u.links = \{v\}$  ( $u$  as a leaf always points either to itself or
    to its parent) */
2   if  $u = v$  then
3      $r$  is scheduled immediately after the last request that has been invoked by  $u$ 
4   else
5      $u$  sends  $\mu(u)$  to  $v$ 
6      $u.links := \{u\}$ 
7   end
8 end

```

Upon w receiving $\mu(u)$ from node v

Suppose that node w receives a find-predecessor message $\mu(u)$ from node v . The node w executes the following steps atomically. If w has at least one downward link, then $\mu(u)$ is forwarded to some child of w through a downward link (ties are broken arbitrarily). Then, w removes the downward link and adds a link to v – independently of whether v is the



■ **Figure 2** GNN protocol: (a) The servers s and s' serve requests in schedules π and π' , respectively. The dummy requests at u_1 and u_3 are the initial tails of π and π' , respectively. (b) Nodes u_2 , u_3 , u_4 , and u_5 respectively issue requests r_2 , r_3 , r_4 , and r_5 at the same time and send the find-predecessor messages μ_2 , μ_3 , μ_4 , and μ_5 , respectively, along the arrows. (c) The request r_3 is the current tail of π' . Both μ_4 and μ_5 reach w at the same time. First, the message μ_4 is arbitrarily processed by w and w forwards μ_4 towards v and therefore μ_5 is deflected towards u_4 .

parent or a child of w . If w does not have a downward link, it either points to itself, or it has an upward link. In the latter case, $\mu(u)$ is atomically forwarded to the parent of w , the upward link from w to its parent is removed and then w points to v using a downward link. Otherwise, w is a leaf and points to itself. The request r invoked by u is scheduled after the last request that has been invoked by w . Then, w removes the link that points to itself and points to v using an upward link. The details of this part of the protocol are given by Algorithm 3. Also, see Figure 2c and Figure 3.

■ **Algorithm 3** Upon w receiving $\mu(u)$ from node v ($w \neq v$).

```

1 do atomically
2   if there exists a child node in  $w.links$  then
3      $z :=$  an arbitrary child node in  $w.links$ 
4   else
5      $z :=$  the only node in  $w.links$ 
6   end
7    $w.links := w.links - \{z\}$ 
8    $w.links := w.links \cup \{v\}$ 
9   if  $z \neq w$  then
10     $w$  sends  $\mu(u)$  to  $z$ 
11  else
12    the corresponding request to  $\mu(u)$  is scheduled immediately after the last
    request that has been invoked by  $w$ 
13  end
14 end

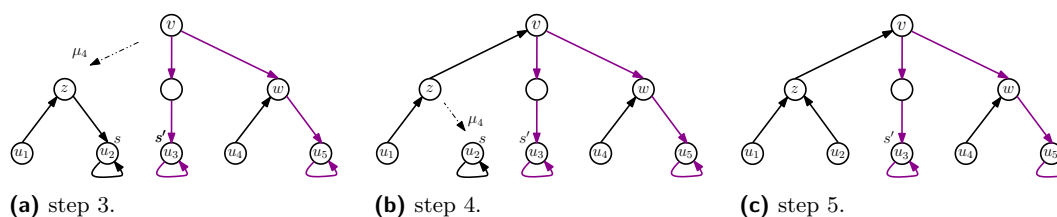
```

3.2 Correctness of GNN

Regarding the description of GNN, we need to show two invariants for GNN. The first is that GNN eventually schedules all requests. The second one is that GNN is starvation-free so that a scheduled request is eventually served.

3.2.1 Scheduling Guarantee

► **Theorem 7.** GNN guarantees that the find-predecessor message of any node that invokes a request always reaches a leaf node v in a finite time such that $v.links = \{v\}$.



■ **Figure 3** GNN protocol: (a) The request r_2 is scheduled after the current tail of π and now r_2 is the current tail of π and u_2 obtains the server s . The request r_5 is scheduled after r_4 while μ_4 is still in transit. (b) μ_4 still follows arrows, reversing the directions of arrows along its way. (c) The request r_4 is scheduled after r_2 and s moves to u_4 . After r_4 served by s , then s moves from u_4 to u_5 since r_5 has already been scheduled after r_4 . Figure 2–Figure 3 illustrates that there is always at least one connected path with purple arrows from the root to some leaf.

We prove the scheduling guarantee stated in Theorem 7 using the following properties of GNN. First, we need to show that any node always has at least one outgoing edge in GNN.

► **Lemma 8.** *In GNN, $v.links$ is never empty for any node $v \in V_T$.*

Proof. At the beginning of any execution, $v.links$ is not empty for any $v \in V_T$. The set $v.links$ changes only when there is a (find-predecessor) message at v (see Line 6 of Algorithm 2 and Line 7 and Line 8 of Algorithm 3). During an execution, every time v receives a message, a node is removed from $v.links$ while a new node is added to $v.links$. This also covers the case when at least two messages are received by v at the same time. The node v atomically processes all these messages in an arbitrary order. Therefore, $v.links$ never gets empty. ◀

► **Lemma 9.** *GNN always guarantees that on each edge of H , there is either exactly one link or exactly one message in transit.*

Proof. Initially, either a node points to its parent with an upward link or a node points to its children with downward links in the GNN protocol. Consider the edge (u, v) where $v \in u.links$. Further, consider the first time in which a message is in transit on (u, v) . Immediately before this transition occurs, u must point to v , and there is not any message in transit on the edge. Therefore, w.r.t. the protocol description, the message must be sent by u to v , and the link that points from u to v has been removed. Since there is not any link while the message is in transit, it is not possible to have a second message to be in transit at the same time. When the message arrives at v , the node v points to u , and the message is removed from the edge. The next time, if a message will be transited on the edge, then v must have sent it to u and removed the link that points from v to u . ◀

► **Lemma 10.** *The directed tree H always remains acyclic during an execution, hence a path from a node to another node in H is always the direct path.*

Proof. The GNN protocol runs on the directed tree H in which the underlying tree – that is, T – is fixed, and the directions of links on H are only changed. Therefore, H is acyclic because the tree is always fixed, and w.r.t. Lemma 9 that shows that it never occurs a state where on the edge (u, v) , u and v point to each other at the same time. ◀

The following lemma implies that a find-predecessor message always reaches the node of its predecessor using a direct path constructed by GNN.

► **Lemma 11.** *GNN guarantees that there is always at least one direct path in H from any leaf node u to a leaf node v where $v.links = \{v\}$.*

Proof. If the leaf node u points to itself, we are done. Otherwise, w.r.t. Lemma 8 there must be a path from u to a leaf node v since the tree H is acyclic. This path must be a direct path w.r.t. Lemma 10. The leaf node v must point to itself w.r.t. Lemma 8. ◀

Proof of Theorem 7. Using Lemma 11, it remains to show that any message traverses in finite time a direct path between two leaves. The number of edges on the direct path between any two leaves of T is upper bounded by the diameter of the tree. Further, any message that is in transit at edge (u, v) from u to v is delivered reliably at v in a finite time. Therefore, to show that a request is eventually scheduled in a finite time, it remains to show that a message will never be at a node for the second time. To obtain a contradiction, assume that the message μ is the first message that visits a node twice, and the first node visited twice by μ denoted by $v \in V_T$. With respect to Lemma 10, there is never a cycle in H . Therefore, the edge $e = (u, v)$ must be the first edge that is traversed by μ first from v to u and immediately from u to v for the second time, and μ must be the first message that traverses an edge twice. This implies that immediately before u receives μ , the node u points to v , and μ is in transit on e at the same time. This contradicts Lemma 9. ◀

3.2.2 Serving Guarantee

► **Theorem 12.** *GNN is starvation-free. In other words, any scheduled request is eventually served by some server.*

Consider any of k global schedules that produced by GNN, say π_{GNN}^w . Assume that there is more than one request scheduled in π_{GNN}^w . For any two requests $r_i = (v_i, t_i)$ and $r_j = (v_j, t_j)$ in π_{GNN}^w where r_i is scheduled immediately before r_j , we see $e = (r_i, r_j)$ as a directed edge where r_j points to r_i . This edge is actually simulated by the direct path – by Lemma 11, a message always finds the node of its predecessor using a direct path on H – between the leaves v_i and v_j that is traversed by the message sent from v_j to v_i . Let F_{ALG}^w denote the graph constructed by the messages of all requests in $\mathcal{R}_{\text{ALG}}^w$.

► **Lemma 13.** *F_{ALG}^w is a directed path towards the head of the schedule, that is, $r_0^w = r_{\pi_{\text{GNN}}^w(0)}$.*

Proof. The proof has three parts.

- 1) **Any node of F_{Alg}^w , except the dummy request, has exactly one outgoing edge:** This is obvious since any node that invokes a request sends exactly one message.
- 2) **Any node in F_{Alg}^w has at most one incoming edge:** For the sake of contradiction, assume that there is a node contained in F_{ALG}^w denoted by $r = (v, t)$ with at least two incoming edges in F_{ALG}^w . This implies that two messages must reach v in H before v invokes any other request after r . However, when the first message reaches v – if any other message does not reach v before these two messages – v removes the link that points to itself and adds a link that points to its parent w.r.t. Line 7 and Line 8 of Algorithm 3. The second message cannot reach v as long as at least one request is invoked by v after invoking r . This contradicts our assumption in which two messages reach v before the time when v invokes another request after invoking r .
- 3) **F_{Alg}^w is connected:** To obtain a contradiction, assume that the graph F_{ALG}^w is not connected. Hence, w.r.t. the first and second parts, we have at least one connected component with at least two requests in $\mathcal{R}_{\text{ALG}}^w$ that form a cycle, and the connected component does not include the dummy request in r_0^w . Let $\mathcal{R}_{\text{ALG}}^{w,c}$ denote the requests in the connected component $F_{\text{ALG}}^{w,c}$ that forms a cycle. Consider the node z in V_T that is the lowest common ancestor of those leaves of H that invoke the requests in $\mathcal{R}_{\text{ALG}}^{w,c}$. Further, let the subtree $H^{w,c}$ of H denote the tree rooted at z . All messages of requests in $\mathcal{R}_{\text{ALG}}^{w,c}$ must traverse inside $H^{w,c}$ since $F_{\text{ALG}}^{w,c}$ is disconnected with any request in $\mathcal{R}_{\text{ALG}}^w \setminus \mathcal{R}_{\text{ALG}}^{w,c}$.

Assume that at least one message of requests in $\mathcal{R}_{\text{ALG}}^{w,c}$ reaches z . Consider the first message μ by r that reaches z at time t . If there is not any downward link at z at t , then μ is forwarded to the parent of z . This is a contradiction with the fact that $F_{\text{ALG}}^{w,c}$ is disconnected with any request in $\mathcal{R}_{\text{ALG}}^w \setminus \mathcal{R}_{\text{ALG}}^{w,c}$. Hence, there must be at least one downward link at z at t . On the other hand, since μ is the first message of requests in $\mathcal{R}_{\text{ALG}}^{w,c}$ that reaches z , all downward links at z at time t must have been created by some messages of requests in $H^{w,c}$ that are not in $\mathcal{R}_{\text{ALG}}^{w,c}$. Note that if a downward link at z is there since the beginning, then we assume that, w.l.o.g., it has been created by a “virtual message” sent by the node of the corresponding dummy request. Suppose μ is forwarded through one of these downward links that was created by the message of r' – as mentioned, r' can be a dummy request – that is in $H^{w,c}$ but not in $\mathcal{R}_{\text{ALG}}^{w,c}$. The original downward path from z to the leaf node of r' can be changed by the message of a request in $H^{w,c}$ – can be a request in $\mathcal{R}_{\text{ALG}}^{w,c}$. Thus, either r is scheduled immediately after some request in $H^{w,c}$ that is not in $\mathcal{R}_{\text{ALG}}^{w,c}$ or some other request in $\mathcal{R}_{\text{ALG}}^{w,c}$. In either case, we get a contradiction with our assumption in which $F_{\text{ALG}}^{w,c}$ is disconnected with any request in $\mathcal{R}_{\text{ALG}}^w \setminus \mathcal{R}_{\text{ALG}}^{w,c}$.

If there is not any message of a request in $\mathcal{R}_{\text{ALG}}^{w,c}$ that can reach z , then there must be at least two downward links during the execution at z that have been created by some messages of requests that are not in $\mathcal{R}_{\text{ALG}}^{w,c}$ – this holds because if there is at most one downward link at z , then a message of some request in $\mathcal{R}_{\text{ALG}}^{w,c}$ must reach z w.r.t. the definition of z . However, the existence of at least two downward links at z implies that $F_{\text{ALG}}^{w,c}$ is not connected. This is true because there are at least two downward paths that partition the requests in $\mathcal{R}_{\text{ALG}}^{w,c}$ into two disjoint components in F_{ALG}^w w.r.t. the definition of z and our assumption in which there is not any message of request in $\mathcal{R}_{\text{ALG}}^{w,c}$ that can reach z . This is a contradiction with our assumption in which $F_{\text{ALG}}^{w,c}$ is a connected component. The above three parts all altogether show that F_{ALG}^w is indeed a directed path that points towards the dummy request in $\mathcal{R}_{\text{ALG}}^w$. ◀

Proof of Theorem 12. Consider any of the k global schedules resulting from GNN, say π_{GNN}^w . If there is only one request in π_{GNN}^w – there must be at least one request, that is the dummy request r_0^w – then we are done. Otherwise, w.r.t. Lemma 13 there is a path of directed edges such as $e = (r_i, r_j)$ over the requests in $\mathcal{R}_{\text{ALG}}^w$. When v_i obtains a server, and after r_i is served, v_i sends the server to v_j for serving r_j using an underlying routing scheme. Consequently, all requests in $\mathcal{R}_{\text{ALG}}^w$ are served. ◀

Proof of Theorem 1. Theorem 7 and Theorem 12 together prove the claim of the theorem. ◀

4 Analysis in a Nutshell

From a technical point of view, we achieve our main result on HSTs. In this section, we provide an analysis of GNN on HSTs in a nutshell. The complete analysis, including all proofs appears in Section 5 of [14]. Our analysis of GNN for general networks appears in Section 5.4 of [14]. The lower bound claimed in Theorem 5 is proved in Section 6 of [14].

Let ALG denote a particular distributed DSMS protocol that sends a unique message from the node of a request to the node of the predecessor request for scheduling the request (the message can be forwarded by many nodes on the path between the two nodes of the predecessor and successor requests). Consider a one-shot execution of ALG where requests are invoked at the same time 0. Let $G = (V, E)$ denote the input graph. Further, let

$B = (V_B = \mathcal{R}, E_B = \binom{\mathcal{R}}{2})$ be the complete graph, and consider two requests $r = (v, 0)$ and $r' = (v', 0)$ in \mathcal{R} where $v, v' \in V$. Assume that r' is scheduled as the successor of r by ALG in the global schedule, and w.r.t. the DSMS problem definition ALG informs v by sending the (find-predecessor) message μ' from v' to v . Therefore, the communication cost for scheduling r' equals the latency of μ' . Formally,

$$c_{\text{ALG}}(r, r') = \ell_{\text{ALG}}(\mu'). \quad (3)$$

Let $r_{\text{src}}(\mu') = r'$ denote the request corresponding with μ' . Further, let $r_{\text{des}}(\mu') = r$ denote the predecessor request r in the global schedule. We see $e = (r, r')$ as an edge in E_B that is constructed by μ' . Let us add $\mu(e)$ and $e(\mu)$ to the notation where $\mu(e)$ is the message that constructs the edge e and $e(\mu)$ is the edge that is constructed by μ . For instance, here, $\mu(e)$ refers to μ' and $e(\mu')$ refers to the edge (r, r') .

Representing the solution of ALG as a forest

We observe that any of the k resulting schedules $\pi_{\text{ALG}}^1, \dots, \pi_{\text{ALG}}^k$ can be seen as a **TSP path** that spans all requests in the corresponding schedule as follows (see Lemma 13). The TSP path F_{ALG}^z starts with the dummy request r_0^z that is the head of π_{ALG}^z , and a request on the TSP path F_{ALG}^z is connected using an edge to its successor in the schedule π_{ALG}^z . As mentioned, the edge is constructed by the message sent by the requesting node to the node of its predecessor request. Therefore, an edge of any TSP path – that is an edge in E_B – is actually a path on the input graph that is traversed by the corresponding message. For any $F \subseteq \mathcal{F}_{\text{ALG}}$, we define the **total communication cost of F** as follows.

$$L_{\text{ALG}}(F) := \sum_{e \in F} \ell_{\text{ALG}}(\mu(e)). \quad (4)$$

Therefore, the **total communication cost of a TSP path** equals the sum of latencies of all messages that construct the TSP path. The k TSP paths represent a forest of B . Let \mathcal{F}_{ALG} be the forest that consists of the k TSP paths $F_{\text{ALG}}^1, F_{\text{ALG}}^2, \dots, F_{\text{ALG}}^k$ constructed by ALG. We slightly abuse notation and identify a subgraph F of $B = (\mathcal{R}, \binom{\mathcal{R}}{2})$ with the set of edges contained in F . The **total communication cost of \mathcal{F}_{ALG}** equals the sum of total costs of the k TSP paths $F_{\text{ALG}}^1, F_{\text{ALG}}^2, \dots, F_{\text{ALG}}^k$. For the input graph $G = (V, E)$, we denote the **weight of edge** $e = (r, r') \in E_B$ by $w_G(e) := d_G(v, v')$ where $v, v' \in V$ (recall $r = (v, t)$ and $r' = (v', t')$). Note that $d_G(v, v')$ is the weight of the shortest path between v and v' on the input graph G . Generally, the **total weight of the subgraph F** of B w.r.t. the input graph G equals the sum of weights of all edges in F . Formally,

$$W_G(F) := \sum_{e \in F} w_G(e). \quad (5)$$

► **Definition 14 (S-Respecting m -Forest).** *Let $G = (V, E)$ be a graph and $m \leq |V|$. A forest \mathcal{F} of G is called an m -forest if \mathcal{F} consists of m trees. Further, let $S \subseteq V$, $|S| \leq m$ be a set of at most m nodes. An m -forest \mathcal{F} of G is S -respecting if the nodes in S appear in different trees of \mathcal{F} .*

Let \mathcal{R}_D denote the set of k dummy requests in \mathcal{R} . W.r.t. the Definition 14, \mathcal{F}_{ALG} is an \mathcal{R}_D -respecting spanning k -forest of $B = (\mathcal{R}, \binom{\mathcal{R}}{2})$. From now on, we consider the HST T as the input graph.

Locality-based forest

For any subtree T' of T and any subgraph F of B , let $F(T')$ denote the subgraph of F that is induced by those requests contained in F that are also in T' . Further, let F^1, F^2, \dots, F^k denote the k trees of the spanning k -forest \mathcal{F} of B . Let \mathcal{F}_{GRD} be any \mathcal{R}_D -respecting spanning k -forest of B with the following basic **locality properties**.

- I. [**Intra-Component Property**] For any subtree T' of T and for any $w \in \{1, \dots, k\}$, the component $F_{\text{GRD}}^w(T')$ is a tree.
- II. [**Inter-Component Property**] For any subtree T' of T , suppose that there are at least two non-empty components $F_{\text{GRD}}^z(T')$ and $F_{\text{GRD}}^w(T')$ where $w \neq z$ and $w, z \in \{1, \dots, k\}$. Any of these components includes a dummy request.

We call such a forest a locality-based forest. Any locality-based forest is denoted by \mathcal{F}_{GRD} . The following theorem provides a general version of Theorem 3.

► **Theorem 15.** *Let I denote an instance of DSMS that consists of an HST T where the communication is asynchronous and a set \mathcal{R} of requests that are simultaneously invoked at leaves of T . The protocol ALG is optimal if the total cost of the resulting forest by ALG is upper bounded by the total weight of \mathcal{F}_{GRD} .*

4.1 Optimality of GNN on HSTs

Consider a one-shot execution of GNN, and suppose that \mathcal{F}_{GNN} is the resulting forest when running GNN on the given HST T w.r.t. the input sequence \mathcal{R} . With respect to Theorem 15, and the fact that GNN only sends one unique message for scheduling a request to its predecessor, it is sufficient to show that the forest \mathcal{F}_{GNN} can be transformed into a locality-based forest such that the total cost of \mathcal{F}_{GNN} is upper bounded by the total weight of \mathcal{F}_{GRD} . During an execution of GNN, the Intra or Inter-Component property can be violated (see Figure 1). Consider the following situations:

1. A server goes back to a subtree after the time when it leaves the subtree.
2. A request in a subtree of T that initially hosts at least one server is served by a server that is not initially in the subtree.
3. Two requests in a subtree of T that does not initially host any server, are served by different servers.

The first situation violates the Intra-Component property. Any of the second and the third situation violates the Inter-Component property. In the following, we characterize the Intra-Component and the Inter-Component properties by considering a timeline for the messages that enter and leave a subtree of T . Consider a message μ that enters the subtree T' of T . Another message can enter T' only after some message μ' has left T' after μ entered T' – the arrival times of messages μ and μ' at the root of T' can be the same (cf. Lemma 5.3 and Lemma 5.6 of [14]). Similarly, a message can leave T' after μ' left T' only after some message has entered T' after μ' left T' . We refer to Lemma 5.6 for more details. Consider a message μ that enters T' . The fact that μ enters T' implies that a server will leave T' for serving $r_{\text{src}}(\mu)$. Let μ' denote the first message that leaves T' after μ entered T' . Leaving μ' from T' implies that a server will enter T' for serving $r_{\text{src}}(\mu')$. If $r_{\text{src}}(\mu')$ is in the same TSP path of \mathcal{F}_{GNN} with $r_{\text{des}}(\mu)$, then the server that had served $r_{\text{des}}(\mu)$ goes back to T' for serving $r_{\text{src}}(\mu')$ after it left T' , and therefore the Intra-Component property is violated. Otherwise, the Inter-Component property is violated since two requests in T' are served by two different servers in which at least one of the servers is initially outside of T' . We say GNN makes an **Inter-Component gap** (μ, μ') on T' in the latter case and an **Intra-Component gap** (μ, μ') on T' in the former case.

Transformation

We transform \mathcal{F}_{GNN} through *closing the gaps* that are made by GNN on all subtrees of T . A message μ' can leave from several subtrees of T such that different messages enter the subtrees before μ' . Therefore, GNN can make different gaps with the same message μ' on this set of subtrees of T . We especially refer to Lemma 5.9 and Lemma 5.11 of [14] for more details on the gaps of the subtrees of T . We consider the lowest subtree in this set and let (μ, μ') be a gap on that. We **close the gap** (μ, μ') by removing $e(\mu')$ and by adding the new edge $(r_{\text{des}}(\mu), r_{\text{src}}(\mu'))$. In the example of Figure 1, for instance, the red edges are removed and the new edges (r_0^1, r_b) and (r_b, r_c) are added. When we close the gap (μ, μ') , all other gaps (μ'', μ') that are on higher subtrees are also closed. Therefore, we transform \mathcal{F}_{GNN} into a new forest \mathcal{F}_{mdf} by means of closing all gaps. The following lemma shows that \mathcal{F}_{mdf} is indeed the locality-based forest.

► **Lemma 16.** \mathcal{F}_{mdf} is an \mathcal{R}_D -respecting spanning k -forest of B that satisfies the Intra-Component and the Inter-Component properties.

It remains to show that the total cost of \mathcal{F}_{GNN} is upper bounded by the total weight of the new forest \mathcal{F}_{mdf} . Formally, we want to show that $L_{\text{GNN}}(\mathcal{F}_{\text{GNN}}) \leq W_T(\mathcal{F}_{\text{mdf}})$. Using Lemma 11, a message always finds the node of its predecessor using a direct path on T in any execution of GNN. Regarding to our communication model described in Section 2.1, therefore, for every edge $e \in \mathcal{F}_{\text{GNN}}$ we have

$$\ell_{\text{GNN}}(\mu(e)) \leq w_T(e) \quad (6)$$

Let (μ, μ') be the gap on the lowest subtree of T among all subtrees of T with gaps (μ'', μ') for any message μ'' that makes a gap with μ' . By closing the gap (μ, μ') , we remove $e^{\text{old}} := (r_{\text{src}}(\mu'), r_{\text{des}}(\mu'))$ and add the new edge $e^{\text{new}} := (r_{\text{src}}(\mu'), r_{\text{des}}(\mu))$. Using (6), we are immediately done if the latency of μ' is upper bounded by the weight of e^{new} . However, the latency of μ' can be larger than the weight of e^{new} . By contrast, the weight of e^{new} is lower bounded by the latency of μ (cf. Corollary 5.10 and Lemma 5.15 of [14]). This lower bound gives us the go-ahead to show that the weight of e^{new} can be seen as an ‘‘amortized’’ upper bound for $\ell_{\text{GNN}}(\mu')$. In the following, we provide **an overview of our amortized analysis** that appears in Section 5.3.3 of [14]. Let $E^{\text{new}} := \mathcal{F}_{\text{mdf}} \setminus \mathcal{F}_{\text{GNN}}$ and $E^{\text{old}} := \mathcal{F}_{\text{GNN}} \setminus \mathcal{F}_{\text{mdf}}$ be the sets of all edges that are added and removed during the transformation of \mathcal{F}_{GNN} , respectively. Further, we consider a set of edges that provides enough ‘‘potential’’ for our amortization.

$$E^{\text{pot}} := \{e \in \mathcal{F}_{\text{GNN}} : (\mu(e), \mu(e')) \text{ is a gap for some } e' \in E^{\text{old}}\}.$$

For every edge $e \in E^{\text{old}}$, let $E^{\text{pot}}(e) := \{e' \in E^{\text{pot}} : (\mu(e'), \mu(e)) \text{ is a gap}\}$. Further, for every edge $e \in E^{\text{pot}}$, let $E^{\text{old}}(e) := \{e' \in E^{\text{old}} : (\mu(e), \mu(e')) \text{ is a gap}\}$. In this overview, we consider the **simple case** where 1) $|E^{\text{old}}(e)| = 1$ for every edge $e \in E^{\text{pot}}$ and $|E^{\text{pot}}(e)| = 1$ for every edge $e \in E^{\text{old}}$. Further, 2) the sets E^{old} and E^{pot} do not share any edge. The execution provided by Figure 1 represents an example of the above simple case. We define the **potential function** $\Phi(F)$ for a subset F of \mathcal{F}_{GNN} as follows $\Phi(F) := W_T(F) - L_{\text{GNN}}(F)$. W.l.o.g., we assume that the edges in E^{old} are sequentially replaced with the edges in E^{new} . Hence, assume that e_i^{old} is replaced with e_i^{new} during the i -th replacement. Let also e_i^{pot} be the only edge in $E^{\text{pot}}(e_i^{\text{old}})$.

► **Lemma 17.** If $|E^{\text{old}}(e)| = 1$ for every edge $e \in E^{\text{pot}}$, $|E^{\text{pot}}(e)| = 1$ for every edge $e \in E^{\text{old}}$, and $E^{\text{old}} \cap E^{\text{pot}} = \emptyset$, then

$$w_T(e_i^{\text{old}}) \leq w_T(e_i^{\text{new}}) + \Phi(E^{\text{pot}} \setminus \{e_1^{\text{pot}}, \dots, e_{i-1}^{\text{pot}}\}) - \Phi(E^{\text{pot}} \setminus \{e_1^{\text{pot}}, \dots, e_i^{\text{pot}}\}) \quad (7)$$

for every $i \geq 1$.

Proof. Using the definition of the potential function Φ and the definitions of the total weight and the total communication cost of a subset of edges in \mathcal{F}_{GNN} , we have

$$\Phi(E^{\text{pot}} \setminus \{e_1^{\text{pot}}, \dots, e_{i-1}^{\text{pot}}\}) - \Phi(E^{\text{pot}} \setminus \{e_1^{\text{pot}}, \dots, e_i^{\text{pot}}\}) = w_T(e_i^{\text{pot}}) - \ell_{\text{GNN}}(e_i^{\text{pot}}).$$

Therefore, we need to show that $w_T(e_i^{\text{old}}) \leq w_T(e_i^{\text{new}}) + w_T(e_i^{\text{pot}}) - \ell_{\text{GNN}}(e_i^{\text{pot}})$. Let the subtree T' of T be the lowest subtree such that $(\mu(e_i^{\text{pot}}), \mu(e_i^{\text{old}}))$ is a gap on T' . This implies that $w_T(e_i^{\text{new}}) = \delta(T')$. On the other hand, using Lemma 5.15 of [14] we have $\ell_{\text{GNN}}(e_i^{\text{pot}}) \leq \delta(T') = w_T(e_i^{\text{new}})$. It remains to show that $w_T(e_i^{\text{old}}) \leq w_T(e_i^{\text{pot}})$. Let T_j'' be the highest subtree of T such that $(\mu(e_i^{\text{pot}}), \mu(e_i^{\text{old}}))$ is a gap on T_j'' and T_j'' is a child subtree of T'' . The message $\mu(e_i^{\text{old}})$ does not leave T'' since $E^{\text{pot}}(e_i^{\text{old}}) = \{e_i^{\text{pot}}\}$. Hence, $w_T(e_i^{\text{old}}) = \delta(T'')$. On the other hand, the fact that the message $\mu(e_i^{\text{pot}})$ enters T_j'' indicates that $w_T(e_i^{\text{pot}}) \geq \delta(T'')$. Consequently, $w_T(e_i^{\text{pot}}) \geq w_T(e_i^{\text{old}})$ and we are done. \blacktriangleleft

When we sum up (7) for all i , we get

$$W_T(E^{\text{old}}) \leq W_T(E^{\text{new}}) + \Phi(E^{\text{pot}}). \quad (8)$$

Using the definition of the potential function Φ and using $L_{\text{GNN}}(E^{\text{old}}) \leq W_T(E^{\text{old}})$ w.r.t (6), therefore we get $L_{\text{GNN}}(E^{\text{pot}}) + L_{\text{GNN}}(E^{\text{old}}) \leq W_T(E^{\text{new}}) + W_T(E^{\text{pot}})$. Hence, we have $L_{\text{GNN}}(\mathcal{F}_{\text{GNN}}) \leq W_T(\mathcal{F}_{\text{mdf}})$ since $\mathcal{F}_{\text{mdf}} = \mathcal{F}_{\text{GNN}} \setminus E^{\text{old}} \cup E^{\text{new}}$ and $L_{\text{GNN}}(\mathcal{F}_{\text{GNN}} \setminus (E^{\text{old}} \cup E^{\text{pot}})) \leq W_T(\mathcal{F}_{\text{GNN}} \setminus (E^{\text{old}} \cup E^{\text{pot}}))$ w.r.t (6).

► **Lemma 18.** *The total cost of \mathcal{F}_{GNN} is upper bounded by the total weight of \mathcal{F}_{mdf} .*

► **Theorem 19.** *The forest \mathcal{F}_{GNN} can be transformed into the locality-based forest \mathcal{F}_{GRD} such that the total cost of \mathcal{F}_{GNN} is upper bounded by the total weight of \mathcal{F}_{GRD} .*

5 Further Related Work

Distributed k -server problem

In Section 1, we have seen that the distributed k -server problem is an application of the DSMS problem. In [8], a general translator that transforms any deterministic global-control competitive k -server algorithm into a distributed competitive one is provided. This yields $\text{poly}(k)$ -competitive distributed protocols for the line, trees, and the ring synchronous network topologies. In [8], a lower bound of $\Omega(\max\{k, (1/D) \cdot (\log n / \log \log n)\})$ on the competitive ratio for the distributed k -server problem against adaptive online adversaries is also provided where n is the number of processors. D is the ratio between the cost to move a server and the cost to transmit a message over the same distance in synchronous networks. [4] and [9] study OSD on HSTs and lines, respectively. [4] provides an upper bound of $O(\log^3 n)$ and [9] provides an upper bound of $O(\log n)$ on the competitive ratio for OSD where n is the number of leaves of the input HST as well as the number of nodes of the input line.

Distributed queuing problem and link-reversal-based protocols

A well-known class of protocols has been devised based on link reversals to solve distributed problems in which the distributed queuing problem is at the core of them [2, 17, 19, 21, 22, 23, 24]. In a distributed link-reversal-based protocol nodes keep a link pointing to neighbors in the current or future direction of the server. When sending a message over an edge to request the server, the direction of the link flips. We devise the GNN protocol that is – to the best of our knowledge – the first link-reversal-based protocol that navigates more

than one server. A well-studied link-reversal-based protocol is called ARROW [19, 21, 22]. Several other tree-based distributed queueing protocols that are similar to ARROW have also been proposed. They operate on fixed trees. The RELAY protocol has been introduced as a distributed transactional memory protocol [24]. It is run on top of a fixed spanning tree similar to ARROW; however, to more efficiently deal with aborted transactions, it does not always move the shared object to the node requesting it. Further, in [2], a distributed directory protocol called COMBINE has been proposed. COMBINE like GNN runs on a fixed overlay tree, and it is in particular shown in [2] that COMBINE is starvation-free.

The first paper to study the competitive ratio of concurrent executions of a distributed queueing protocol is [16]. It shows that in synchronous executions of ARROW on a tree T for one-shot executions, the total cost of ARROW is within a factor $O(\log m)$ compared to the optimal queueing cost on where m is the number of requests. This analysis has later been extended to the general concurrent setting where requests are invoked over time. In [15], it is shown that in this case, the total cost of ARROW is within a factor $O(\log D)$ of the optimal cost on T where D is the diameter of T . Later, the same bounds have also been proven for RELAY [24]. Typically, these protocols are run on a spanning tree or an overlay tree on top of an underlying general network topology. In this case, the competitive ratio becomes $O(s \cdot \log D)$, where s is the stretch of the tree. Finally, [13] has shown that when running ARROW on top of HSTs, a randomized distributed online queueing protocol is obtained with expected competitive ratio $O(\log n)$ against an oblivious adversary even on general n -node network topologies. The result holds even if the queueing requests are invoked over time and even if communication is asynchronous. The main technical result of the paper shows that the competitive ratio of ARROW is constant on HSTs.

Online tracking of mobile users

A similar problem to DSMS is the online mobile user tracking problem [3]. In contrast with DSMS where a request r results in moving a server to the requesting point, here the request r can have two types: find request that does not result in moving the mobile user and move request. A request in DSMS that is invoked by v can be seen as a combination of a find request that is invoked at v in the mobile user problem and a move request invoked at the current address of the mobile user. The goal is to minimize the sum of the total communication cost and the total cost incurred for moving the mobile user. [3] provides an upper bound of $O(\log^2 n)$ on the competitive ratio for the online mobile user problem for one-shot executions. Further, [1] provides a lower bound of $\Omega(\log n / \log \log n)$ on the competitive ratio for this problem against an oblivious adversary.

References

- 1 N. Alon, G. Kalai, M. Ricklin, and L. Stockmeyer. Lower bounds on the competitive ratio for mobile user tracking and distributed job scheduling. In *FOCS*, 1992.
- 2 H. Attiya, V. Gramoli, and A. Milani. A provably starvation-free distributed directory protocol. In *SSS*, 2010.
- 3 B. Awerbuch and D. Peleg. Online tracking of mobile users. *Journal of the ACM*, 1995.
- 4 Y. Azar, A. Ganesh, R. Ge, and D. Panigrahi. Online Service with Delay. In *STOC*, 2017.
- 5 N. Bansal, N. Buchbinder, A. Madry, and J. S. Naor. A Polylogarithmic-Competitive Algorithm for the k -Server Problem. In *FOCS*, 2011.
- 6 Y. Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *FOCS*, 1996.

- 7 Y. Bartal, A. Fiat, and Y. Rabani. Competitive algorithms for distributed data management. In *STOC*, 1992.
- 8 Y. Bartal and A. Rosen. The distributed k -server problem—a competitive distributed translator for k -server algorithms. In *FOCS*, 1992.
- 9 M. Bienkowski, A. Kraska, and P. Schmidt. Online Service with Delay on a Line. In *SIROCCO*, 2018.
- 10 M. J. Demmer and M. Herlihy. The arrow distributed directory protocol. In *DISC*, 1998.
- 11 J. Fakcharoenphol, S. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *STOC*, 2003.
- 12 M. Ghaffari and C. Lenzen. Near-optimal distributed tree embedding. In *DISC*, 2014.
- 13 A. Ghodselahi and F. Kuhn. Dynamic Analysis of the Arrow Distributed Directory Protocol in General Networks. In *DISC*, 2017.
- 14 Abdolhamid Ghodselahi, Fabian Kuhn, and Volker Turau. Concurrent Distributed Serving with Mobile Servers. *arXiv*, 2019. [arXiv:1902.07354](https://arxiv.org/abs/1902.07354).
- 15 M. Herlihy, F. Kuhn, S. Tirthapura, and R. Wattenhofer. Dynamic analysis of the arrow distributed protocol. *Theoretical Computer Science*, 2006.
- 16 M. Herlihy, S. Tirthapura, and R. Wattenhofer. Competitive concurrent distributed queuing. In *PODC*, 2001.
- 17 P. Khanchandani and R. Wattenhofer. The Arvy Distributed Directory Protocol. In *SPAA*, 2019.
- 18 M. Manasse, L. McGeoch, and D. Sleator. Competitive algorithms for on-line problems. In *STOC*, 1988.
- 19 M. Naimi and M. Trehel. An Improvement of the $\log n$ Distributed Algorithm for Mutual Exclusion. In *ICDCS*, 1987.
- 20 D. Peleg. *Distributed computing: a locality-sensitive approach*. SIAM, 2000.
- 21 K. Raymond. A tree-based algorithm for distributed mutual exclusion. *ACM Transactions on Computer Systems*, 1989.
- 22 J. L. van de Snepscheut. Fair mutual exclusion on a graph of processes. *Distributed Computing*, 1987.
- 23 J. Welch and J. Walter. Link reversal algorithms. *Synthesis Lectures on Distributed Computing Theory*, 2011.
- 24 B. Zhang and B. Ravindran. Dynamic analysis of the relay cache-coherence protocol for distributed transactional memory. In *IPDPS*, 2010.

■ **Table 1** The essential notations used throughout the paper.

Notation	Definition	Page
n	number of pints/nodes/processors	2
k	number of servers	2
\mathcal{R}	input requests	5
$r_i = (v_i, t_i)$	request r_i that is invoked by node v_i at time t_i	5
π_{ALG}^z	z -th schedule as one of the k resulting schedules by ALG	5
$r_0^z = (v^z, 0)$	dummy request z as the tail of π_{ALG}^z	5
s^z	z -th server that serves all requests in π_{ALG}^z	5
$\mathcal{R}_{\text{ALG}}^z$	request set of π_{ALG}^z	6
$\pi_{\text{ALG}}^z(i)$	index of the request scheduled at the i -th position of π_{ALG}^z	6
$\ell_{\text{ALG}}(\mu)$	latency of message μ in an execution of ALG	6
$c_{\text{ALG}}(r_i, r_j)$	cost incurred by ALG for scheduling r_j as the successor of r_i	6
$C_{\text{ALG}}(\pi_{\text{ALG}}^z)$	total cost incurred by ALG for scheduling requests in z -th schedule	6
C_{ALG}	total cost incurred by ALG	6
$d_G(u, v)$	weight of the shortest path between u and v on the input graph G	6
H	directed version of T that is changing during a GNN execution	7
$\mu(v)$	find-predecessor message sent by v	7
B	complete graph on requests in \mathcal{R}	12
$r_{\text{src}}(\mu)$	corresponding request with message μ	12
$r_{\text{des}}(\mu)$	predecessor request of $r_{\text{src}}(\mu)$	12
$e(\mu)$	edge constructed by message μ	12
$\mu(e)$	message that constructs the edge e	12
T	input HST	12
\mathcal{F}_{ALG}	resulting forest by ALG; also, set of edges of the forest	12
F_{ALG}^z	z -th TSP path of \mathcal{F}_{ALG} ; also, set of edges of the z -th TSP path	12
$L_{\text{ALG}}(F)$	total cost of F such that $F \subseteq \mathcal{F}_{\text{ALG}}$	12
$w_G(e = (r_i, r_j))$	weight of the shortest path between v_i and v_j on the input graph G	12
$W_G(F)$	total weight of F w.r.t. measurements on the input graph G	12
\mathcal{R}_D	set of k dummy requests; $\mathcal{R}_D \subseteq \mathcal{R}$	12
$F(T')$	subgraph of F induced by the requests contained in F and T'	13
\mathcal{F}_{GRD}	locality-based forest	13
(μ, μ')	gap	13
\mathcal{F}_{mdf}	resulting forest by the transformation of \mathcal{F}_{GNN}	14
E^{old}	set of edges removed throughout the transformation of \mathcal{F}_{GNN}	14
E^{new}	set of edges added throughout the transformation of \mathcal{F}_{GNN}	14
E^{pot}	$\{e \in \mathcal{F}_{\text{GNN}} : (\mu(e), \mu(e')) \text{ is a gap for some } e' \in E^{\text{old}}\}$	14
$E^{\text{pot}}(e)$	subset of E^{pot} filtered out by $e \in E^{\text{old}}$	14
$E^{\text{old}}(e)$	subset of E^{old} filtered out by $e \in E^{\text{pot}}$	14
$\Phi(F)$	potential of F	14

Tracking Paths in Planar Graphs

David Eppstein

Department of Computer Science, University of California, Irvine, USA
eppstein@uci.edu

Michael T. Goodrich 

Department of Computer Science, University of California, Irvine, USA
goodrich@uci.edu

James A. Liu

Department of Computer Science, University of California, Irvine, USA
jamesall@uci.edu

Pedro Matias 

Department of Computer Science, University of California, Irvine, USA
pmatias@uci.edu

Abstract

We consider the NP-complete problem of tracking paths in a graph, first introduced by Banik et al. [4]. Given an undirected graph with a source s and a destination t , find the smallest subset of vertices whose intersection with any $s - t$ path results in a unique sequence. In this paper, we show that this problem remains NP-complete when the graph is planar and we give a 4-approximation algorithm in this setting. We also show, via Courcelle's theorem, that it can be solved in linear time for graphs of bounded-clique width, when its clique decomposition is given in advance.

2012 ACM Subject Classification Mathematics of computing → Graph theory; Theory of computation → Computational complexity and cryptography; Theory of computation → Design and analysis of algorithms

Keywords and phrases Approximation Algorithm, Courcelle's Theorem, Clique-Width, Planar, 3-SAT, Graph Algorithms, NP-Hardness

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.54

Acknowledgements We thank Nil Mamano for suggesting the problem of tracking paths on a graph.

1 Introduction

Motivated by applications in surveillance and monitoring, Banik et. al. [4, 3] introduced the problem of tracking paths in a graph. In essence, the goal is to uniquely determine the path traversed by a moving subject or object, based on a sequence of vertices sampled from that path. Examples of surveillance applications include the following: (i) vehicle tracking in road networks; (ii) habitat monitoring; (iii) intruder tracking and securing large infrastructures; (iv) tracing back of illicit Internet activities by tracking data packets. Another application would be to determine the nodes in a network that have been compromised by a spreading infection, given an incomplete transmission history of a pathogen. This information can be helpful in identifying and attenuating the negative impact caused by biological and non-biological infectious agents, such as:

- Highly-contagious diseases (e.g. Severe Acute Respiratory Syndrome) which could lead to epidemics [17].
- Fake news and hate speech being disseminated in social networks, as well as violations of privacy (e.g. sharing without permission highly sensitive content owned by a user, such as intimate pictures).
- Computer viruses, which spread throughout servers scattered across the Internet.



© David Eppstein, Michael T. Goodrich, James A. Liu, and Pedro Matias;
licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 54; pp. 54:1–54:17

Leibniz International Proceedings in Informatics



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

54:2 Tracking Paths in Planar Graphs

Some of these applications have been studied empirically or using heuristics in [6, 18, 25, 28] or, in the case of infections, [22, 23, 27, 1]. To the best of our knowledge, Banik et. al. were the first to approach the problem of tracking paths from a theory perspective. In this work, we extend some of their work and give new algorithms. Our main results apply to graphs that can: be embedded on the plane (of interest to surveillance in road networks and similar infrastructures) or that have bounded clique-width (mainly of theoretical interest).

Preliminaries. In the tracking paths problem, we are given an undirected graph $G = (V, E)$ with no self loops or parallel edges and a source $s \in V$ and a destination $t \in V$. The goal is to place trackers on a subset of the vertices in a way that enables us to reconstruct exactly the path traversed from s to t . Let $T \subseteq V$ be a set of vertices (where we wish to place trackers) and let \mathcal{S}_P^T be the *sequence* of vertices in T visited during the traversal of a path¹ P . Let $u - v$ denote a path from u to v . We say that T is a *tracking set* if every $s - t$ path yields a unique sequence of observed vertices in T , that is, $\mathcal{S}_{P_1}^T \neq \mathcal{S}_{P_2}^T$ for all distinct $s - t$ paths P_1 and P_2 . We consider the following problem.

PLANAR-TRACKING (G, s, t)

Input: Undirected *planar* graph $G = (V, E)$ and two vertices $s \in V$ and $t \in V$.

Question: What is the smallest tracking set for G ?

We denote by TRACKING the problem of tracking paths when the input graph is not restricted to be planar. Due to space constraints, we defer proofs of Lemmas/Theorems marked with \star to the appendix.

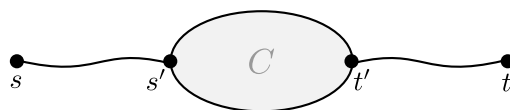
Related work. Banik et. al. first introduced TRACKING in [3], where it is shown to be NP-hard by reducing from Vertex Cover, which seems unlikely to work in the planar case. Although not immediately obvious, they also show that TRACKING is in NP, by observing that every tracking set is also a *feedback vertex set*, i.e. a set of vertices whose removal yields an acyclic graph. Finally, they present a fixed-parameter tractable (FPT) algorithm (parameterized by the solution size) for the decision version of the problem, where they obtain a kernel of size $O(k^7)$ edges.

The concept of tracking set, however, first appeared in Banik et. al. [4], where they considered a variant of TRACKING that only concerns shortest $s - t$ paths, essentially modeling the input as a directed acyclic graph (DAG). Using a similar reduction from Vertex Cover, they show that this variant cannot be approximated within a factor of 1.3606, unless $P=NP$. They also give a 2-approximation for the planar version of tracking shortest paths, but they omit any hardness results for this variant.

More recently, Bilò et. al. [7] generalized the version of the problem concerning shortest $s - t$ paths, into the case of multiple source-destination pairs, for which they claim the first $O(\sqrt{n \log n})$ -approximation algorithm for general graphs. They also study a version of this multiple source-destination pairs problem in which the set of trackers itself (excluding the order in which they are visited) is enough to distinguish between $s - t$ shortest paths². In this setting, they claim a $O(\sqrt{n})$ -approximation algorithm and they show that it is NP-hard even for cubic planar graphs. The hardness construction intrinsically relies on the multiplicity of source-destination pairs and, therefore, cannot be adapted to the problem studied in this

¹ Some authors use the terms “path” and “walk” interchangeably, where vertices may be repeated, but in this paper, paths are required to have distinct vertices.

² Notice that when tracking shortest paths only and using a single source-destination pair, these two versions of the problem are the same.



■ **Figure 1** Entry-exit pair illustration, with entry vertex s' and exit vertex t' .

paper. They also give an FPT algorithm (w.r.t to the maximum number of vertices at the same distance from the source) for the problem concerning a single source-destination pair that was introduced in [4].

In [2], Banik and Choudhary generalize TRACKING into a problem on set systems³, which are characterized by a universe (e.g. vertex set) and a family of subsets of the universe (e.g. $s - t$ paths). They show that this generalized version of the problem is fixed-parameter tractable, by establishing a correspondence with the well known Test Cover problem.

Our results. In this paper, we give a 4-approximation for PLANAR-TRACKING (Section 3) and prove that it is NP-complete (Section 4). In addition, we show that TRACKING can be solved in cubic time for graphs of bounded clique-width and linear time if the clique decomposition of bounded width is given in advance (Section 5).

2 Definitions

► **Definition 1** (Entry-exit pair). *Let (G, s, t) be an instance of TRACKING. An entry-exit pair is, with respect to some simple cycle C in $G = (V, E)$, an ordered pair (s', t') of vertices in C that satisfy the following conditions:*

1. *There exists a path $s - s'$ from s to the entry vertex s'*
2. *There exists a path $t' - t$ from the exit vertex t' to t*
3. *Paths $s - s'$ and $t' - t$ are vertex-disjoint*
4. *Path $s - s'$ (resp. $t' - t$) and C share exactly one vertex: s' (resp. t').*

Essentially, an entry-exit pair (s', t') with respect to a cycle C (see Figure 1) represents two alternative $s - t$ paths and, thus, requires tracking at least one of them. We say that (s', t') is *tracked with respect to C* if and only if $C \setminus \{s', t'\}$ contains a tracker. In addition, C is *tracked* if and only if there is no entry-exit pair with respect to C that is untracked. If a cycle contains either (i) 3 trackers or (ii) s or t and 1 tracker in a non-entry/non-exit vertex, then it must be tracked. We say that these cycles are *trivially tracked*.

An alternative characterization of a tracking set, first given by Banik et al [3, Lemma 2], is the following.

► **Lemma 2** ([3]). *For a graph $G = (V, E)$, a subset $T \subseteq V$ is a tracking set if and only if every simple cycle C in G is tracked with respect to T .*

3 Approximation algorithm

► **Theorem 3.** *There exists a 4-approximation algorithm for PLANAR-TRACKING.*

The overall idea for the approximation algorithm builds on the following two insights:

1. The cardinality of the optimal solution cannot be much smaller than the number of faces in the graph.
2. The average number of trackers per face does not need to be very large.

³ Also called hypergraphs.

The first idea gives us a lower bound on OPT , the cardinality of an optimal solution. The second gives us an upper bound on ALG , the cardinality of our approximation algorithm.

3.1 Lower bound on OPT

We consider the following reduction, which takes care of disconnected components, or components that are “attached” to the graph by a cut vertex that is not in an $s - t$ path. We say that a reduction is *safe*, if it does not eliminate any untracked cycles.

Reduction 1. While there exists an edge or vertex that does not participate in any $s - t$ path, remove it from the graph.

► **Lemma 4** ([3]). ★ *Reduction 1 is safe and can be done in polynomial time.*

► **Lemma 5.** *After Reduction 1, every simple cycle in the graph contains at least one entry-exit pair. This holds for non-planar graphs as well.*

Proof. Let C be some simple cycle in the graph. After Reduction 1, there must exist an $s - t$ path that shares an edge with C . The first and last vertices on this path that belong to C correspond to an entry-exit pair. ◀

► **Lemma 6.** *In an embedded undirected planar graph G that results from Reduction 1, $OPT \geq (|F| - 1)/2$, where F is the set of faces of G .*

On a high level, the proof of Lemma 6 is done by keeping a set of “active” trackers while reconstructing a planar embedding \mathcal{E} of G : we start, as a base case, with any simple $s - t$ path in \mathcal{E} and iteratively add faces to it until it matches \mathcal{E} . Given a fixed, optimal tracking set T^* , the addition of each face requires either (i) adding a new tracker from T^* to the active set, or (ii) deactivating an active tracker, rendering it useless for distinguishing paths on future faces. As a consequence, each tracker charges at most two faces: the one adding the tracker and the one deactivating it. This demonstrates that $|T^*| \geq (|F| - 1)/2$.

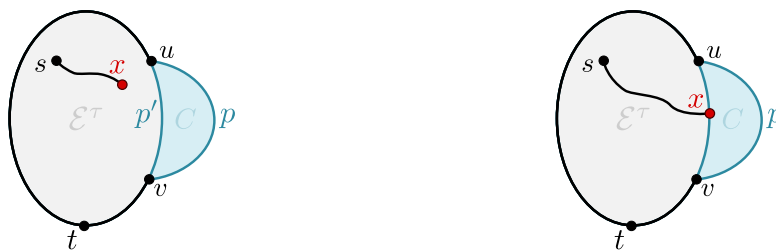
Let $\text{OUTER}(\mathcal{E}^\tau)$ be the set of outer-edges of our planar reconstructing embedding \mathcal{E}^τ at time τ . At time $\tau = 0$, our embedding corresponds to an $s - t$ path and, for all $0 \leq \tau \leq |F| - 1$, we add exactly one face C from \mathcal{E} to \mathcal{E}^τ , by connecting two vertices u and v in $\text{OUTER}(\mathcal{E}^\tau)$ with a simple path p (see Figure 8 in the appendix). In doing so, we erase an $u - v$ path p' in $\text{OUTER}(\mathcal{E}^\tau)$, so we have that $\text{OUTER}(\mathcal{E}^{\tau+1}) = \text{OUTER}(\mathcal{E}^\tau) \setminus p' \cup p$. In the end, $\mathcal{E}^{|F|} = \mathcal{E}$.

By Lemma 5, there is at least one entry-exit pair in \mathcal{E} with respect to face C , so any tracking set must contain a tracker on some vertex of C . During the reconstruction process, we maintain a list of trackers in sets A and A' , such that $(A \cup A') \subseteq T^*$, where A contains active trackers and A' contains inactive ones. A tracker in vertex v is *active* at time τ if and only if it meets both of the following conditions:

Condition (i) $v \in \text{OUTER}(\mathcal{E}^\tau)$

Condition (ii) There is no $s - v$ path in \mathcal{E}^τ that traverses vertices in $\text{OUTER}(\mathcal{E}^\tau)$

Intuitively, an active tracker can be used to track future faces, although no more than one (see below). An inactive tracker, on the other hand, either cannot be used to track future faces (Condition (i)), or its corresponding vertex is entry/exit for some future face (Condition (ii)), in which case we require yet another tracker on that face (see Figure 2). Condition (ii) is necessary for dealing with embeddings of G where at least one of s and t is not in the outer face.



(a) Tracker on x (red) is inactive due to the violation of Condition (i).

(b) Tracker on x (red) is inactive due to the violation of Condition (ii). Notice that x is entry for exit vertices u and v (with respect to C), therefore C needs another tracker.

■ **Figure 2** Examples of inactive trackers used in the proof of Lemma 6.

Proof of Lemma 6.

First, we argue that each time we add a face during the reconstruction process described above, we either (i) need to increase the number of active trackers (by adding it to either A or A'), or (ii) we can get away by re-using and, therefore, deactivating an active tracker.

We assume for the rest of the argument that t is on the outer face, because such a planar embedding is always possible to construct.

Let C be the face added a time τ by connecting vertices u and v in $\text{OUTER}(\mathcal{E}^\tau)$, as specified above. In addition, let T^* be any optimal solution (i.e. $|T^*| = \text{OPT}$). We consider two cases, depending on the existence of a tracker in C at time τ :

Case 1: $C \cap A = \emptyset$ at time τ .

By Lemma 5, there exists a vertex $x \in C$ such that $x \in T^*$. We place a tracker on x . If $x \in \text{OUTER}(\mathcal{E}^{\tau+1})$ we add x to A , otherwise we add it to A' .

Case 1: $C \cap A \neq \emptyset$ at time τ .

Let $y \in C \cap A$ be a vertex of C with a tracker. We again consider two cases:

- (i) $y \notin \{u, v\}$. Then, $y \in \text{OUTER}(\mathcal{E}^\tau)$ but $y \notin \text{OUTER}(\mathcal{E}^{\tau+1})$, which amounts to moving y from A to A' .
- (ii) $y \in \{u, v\}$. If (u, v) is an entry-exit pair with respect to C , or if the tracker in y is not active, then there exists $x' \in C \setminus \{u, v\}$ such that $x' \in T^*$. Similarly to Case 1, we place a tracker on x' , which corresponds to adding x' either to A or A' .

Otherwise, the tracker in y is active and (u, v) is not an entry-exit pair with respect to C . Let us assume without loss of generality that $y = u$. Then, the addition of C deactivates the tracker in u by definition of active tracker (Condition (ii) is now violated), so we move u from A to A' .

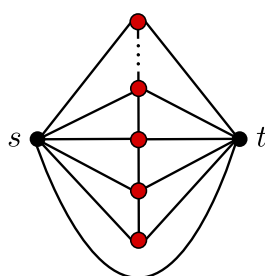
Every tracker in A' is charged by at most two faces: one for adding an active tracker to A and another for deactivating it and moving it to A' . Therefore, $|F| - 1 \leq |A| + 2|A'|$. Since $|A| + |A'| \leq |T^*|$, it follows that $|F| - 1 \leq 2\text{OPT}$. ◀

A tight example for the lower bound on OPT is illustrated in Figure 3a.

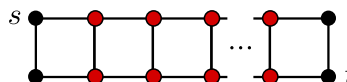
3.2 Upper bound on ALG

We say that an undirected planar graph is *reduced* if it cannot be further reduced by Reduction 1 or any of the following reductions.

Reduction 2. While there exist two adjacent vertices of degree 2, remove one of them (and its edges) and add an edge connecting its neighbors.



(a) Example of a planar graph where $OPT = |F|/2$ (in red).



(b) Example of a planar graph where $ALG = 2(|F| - 2)$ (in red).

■ **Figure 3** Tight examples for the lower bound on OPT (left) and the upper bound on ALG (right) in planar graphs.

Reduction 3. While there exists vertex $v \notin \{s, t\}$ of degree 2 in a 3-cycle, place a tracker on v and remove it and its edges from the graph.

Reduction 4. While there exist non-adjacent vertices $u, v \notin \{s, t\}$ of degree 2 in a 4-cycle, place a tracker on either u or v and remove it and its edges from the graph.

Notice that all reduction rules are valid for general graphs, not only planar ones. In addition, Reductions 2, 3 and 4 can be applied interchangeably and in any order until none of them is applicable, but we will see that they need to be carried out after Reduction 1. Fortunately, we will not be required to re-apply Reduction 1 after performing the remaining reductions.

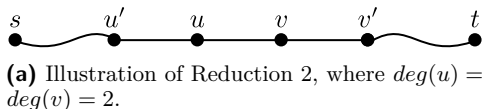
We denote the degree of a vertex v by $deg(v)$, where the underlying graph can be determined from its context.

▷ **Claim 7.** If vertex v is on an entry-exit pair, then $deg(v) > 2$.

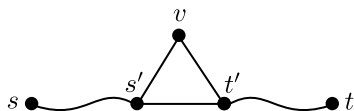
Proof. Trivial by Definition 1. ◁

▷ **Claim 8.** Reductions 2, 3 and 4 maintain the property that every cycle in G contains at least one entry-exit pair (see Lemma 5).

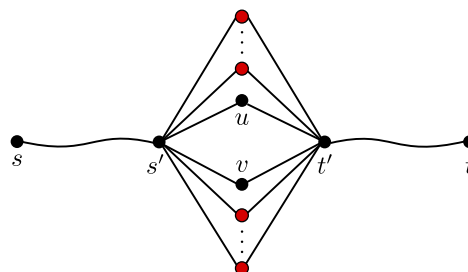
Proof. Reductions 2, 3 and 4 only erase faces and vertices of degree 2, which cannot be in entry-exit pairs (by Claim 7), so every simple cycle of the graph still contains an entry-exit pair. ◁



(a) Illustration of Reduction 2, where $deg(u) = deg(v) = 2$.



(b) Illustration of Reduction 3, where $deg(s') \geq 3$, $deg(t') \geq 3$ and $deg(v) = 2$.



(c) Illustration of Reduction 4, where $deg(s') \geq 3$, $deg(t') \geq 3$ and $deg(u) = deg(v) = 2$.

■ **Figure 4** Illustration of Reductions 2, 3 and 4.

► **Lemma 9.** *★ Reduction 2 is safe and can be done in polynomial time, if done after Reduction 1.*

► **Lemma 10.** *★ Reduction 3 is safe and can be done in polynomial time, if done after Reduction 1.*

► **Lemma 11.** *★ Reduction 4 is safe and can be done in polynomial time, if done after Reduction 1.*

► **Remark 12.** None of Reductions 2, 3 and 4 compromise planarity.

■ **Algorithm 1** \mathcal{A} .

Input : Undirected planar graph $G = (V, E)$ and vertices $s \in V$ and $t \in V$

Output : Tracking set

- 1 Perform Reduction 1 in G
 - 2 Perform Reductions 2, 3 and 4 repeatedly until G is reduced.
 - 3 Output remaining vertices of degree at least 3 (except s or t)
-

► **Lemma 13.** *Algorithm \mathcal{A} outputs a tracking set for the input graph G .*

Proof. By Lemmas 4, 9, 10 and 11, Reductions 1-4 are safe, so let us assume without loss of generality that G is reduced. Then, every cycle of G of 5 or more vertices is trivially tracked, because it must contain at least 3 vertices of degree at least 3 (by Reduction 2). Similarly, every 3- or 4-cycle must contain at least 3 vertices of degree at least 3 by Reduction 2 and Reductions 3 and 4 (respectively). ◀

► **Lemma 14.** *Algorithm \mathcal{A} outputs a tracking set of size at most $2(|F| - 2)$, where F is the set of faces of the input graph G .*

Proof. Notice that each tracker added during Reductions 3 and 4 is associated with the removal of one face from G . Therefore, it is enough to show that the lemma holds with respect to a reduced graph G . Let us partition V into $V = (V_2 \cup V_{\geq 3})$, where V_2 and $V_{\geq 3}$ consist of the vertices of degree 2 and degree at least 3, respectively (notice that there cannot be vertices of degree 1). The lemma statement follows from Lemma 13 and the fact that $|F| \geq \frac{|V_{\geq 3}|}{2} + 2$. This inequality can be derived by plugging in the inequality $2|E| \geq 3|V_{\geq 3}| + 2|V_2|$ in Euler's formula for planar graphs: $|V| - |E| + |F| = 2$, where E is the set of edges of G . ◀

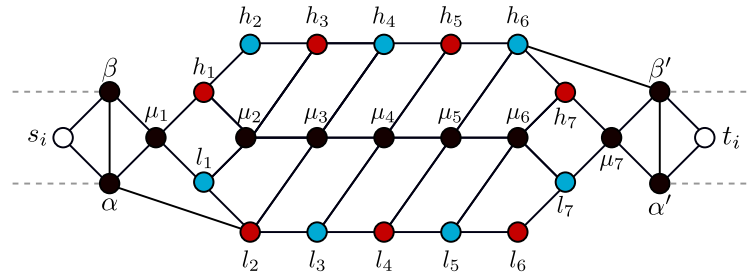
A tight example is illustrated in Figure 3b.

Proof of Theorem 3. By Lemmas 6 and 14, Algorithm \mathcal{A} is a 4-approximation to PLANAR-TRACKING. ◀

4 Hardness of tracking paths

We show that PLANAR-TRACKING is NP-hard, by reducing from PLANAR-3-SAT, a special version of the satisfiability problem, shown to be NP-complete by Lichtenstein [21].

In 3-SAT, we are given a set $\mathcal{X} = \{x_1, \dots, x_p\}$ of variables and a 3-CNF formula ϕ , where each clause in ϕ is a disjunction of exactly three distinct literals with respect to \mathcal{X} . The goal is to find a boolean assignment to all variables in \mathcal{X} that satisfies ϕ . Consider the bipartite



■ **Figure 5** Illustration of x_i 's gadget, containing m_i vertices for x_i (in blue) and m_i vertices for \bar{x}_i (in red). Vertices colored black require trackers in any minimum tracking set. The dashed edges are added to force trackers in s_i and t_i .

graph with a vertex for each clause C in ϕ and each variable $x_i \in \mathcal{X}$, and edges (x_i, C) if and only if C contains x_i or its negation \bar{x}_i . Lichtenstein [21] showed that PLANAR-3-SAT, the subset of instances of 3-SAT whose underlying bipartite graph is planar, remains NP-complete. In particular, the definition of PLANAR-3-SAT requires that a cycle can be drawn connecting all of the variables while maintaining planarity. Later, Knuth and Raghunatan [20] exploited this condition to show that we can always draw the underlying bipartite graph of a PLANAR-3-SAT instance in a *rectilinear* fashion without crossings (example in Figure 10a): variables are arranged in a horizontal line and clauses are horizontal line segments with vertical legs to represent the literals present in the clause. Vertical legs attach to the appropriate variables and are labeled *red* for negated literals and *blue*, otherwise. In particular, a given clause is drawn completely above or below the line of variables.

We convert a planar rectilinear drawing \mathcal{D} of an instance of PLANAR-3-SAT, with formula ϕ and a set \mathcal{X} of variables, into a planar drawing \mathcal{G} corresponding to the instance of PLANAR-TRACKING. The reduction is straightforward:

- (i) transform each variable x_i in \mathcal{D} into a gadget containing m_i copies of literal vertices x_i and \bar{x}_i ;
- (ii) transform each 3-legged clause into a face containing corresponding literals vertices and an entry-exit pair;
- (iii) choose the boolean assignment according to the placement of trackers, such that a clause is satisfied if and only if its corresponding face is tracked.

The union of all the variable and clause gadgets constitutes \mathcal{G} (see example in Figure 10 in the appendix). Details of each gadget are given below. For simplicity, we avoid introducing too many subscripts and we rely on pictures to describe the gadgets.

Variable gadget. Each variable gadget converts a variable x_i in \mathcal{D} into a connected subgraph corresponding to Figure 5, with length parameterized by m_i . We refer to the set $\{h_k, \mu_k, l_k\}$ as column k and we refer to the vertices $\{h_1, \dots, h_{m_i}\} \cup \{l_1, \dots, l_{m_i}\}$ as *literal vertices*.

Each variable gadget is linked with the next one by setting $t_i = s_{i+1}$, to form a horizontal chain of gadgets, where $s = s_1$ and $t = t_p$. For convenience, we force trackers in all the s_i (except s), by drawing edges between the α' (β') of a variable gadget and the α (β) of the next variable gadget in the chain. We refer to the resulting drawing as the *spine*.

There are exactly two minimum tracking sets associated with the variable gadget with source s_i and destination t_i . One of them corresponds to a true assignment of x_i and the other one to a false assignment. Both of them require tracking the vertices in $R =$

$\{\alpha, \alpha', \beta, \beta', \mu_1, \mu_{m_i}\}$, as well as the remaining μ_k . In addition, the true assignment tracks the even-indexed h_k and odd-indexed l_k , while the false assignment tracks the odd-indexed h_k and even-indexed l_k . This requires $2m_i + 4$ trackers in total.

► **Lemma 15.** *★ The true and false assignments are the only minimum tracking sets.*

Clause gadget. Let $C = (\ell_a \vee \ell_b \vee \ell_c)$ be a clause in ϕ with literals corresponding to variables $x_a, x_b, x_c \in \mathcal{X}$. Its gadget, depicted in Figure 6, is a face F_C consisting of:

- literal vertex α from x_a 's gadget, corresponding to literal ℓ_a ;
- adjacent literal vertices $\beta_1, \overline{\beta_1}, \beta_2, \overline{\beta_2}, \beta_3$ from x_b 's gadget, corresponding to literals alternating between ℓ_b and $\overline{\ell_b}$;
- literal vertex γ from x_c 's gadget, corresponding to literal ℓ_c ;
- edges $(\alpha, \gamma), (\alpha, \beta_1), (\beta_3, \gamma)$ and the edges from x_b 's gadget connecting all of the β_k and $\overline{\beta_k}$.

We can increase the lengths of the variable gadgets to any polynomial that provides enough literal vertices for all clauses. Since \mathcal{D} is planar, there are no crossings between clauses. We also impose the following restrictions:

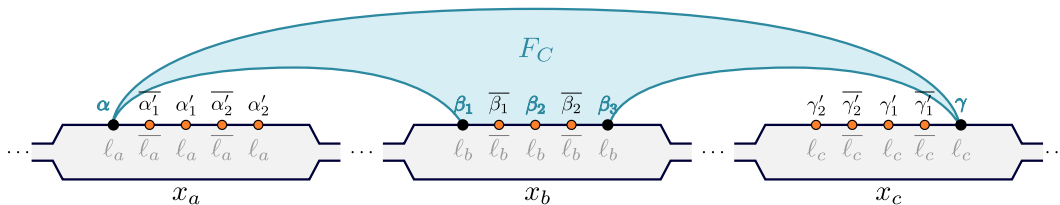
1. α cannot be one of $\{h_1, l_{m_a}\}$; this ensures that the only faces in x_a 's gadget that do not require 3 trackers do not become untracked. We apply the equivalent restriction to β_1, β_3 and γ .
2. The $\alpha'_k/\overline{\alpha'_k}$, (see Figure 6) cannot belong to any other clause gadget; these correspond to the 4 literal vertices following α in x_a 's gadget and reserving them ensures that non-clause faces, between nested clauses, are tracked. We apply the equivalent restriction to the $\gamma'_k/\overline{\gamma'_k}$.
3. All literal vertices in a clause need to be on the same side of the spine; this restriction is trivial because \mathcal{D} is rectilinear, but it simplifies the analysis.

► **Lemma 16.** *★ Clause C is satisfied if and only if its corresponding gadget face F_C is tracked.*

► **Theorem 17.** *★ There exists a polynomial time reduction from PLANAR-3-SAT to PLANAR-TRACKING.*

► **Corollary 18.** PLANAR-TRACKING is NP-hard.

It remains to show that PLANAR-TRACKING is in NP; Banik et. al. [3] prove this in the more general case of TRACKING.



■ **Figure 6** Illustration of the gadget for clause $C = (\ell_a \vee \ell_b \vee \ell_c)$, where the vertices in each variable gadget are all adjacent. The entry-exit (β_1, β_2) are responsible for satisfying C .

5 Bounded clique-width graphs

We show that TRACKING can be solved in linear time when the input graph has bounded clique-width, by applying Courcelle’s theorem [9, 12, 14], a powerful meta-theorem that establishes fixed-parameter tractability of *any* graph property that is expressible in monadic second order logic.

Clique-width, first introduced by Courcelle et. al. [13] and revisited by Courcelle and Olariu [16], is an important graph parameter that, intuitively, measures the closeness of a graph to a cograph – a graph with no induced 4-vertex paths. It is closely related to *tree-width*, another influential graph parameter that measures closeness of a graph to a tree and that was first introduced by Bertelé and Brioschi [5] and later rediscovered by Halin [19] and Robertson and Seymour [26]. While both parameters are determined based on specific hierarchical decompositions of a graph, the clique-width is strictly more powerful in the sense that the class of graphs of bounded clique-width includes all graphs of bounded tree-width, but not vice-versa. Details on the relationship between these parameters can be found in Courcelle and Engelfriet [12]. Graphs of bounded clique-width include series-parallel graphs, outerplanar graphs, pseudoforests, cographs, distance-hereditary graphs, etc.

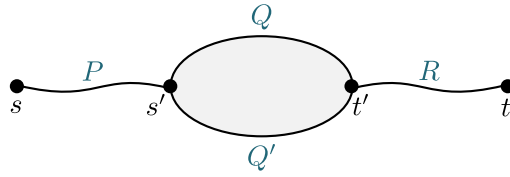
MSO₁ vs MSO₂. Second order logic extends first order logic, by allowing quantification over relations (of any fixed arity) on the elements of the domain of discourse. Monadic second order logic itself only allows quantification over unary relations (subsets of the domain of discourse) and, in the logic of graphs, it comes in two flavors: MSO₁ and MSO₂. The only distinction between these is that the latter allows edges to be elements of the domain of discourse (and thus be quantified over), while the former does not. Besides the quantifiers (\forall and \exists) and the standard logic operations $\neg, \wedge, \vee, \rightarrow$, both logics include predicates for equality ($=$) and relation membership (\in). In addition, MSO₁ includes a predicate (\sim) that determines vertex adjacency and MSO₂ includes a predicate for vertex-edge incidence. MSO₂ is more expressive, for example: Hamiltonicity can be expressed using MSO₂, but not using MSO₁. Details on the distinction between the two logics can be found in [12].

Courcelle’s theorem. Courcelle et. al. [9, 14] showed that any graph property expressed in MSO₁ and MSO₂ is FPT under clique-width and tree-width (respectively). More specifically, they showed that any MSO₁-, MSO₂-expressible property can be tested in $f(k, l) \cdot n$ and $g(k', l') \cdot (n + m)$ time (respectively) for graphs of clique-width k and tree-width k' , where: f and g are computable functions, l and l' are the lengths of the logic formulas, n is the number of vertices and m is the number of edges. The result for MSO₂ is valid in optimization problems with linear evaluation functions [15]. Later, Courcelle, Makowsky and Rotics [14] extended these results for MSO₁. Examples of constructing FPT graph algorithms parameterized by clique-width or tree-width, which are based on automata, are given in [10, 11]. While it is possible to construct tree decompositions of width k' in linear time [8], there is no FPT algorithm for finding clique decompositions of clique-width $k > 3$. Fortunately, it is possible to construct a clique decomposition of width exponential in k in cubic time [24]. In this section, we will take advantage of the latter.

We give an alternative definition for tracking set that is easier to express using the logic of graphs.

► **Lemma 19 (Tracking set).** *For an undirected graph $G = (V, E)$, a subset $T \subseteq V$ is a tracking set if and only if there is no $s - t$ path $P_{st} = P_{ss'} \cup P_{s't'} \cup P_{t't}$ for $s', t' \in V$ and corresponding $s - s'$, $s' - t'$ and $t' - t$ paths, such that:*

1. *There exists an alternative $s' - t'$ path $P'_{s't'} \neq P_{s't'}$ and*
2. *$T \cap (P_{s't'} \cup P'_{s't'}) \subseteq \{s', t'\}$.*



■ **Figure 7** Illustration of variable sets P , Q , Q' and R as well as vertex variables s , s' , t' and t used in expressing ISTRACKINGSET using MSO_1 .

Proof. This follows directly from Lemma 2. ◀

In the logic formulas presented below, we use lowercase letters to quantify over vertices and uppercase letters to quantify over sets of vertices. We use s and t as free variables and, for convenience, we also use set intersection (\cap), union (\cup) and containment (\subseteq), without explicitly expressing these operations using MSO_1 .

$$\begin{aligned} & \text{ISTRACKINGSET}(T, s, t) \\ \iff & \exists P, Q, R [\exists s', t' [\text{HASPATH}(P, s, s') \wedge \text{HASPATH}(Q, s', t') \wedge \text{HASPATH}(R, t', t) \\ & \wedge P \cap Q = \{s'\} \wedge Q \cap R = \{t'\} \wedge P \cap R = \emptyset \\ & \wedge \exists Q' \neq Q [\text{HASPATH}(Q', s', t') \wedge T \cap (Q \cup Q') \subseteq \{s', t'\}]]] \end{aligned}$$

The first two lines of the above equivalence establish that P , Q and R form an $s - t$ path. The last line restricts s' and t' to be an entry-exit pair with respect to the cycle $Q \cup Q'$ (see Figure 7) and, in addition, establishes that the cycle $Q \cup Q'$ is not tracked.

The primitive $\text{HASPATH}(X, a, b)$, whose input consists of a set $X \subseteq V$ and vertices $a, b \in V$, verifies the existence of a simple path between a and b that only uses vertices in X . We define it as follows:

$$\begin{aligned} & \text{HASPATH}(X, a, b) \\ \iff & \exists X_1, X_2 \subseteq X [X_1 \cup X_2 = X \wedge a \in X_1 \wedge b \in X_2 \wedge \neg(\exists u \in X_1 \wedge \exists v \in X_2 [u \sim v])] \end{aligned}$$

► **Remark 20.** $\text{HASPATH}(X, a, b)$ is correctly expressed under MSO_1 and it correctly verifies that there exists an $a - b$ path using only vertices in X .

► **Remark 21.** ISTRACKINGSET is correctly expressed under MSO_1 and it correctly verifies that the given subset of vertices is a tracking set.

► **Theorem 22.** $\text{TRACKING}(G, s, t)$ can be solved in polynomial time if G has bounded clique-width. Moreover, if a clique decomposition of bounded width is given, it can be solved in linear time.

Proof. This follows directly from Lemma 19, Remarks 20 and 21, and the linear time algorithm given by Courcelle [14] for any optimization problem on graphs of bounded clique-width, whose decomposition is given in advance. ◀

6 Conclusion and open questions

We showed that PLANAR-TRACKING is NP-complete and we give a 4-approximation algorithm. We also show that, for graphs of bounded-clique width, TRACKING can be solved in linear time by applying Courcelle's theorem, as long as its clique decomposition is given in advance. A natural direction of future study would be to improve the approximation ratio of PLANAR-TRACKING or establish constant approximation factors for graphs of larger genus or, more

generally, for TRACKING. Another open question is to establish the difficulty of TRACKING on directed graphs: on one side planar directed acyclic graphs are not known to be NP-hard and, on the other side, it is not known whether its general or planar versions are in NP. Finally, it would be interesting to find efficient algorithms for graphs of bounded tree-width or clique-width without resorting to the finite automaton approach used in Courcelle's theorem.

References

- 1 Norman TJ Bailey. *The Mathematical Theory of Infectious Diseases and its Applications*. Charles Griffin & Company Ltd, High Wycombe, United Kingdom, 2nd edition, 1975.
- 2 Aritra Banik and Pratibha Choudhary. Fixed-Parameter Tractable Algorithms for Tracking Set Problems. In B. S. Panda and Partha P. Goswami, editors, *Algorithms and Discrete Applied Mathematics - 4th International Conference, CALDAM 2018, Guwahati, India, February 15-17, 2018, Proceedings*, volume 10743 of *Lecture Notes in Computer Science*, pages 93–104. Springer, 2018. doi:10.1007/978-3-319-74180-2_8.
- 3 Aritra Banik, Pratibha Choudhary, Daniel Lokshtanov, Venkatesh Raman, and Saket Saurabh. A polynomial sized kernel for tracking paths problem. In *Latin American Symposium on Theoretical Informatics*, volume 10807 of *Lecture Notes in Computer Science*, pages 94–107. Springer, 2018. doi:10.1007/978-3-319-77404-6_8.
- 4 Aritra Banik, Matthew J Katz, Eli Packer, and Marina Simakov. Tracking paths. In *International Conference on Algorithms and Complexity*, volume 10236 of *Lecture Notes in Computer Science*, pages 67–79. Springer, 2017. doi:10.1007/978-3-319-57586-5_7.
- 5 Umberto Bertele and Francesco Brioschi. *Nonserial Dynamic Programming*. Academic Press, 1972. URL: <https://www.elsevier.com/books/nonserial-dynamic-programming/bertele/978-0-12-093450-8>.
- 6 Sania Bhatti and Jie Xu. Survey of target tracking protocols using wireless sensor network. In *2009 Fifth International Conference on Wireless and Mobile Communications*, pages 110–115. IEEE, 2009. doi:10.1109/ICWMC.2009.25.
- 7 Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Tracking Routes in Communication Networks. In Keren Censor-Hillel and Michele Flammini, editors, *Structural Information and Communication Complexity - 26th International Colloquium, SIROCCO 2019, L'Aquila, Italy, July 1-4, 2019, Proceedings*, volume 11639 of *Lecture Notes in Computer Science*, pages 81–93. Springer, 2019. doi:10.1007/978-3-030-24922-9_6.
- 8 Hans L Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.
- 9 Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.
- 10 Bruno Courcelle and Irène Durand. Automata for the verification of monadic second-order graph properties. *J. Applied Logic*, 10(4):368–409, 2012. doi:10.1016/j.jal.2011.07.001.
- 11 Bruno Courcelle and Irène Durand. Computations by fly-automata beyond monadic second-order logic. *Theor. Comput. Sci.*, 619:32–67, 2016. doi:10.1016/j.tcs.2015.12.026.
- 12 Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*, volume 138 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2012. URL: http://www.cambridge.org/fr/knowledge/isbn/item5758776/?site_locale=fr_FR.
- 13 Bruno Courcelle, Joost Engelfriet, and Grzegorz Rozenberg. Handle-rewriting hypergraph grammars. *Journal of Computer and System Sciences*, 46(2):218–270, 1993. doi:10.1016/0022-0000(93)90004-G.
- 14 Bruno Courcelle, Johann A Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems*, 33(2):125–150, 2000. doi:10.1007/s002249910009.

- 15 Bruno Courcelle and Mohamed Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theoretical Computer Science*, 109(1-2):49–82, 1993. doi:10.1016/0304-3975(93)90064-Z.
- 16 Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000. doi:10.1016/S0166-218X(99)00184-5.
- 17 Kee-Tai Goh, Jeffery Cutter, Bee-Hoon Heng, Stefan Ma, Benjamin KW Koh, Cynthia Kwok, Cheong-Mui Toh, and Suok-Kai Chew. Epidemiology and control of SARS in Singapore. *Annals of the Academy of Medicine, Singapore*, 35(5):301, 2006. PMID:16829997.
- 18 Rahul Gupta and Samir R Das. Tracking moving targets in a smart sensor network. In *2003 IEEE 58th Vehicular Technology Conference. VTC 2003-Fall*, volume 5, pages 3035–3039. IEEE, 2003. doi:10.1109/VETEFC.2003.1286181.
- 19 Rudolf Halin. S-functions for graphs. *Journal of Geometry*, 8(1-2):171–186, 1976. doi:10.1007/BF01917434.
- 20 Donald E Knuth and Arvind Raghunathan. The problem of compatible representatives. *SIAM Journal on Discrete Mathematics*, 5(3):422–427, 1992. doi:10.1137/0405033.
- 21 David Lichtenstein. Planar formulae and their uses. *SIAM Journal on Computing*, 11(2):329–343, 1982. doi:10.1137/0211025.
- 22 Cristopher Moore and Mark EJ Newman. Epidemics and percolation in small-world networks. *Physical Review E*, 61(5):5678, 2000. doi:10.1103/PhysRevE.61.5678.
- 23 Mark EJ Newman. Spread of epidemic disease on networks. *Physical Review E*, 66(1):016128, 2002. doi:10.1103/PhysRevE.66.016128.
- 24 Sang-Il Oum. Approximating rank-width and clique-width quickly. *ACM Transactions on Algorithms*, 5(1):10, 2008. doi:10.1145/1435375.1435385.
- 25 Tao Peng, Christopher Leckie, and Kotagiri Ramamohanarao. Survey of network-based defense mechanisms countering the DoS and DDoS problems. *ACM Computing Surveys*, 39(1):3, 2007. doi:10.1145/1216370.1216373.
- 26 Neil Robertson and Paul D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, 1986. doi:10.1016/0196-6774(86)90023-4.
- 27 Devavrat Shah and Tauhid Zaman. Rumors in a network: Who’s the culprit? *IEEE Transactions on Information Theory*, 57(8):5163–5181, 2011. doi:10.1109/TIT.2011.2158885.
- 28 Alex C Snoeren, Craig Partridge, Luis A Sanchez, Christine E Jones, Fabrice Tchakountio, Stephen T Kent, and W Timothy Strayer. Hash-based IP traceback. *ACM SIGCOMM Computer Communication Review*, 31(4):3–14, 2001. doi:10.1145/383059.383060.

A Deferred proofs on the approximation algorithm

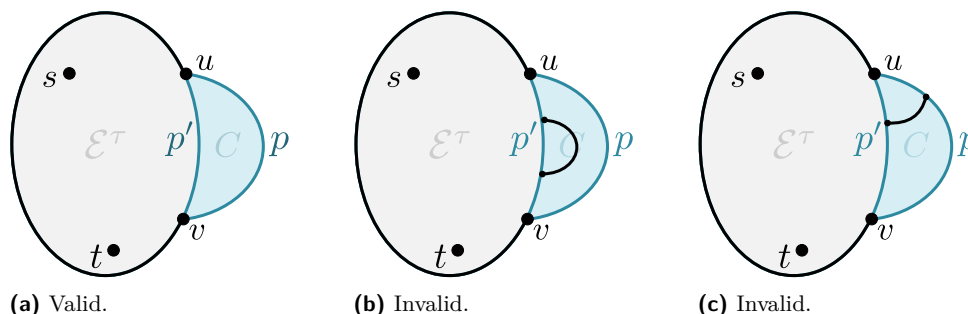
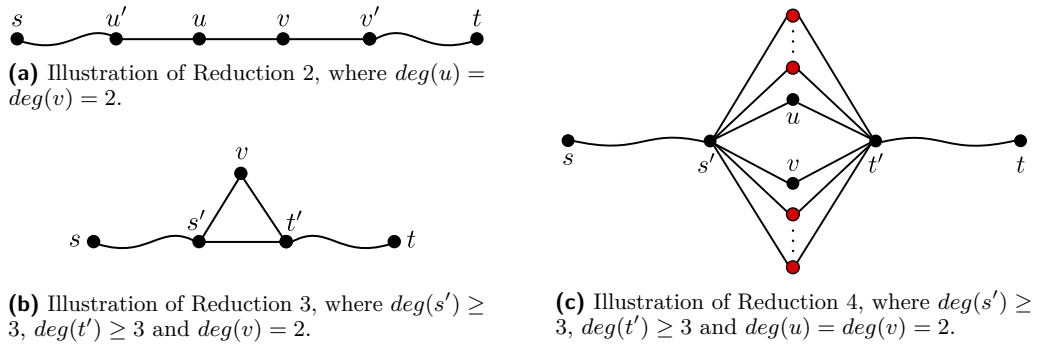


Figure 8 Illustration of valid and invalid choices of vertices u and v in the proof of Lemma 6. Cycle C must correspond to a face in the fixed embedding \mathcal{E} .

54:14 Tracking Paths in Planar Graphs



■ **Figure 9** Illustration of Reductions 2, 3 and 4.

► **Lemma 4** ([3]). *★ Reduction 1 is safe and can be done in polynomial time.*

Proof. See proof in [3]. ◀

► **Lemma 9.** *★ Reduction 2 is safe and can be done in polynomial time, if done after Reduction 1.*

Proof. Banik et. al. [3, Lemma 16] showed that having three adjacent vertices of degree 2 is redundant. Here, we extend their proof and show that we can also get rid of the second adjacent vertex of degree 2. Let u and v be the two adjacent vertices of degree 2, and let u' be the other neighbor of u . Notice that v, u' are not adjacent, otherwise the u, v edge would then not belong to an $s - t$ path and thus be removed by Reduction 1. Hence, no parallel edges are added to the graph after this reduction. We argue that this reduction is safe by showing that there exists a minimum tracking set that does not include u and v simultaneously. Take any minimum tracking set T that includes u and v . We can always move the tracker from u to u' ; this remains a tracking set, because u immediately follows or precedes u' on any $s - t$ path. This can only decrease T 's cardinality. This reduction can be done in $O(\text{poly}(n))$ time, by repeatedly checking for the existence of adjacent vertices of degree 2. ◀

► **Lemma 10.** *★ Reduction 3 is safe and can be done in polynomial time, if done after Reduction 1.*

Proof. Let $v \notin \{s, t\}$ be the vertex of degree 2 in the triangle $\Delta vs't'$ (see Figure 9b). Then, it must be the case that s' and t' form an entry-exit pair. This follows from (i) the fact that there exists an entry-exit pair in $\Delta vs't'$ (by Lemma 5) and (ii) the fact that v cannot be in an entry-exit pair (by Claim 7). Then, by Lemma 2, any feasible solution must place a tracker on v . Therefore, v and its edges can be removed, since v is neither a cut-vertex, nor in any entry-exit pair, so that its removal does not eliminate an untracked cycle. Clearly, this reduction can be done in $O(\text{poly}(n))$ time. ◀

► **Lemma 11.** *★ Reduction 4 is safe and can be done in polynomial time, if done after Reduction 1.*

Proof. Let $u, v \notin \{s, t\}$ be the two vertices of degree 2 that connect the same pair of vertices s' and t' . Similarly to the proof of Lemma 10, s' and t' must form an entry-exit pair. So, by Lemma 2, any feasible solution must place a tracker in either u or v . By symmetry, we can track and remove u and its edges. Therefore, Reduction 4 is safe by Claim 7 and the facts that u is neither a cut-vertex nor in an entry-exit pair. ◀

B

 Deferred proofs on NP-completeness

To prove Lemma 15, we first prove a couple of helpful lemmas and make a few observations:

- Each vertex in R belongs to a triangle, such that the other two vertices form an entry-exit pair, so must be tracked.
- Each square face, besides the two including h_2 and l_{m_i-1} , requires 3 tracked vertices.
- Two adjacent vertices cannot both be untracked.

► **Lemma 23.** *The true/false assignments of x_i correspond to tracking sets with respect to x_i 's gadget with source s_i and destination t_i .*

Proof. In a true/false assignment of x_i , every face of the gadget is trivially tracked, except the faces including s_i/t_i (which are clearly tracked) and the faces including h_2 and l_{m_i-1} . Though the latter faces may only contain 2 trackers (depending on x_i 's truth value and/or m_i 's parity), they are nevertheless tracked because one of these trackers cannot be in an entry-exit pair. The remaining cycles, i.e. the ones which are not faces, are also trivially tracked: since these cycles must have size at least 6, they must contain 3 trackers given the observation that no adjacent vertices are untracked. ◀

► **Lemma 24.** *In a minimum tracking set, each column has exactly 2 trackers.*

Proof. Since the true/false assignments achieve this property, we only have to show that each column requires at least two trackers. Assume that a minimum tracking set has only one tracker in column $1 < k < m_i$; it must be on μ_k . Then all vertices on columns $k-1, k+1$ must be tracked. For the average number of trackers per column to be at most 2, the number of trackers per column must be an alternating sequence of $\dots 3, 1, 3, 1, \dots$, with column 1 and/or column m_i only having 1 tracker, which contradicts the square face property. ◀

► **Lemma 15.** *★ The true and false assignments are the only minimum tracking sets.*

Proof. Assume that some μ_k is not tracked in a minimum tracking set, for $1 < k < m_i$. By Lemma 24, we only have to show that this causes a column to have 3 trackers. If $k = m_i - 1$ then column m_i must have three trackers. Otherwise, $\mu_{k+1}, h_{k+1}, h_{k+2}$ must be tracked, since they share a square face with μ_k . If l_{k+1} is tracked we are done, otherwise, l_{k+2} must be tracked. Additionally, μ_{k+2} must be tracked, either by the square face property, or in the case where $k = m_i - 2$, because it is in R . Then, column $k+2$ has 3 trackers.

Now, if all the μ_k are tracked, then the only minimum tracking sets are the true and false assignments, by the observation that two adjacent vertices cannot both be untracked and Lemma 23. ◀

► **Lemma 16.** *★ Clause C is satisfied if and only if its corresponding gadget face F_C is tracked.*

Proof. The entry-exit $(\overline{\beta_1}, \overline{\beta_2})$ is the only one, with respect to F_C , that is tracked if and only if C is satisfied. The remaining entry-exit pairs are tracked by either a β_k or a $\overline{\beta_k}$. ◀

► **Theorem 17.** *★ There exists a polynomial time reduction from PLANAR-3-SAT to PLANAR-TRACKING.*

Proof. Let (G, \mathcal{G}) be an instance of PLANAR-TRACKING that results from applying the transformation described above to an instance (ϕ, \mathcal{D}) of PLANAR-3-SAT, where \mathcal{G} and \mathcal{D} are the underlying planar embeddings. We show that ϕ is satisfiable if and only if G has a tracking set of size $T = (\sum_{i=1}^p 2m_i + 4) + p - 1$.

- (\Leftarrow). Choose the truth assignment of each variable according to the given tracking set. The implication follows from Lemma 15 and the fact that if some clause in ϕ was not satisfied, then its gadget face would have been untracked, a contradiction.
- (\Rightarrow). Place T trackers on the literal vertices that correspond to the satisfiable truth assignment of every variable and on all non-literal vertices (except s and t). We show that this corresponds to a tracking set by arguing that every cycle C in G is tracked. We distinguish between two cases:

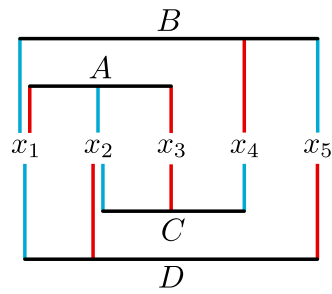
Case 1: C contains no clause edges.

Then C is tracked by (almost) the same argument given in Lemma 23 that shows that a truth assignment of x_i corresponds to a tracking set with respect to x_i 's gadget. Notice that, because of Restriction 1, clause edges are only added to faces that require 3 trackers, so this does not change the argument for the faces which do not require 3 trackers. The only differences are: (i) the addition of the edges that force trackers in every s_i , which only helps the argument, and (ii) the fact that C may span multiple variable gadgets, in which case C must traverse at least 3 trackers on the non-literal vertices between two variable gadgets.

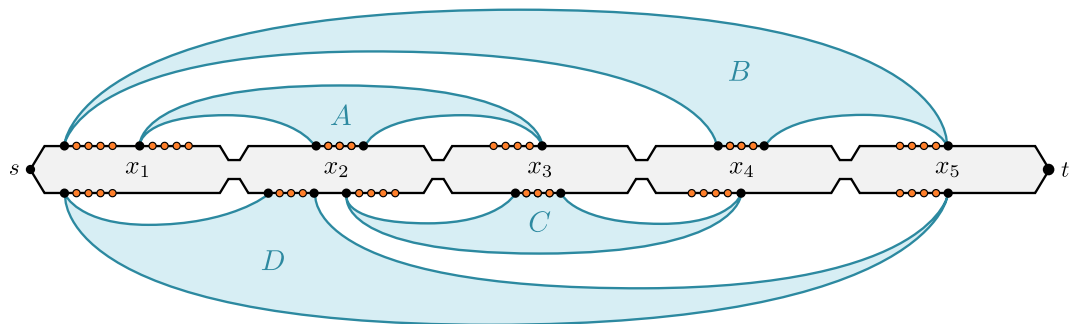
Case 2: C contains clause edges.

Notice that, by construction of G , C must have at least one spine edge. If C corresponds to a clause face, then it must be tracked by Lemma 16. Otherwise, we show that C contains at least 3 trackers and, thus, is trivially tracked. Let us think of C as alternating non-empty paths of two types: *clause paths*, which only contain clause edges and *spine paths*, which only contain spine edges. To avoid dealing with complex cycles, we observe that each spine path in C must contain at least 1 tracker; this follows from the fact at least one of the 2 vertices sharing a spine edge must have a tracker (see variable gadget). Thus, let us assume that C contains no more than 2 spine/clause paths, or otherwise C immediately contains 3 trackers. Notice that if one of the spine paths spans 2 or more variable gadgets, then it must traverse at least 3 trackers on the non-literal vertices between two variable gadgets. Since every clause in ϕ contains exactly 3 distinct literals and C is simple, the only cases where none of the spine paths span multiple variable gadgets are the following:

- (i) C contains exactly 2 clause paths in different sides of the spine.
Then, the 2 spine paths connecting the two sides of the spine must traverse 2 trackers each (see variable gadget). Therefore, C contains at least 4 trackers.
- (ii) C contains exactly 2 clause paths on the same side of the spine, which both start or both end at the same variable gadget, one nested in the other.
Then, by Restriction 2 one of the spine paths contains at least 6 vertices, half of which must be tracked. \blacktriangleleft



(a) Example of a PLANAR-3-SAT instance drawn in a rectilinear fashion with clauses $A = (\overline{x_1} \vee x_2 \vee \overline{x_3})$, $B = (x_1 \vee \overline{x_4} \vee x_5)$, $C = (x_2 \vee \overline{x_3} \vee x_4)$, $D = (x_1 \vee \overline{x_2} \vee \overline{x_5})$.



(b) Example of a PLANAR-TRACKING instance. The orange vertices ensure that no cycle is untracked (see Figure 6).

■ **Figure 10** Illustration of a PLANAR-3-SAT instance (above) and the corresponding PLANAR-TRACKING instance associated with the reduction described in Section 4 (below).

Distance Measures for Embedded Graphs

Hugo A. Akitaya

Department of Computer Science, Tufts University, Medford, MA, USA
hugo.alves_akitaya@tufts.edu

Maike Buchin

Department of Mathematics, Ruhr University Bochum, Bochum, Germany
Maike.Buchin@rub.de

Bernhard Kilgus

Department of Mathematics, Ruhr University Bochum, Bochum, Germany
Bernhard.Kilgus@rub.de

Stef Sijben

Department of Mathematics, Ruhr University Bochum, Bochum, Germany
Stef.Sijben@rub.de

Carola Wenk

Department of Computer Science, Tulane University, New Orleans, LA, USA
cwenk@tulane.edu

Abstract

We introduce new distance measures for comparing straight-line embedded graphs based on the Fréchet distance and the weak Fréchet distance. These graph distances are defined using continuous mappings and thus take the combinatorial structure as well as the geometric embeddings of the graphs into account. We present a general algorithmic approach for computing these graph distances. Although we show that deciding the distances is NP-hard for general embedded graphs, we prove that our approach yields polynomial time algorithms if the graphs are trees, and for the distance based on the weak Fréchet distance if the graphs are planar embedded. Moreover, we prove that deciding the distances based on the Fréchet distance remains NP-hard for planar embedded graphs and show how our general algorithmic approach yields an exponential time algorithm and a polynomial time approximation algorithm for this case. Our work combines and extends the work of Buchin et al. [13] and Akitaya et al. [7] presented at EuroCG.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Fréchet distance, graph comparison, embedded graphs

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.55

Related Version A full version of the paper is available at <https://arxiv.org/pdf/1812.09095.pdf>.

Funding *Hugo A. Akitaya*: Supported by National Science Foundation grants CCF-1422311 and CCF-1423615, and the Science Without Borders scholarship program.

Bernhard Kilgus: Supported by the Deutsche Forschungsgemeinschaft (DFG), project BU 2419/3-1.

Carola Wenk: Supported by National Science Foundation grant CCF-1618469.

1 Introduction

There are many applications that work with graphs that are embedded in Euclidean space. One task that arises in such applications is comparing two embedded graphs. For instance, the two graphs to be compared could be two different representations of a geographic network (e.g., roads or rivers). Oftentimes these networks are not isomorphic, nor is one interested in subgraph isomorphism, but one would like to have a mapping of one graph to the other, and ideally such a mapping would be continuous. For instance, this occurs when we have a ground truth of a road network and a simplification or reconstruction of the same network



© Hugo A. Akitaya, Maike Buchin, Bernhard Kilgus, Stef Sijben, and Carola Wenk;
licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 55; pp. 55:1–55:15

Leibniz International Proceedings in Informatics



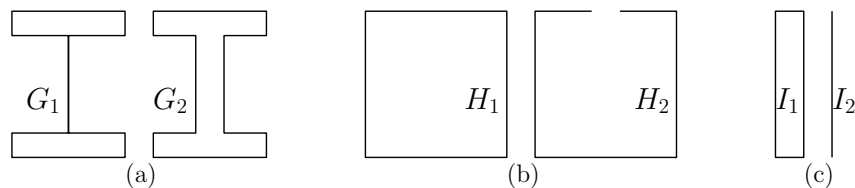
LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

and we would like to measure the error of the latter. In this case, a mapping would identify the parts of the ground truth that are reconstructed/simplified and would allow to study the local error.

We present new graph distance measures that are well-suited for comparing such graphs. Our distance measures are natural generalizations of the Fréchet distance [10] to graphs and require a continuous mapping, but they don't require graphs to be homeomorphic. One graph is mapped continuously to a portion of the other, in such a way that edges are mapped to paths in the other graph. The graph distance is then defined as the maximum of the strong (or weak) Fréchet distances between the edges and the paths they are mapped to. This results in a directed or asymmetric notion of distance, and we define the corresponding undirected distances as the maximum of both directed distances. The directed distances naturally arise when seeking to measure subgraph similarity, which requires mapping one graph to a subgraph of the other.

For comparing two not necessarily isomorphic graphs only few measures were known previously. One such measure is the traversal distance suggested by Alt et al. [9] and another is the geometric edit distance suggested by Cheong et al. [14]. The traversal distance converts graphs into curves by traversing the graphs continuously and comparing the resulting curves using the Fréchet distance. It is also a directed distance that compares the traversal of one graph with the traversal of a part of the other graph. However, an explicit mapping between the two graphs is not established, and part of the connectivity information of the graphs is lost due to the conversion to curves. The geometric edit distance minimizes the cost of edit operations from one graph to another, where cost is measured by Euclidean lengths and distances. But again, connectivity is not well maintained. Figure 1 shows some examples of graphs where our graph distances, the traversal distance, and the geometric edit distance differ. In particular, only our graph distances capture the difference in connectivity between graphs G_1 and G_2 , as well as between H_1 and H_2 .

Our graph distances map one graph onto a subgraph of the other and they measure the Fréchet distance between the mapped parts (see Section 2.1 for a formal definition). Hence connectivity information is preserved and an explicit mapping between the two (sub-)graphs is established. One possible application of these new graph distances is the comparison of geographic networks, for instance evaluating the quality of map reconstructions and map simplification. In Section 5, we show first experimental results on graphs of map reconstructions that illustrate that our approach considers both, geometry and connectivity.



■ **Figure 1** Examples where our graph distances, the traversal distance, and the geometric edit distance differ. For clarity the graphs are shown side-by-side, but in the embedding they lie on top of each other. (a): Graphs G_1 and G_2 have large graph distance (because G_1 needs to be mapped to one side of G_2), large edit distance (because a long edge needs to be added), but small traversal distance. (b): Graphs H_1 and H_2 have large graph distance (because all of H_1 needs to be mapped to only one side of H_2), but small traversal distance and small edit distance. (c): Graphs I_1 and I_2 have small graph distance and small traversal distance, but a large edit distance (because a long edge needs to be added).

Related work

A few approaches have been proposed in the literature for comparing geometric embedded graphs. Subgraph-isomorphism considers only the combinatorial structure of the graphs and not its geometric embedding. It is NP-hard to compute in general, although it can be computed in linear time if both graphs are planar and the pattern graph has constant size [17]. If we consider the graphs as metric spaces, the Gromov-Hausdorff distance (GH) between two graphs is the minimum Hausdorff distance between isometric embeddings of the graphs into a common metric space. While it is unknown how to compute GH for general graphs, recently Agarwal et al. [1] gave a polynomial time approximation algorithm for the GH between a pair of metric trees. We are however interested in measuring the similarity between two specific embeddings of the graphs. Armiti et al. [11] suggest a probabilistic approach for comparing graphs that are not required to be isomorphic, using spatial properties of the vertices and their neighbors. However, they require vertices to be matched to vertices, which can result in a large graph distance when an edge in one graph is subdivided in the other graph. Furthermore, the spatial properties used are invariant to translation and rotation, whereas we consider a fixed embedding. Cheong et al. [14] proposed the geometric edit distance for comparing embedded graphs, however it is NP-hard to compute. Alt et al. [9] defined the traversal distance, which is most similar to our graph distance measures, but it does not preserve connectivity. comparison with the traversal distance.

For assessing the quality of map construction algorithms, several approaches have been proposed. One approach is to compare all paths [2] or random samples of shortest paths [18]. However, these measures ignore the local structure of the graphs. In order to capture more topological information, Biagioni and Eriksson developed a sampling-based distance [12] and Ahmed et al. introduced the local persistent homology distance [3]. The latter distance measure focuses on comparing the topology and does not encode geometric distances between the graphs. The sampling-based distance is not a formally defined distance measure, and it crucially depends on parameters (in particular *matched_distance*, to decide if points are sufficiently close to be matched); in practice it is unclear how these parameters should be chosen. However, it captures the number of matched edges, which is useful when comparing reconstructed road networks. In contrast to these measures, our graph distances capture more topology than the path-based distance [2], and capture differences in geometry better than the local persistent homology distance [3]. Also our graph distances are well-defined distance measures that do not require specific parameters to be set, unlike [12].

Contributions

We present new graph distance measures that compare graphs based on their geometric embeddings while respecting their combinatorial structure. To the best of our knowledge, our graph distances are the first to establish a continuous mapping between the embedded graphs. In Section 2 we define several variants of our graph distances (weak, strong, directed, undirected) and study their properties. In Section 3 we develop an algorithmic approach for computing the graph distances. On the one hand, we prove that for general embedded graphs, deciding these distances is NP-hard. On the other hand, we also show that our algorithmic approach gives polynomial time algorithms in several cases, e.g., when one graph is a tree. The most interesting case is when both graphs are plane. Here, we show that our algorithmic approach yields a quadratic time algorithm for the weak Fréchet distance. In Section 4 we focus on plane graphs and the strong Fréchet distance. For this case, we show that the problem is NP-hard, even though it is polynomial time solvable for the weak Fréchet

distance. Furthermore, we show how to obtain an approximation, that depends on the angle between incident edges, in polynomial time and an exact result in exponential time. For this version, we omit some of the proofs or present only proof sketches. Detailed proofs can be found within the version published on arXiv [8].

2 Graph Distance Definition and Properties

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two undirected graphs with vertices embedded as points in \mathbb{R}^d (typically \mathbb{R}^2) that are connected by straight-line edges. We refer to such graphs as *(straight-line) embedded graphs*. Generally, we do not require the graphs to be planar. We denote a crossing free embedding of a planar graph shortly as a *plane graph*. Note that for plane graphs G_1 and G_2 , crossings between edges of G_1 and edges of G_2 are still allowed.

2.1 Strong and Weak Graph Distance

We define distance measures between embedded graphs that are based on mapping one graph to the other. We consider a particular type of graph mappings, as defined below:

- **Definition 1** (Graph Mapping). *We call a mapping $s: G_1 \rightarrow G_2$ a graph mapping if*
1. *it maps each vertex $v \in V_1$ to a point $s(v)$ on an edge of G_2 , and*
 2. *it maps each edge $\{u, v\} \in E_1$ to a simple path from $s(u)$ to $s(v)$ in the embedding of G_2 .*

Note that a graph mapping results in a continuous map if we consider the graphs as topological spaces. To measure similarity between edges and mapped paths, our graph distances use the Fréchet distance or the weak Fréchet distance, which are popular distance measures for curves [10]. For two curves $f, g: [0, 1] \rightarrow \mathbb{R}^d$ their *Fréchet distance* is defined as

$$\delta_F(f, g) = \inf_{\sigma: [0,1] \rightarrow [0,1]} \max_{t \in [0,1]} \|f(t) - g(\sigma(t))\|,$$

where σ ranges over orientation preserving homeomorphisms. The *weak Fréchet distance* is

$$\delta_{wF}(f, g) = \inf_{\alpha, \beta: [0,1] \rightarrow [0,1]} \max_{t \in [0,1]} \|f(\alpha(t)) - g(\beta(t))\|,$$

where α, β range over all continuous onto functions that keep the endpoints fixed.

Typically, the Fréchet distance is illustrated by a man walking his dog. Here, the Fréchet distance equals the shortest length of a leash that allows the man and the dog to walk on their curves from beginning to end. For the weak Fréchet distance man and dog may walk backwards on their curves, for the Fréchet distance they may not. The Fréchet distance and weak Fréchet distance between two polygonal curves of complexity n can be computed in $O(n^2 \log n)$ time [10]. Now, we are ready to define our graph distance measures.

- **Definition 2** (Graph Distances). *We define the directed (strong) graph distance $\vec{\delta}_G$ as*

$$\vec{\delta}_G(G_1, G_2) = \inf_{s: G_1 \rightarrow G_2} \max_{e \in E_1} \delta_F(e, s(e))$$

and the directed weak graph distance $\vec{\delta}_{wG}$ as

$$\vec{\delta}_{wG}(G_1, G_2) = \inf_{s: G_1 \rightarrow G_2} \max_{e \in E_1} \delta_{wF}(e, s(e)),$$

where s ranges over graph mappings from G_1 to G_2 , and e and its image $s(e)$ are interpreted as curves in the plane. The undirected graph distances are

$$\delta_G(G_1, G_2) = \max(\vec{\delta}_G(G_1, G_2), \vec{\delta}_G(G_2, G_1)) \quad \text{and}$$

$$\delta_{wG}(G_1, G_2) = \max(\vec{\delta}_{wG}(G_1, G_2), \vec{\delta}_{wG}(G_2, G_1)).$$

According to Definition 1, a graph mapping s maps each edge of G_1 to a simple path $s(e)$ in G_2 . This is justified by the following observation: Mapping e to a non-simple path $s'(e)$, where $s(e)$ and $s'(e)$ have the same endpoints and $s(e) \subset s'(e)$, does not decrease the (weak) graph distance because $\delta_{(w)F}(e, s(e)) \leq \delta_{(w)F}(e, s'(e))$. From this observation also follows that we cannot decrease $\vec{\delta}_G(G_1, G_2)$ by adding additional vertices to subdivide an edge e of G_1 : While the concatenation of the resulting mapped paths in G_2 may not be simple, it can be replaced by the image of the entire edge e , which by the observation has to be simple.

We state a first important property of the graph distances:

► **Lemma 3.** *For embedded graphs, the strong graph distances and the weak graph distances fulfill the triangle inequality. The undirected distances are pseudo-metrics. For plane graphs they are metrics.*

Proof. Symmetry follows immediately for the undirected distances. The directed distances fulfill the triangle inequality because we can concatenate two maps and use the triangle inequality of \mathbb{R}^d : Let G_1 , G_2 and G_3 be three embedded graphs. An edge e of G_1 is mapped to a simple path p in G_2 . The segments of p are again mapped to a sequence of simple paths in G_3 . Thus, when concatenating two maps, one possible mapping maps each edge e of G_1 to a sequence S of simple paths in G_3 . Note, that S need not be simple. However, in that case we can instead map e to a shortest path \hat{p} in S from beginning to end. As $\delta_{(w)F}(e, \hat{p}) \leq \delta_{(w)F}(e, S)$ for each edge of G_1 , we have $\vec{\delta}_G(G_1, G_2) + \vec{\delta}_G(G_2, G_3) \geq \vec{\delta}_G(G_1, G_3)$ and $\vec{\delta}_{wG}(G_1, G_2) + \vec{\delta}_{wG}(G_2, G_3) \geq \vec{\delta}_{wG}(G_1, G_3)$ by definition of the directed (weak) graph distance as the maximum Fréchet distance of an edge and its mapping. Analogously, the undirected distances fulfill the triangle inequality as well.

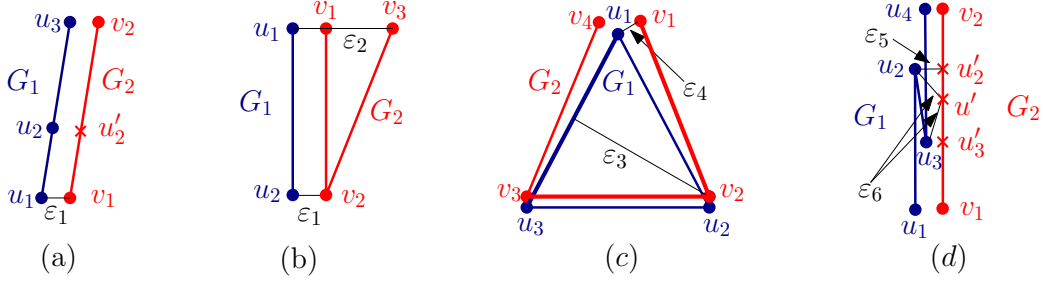
For plane graphs, their (weak) graph distance is zero iff their embeddings are the same, hence the distances are metrics. If the (weak) graph distance is zero, every edge needs to be mapped to itself, hence the embeddings are the same. If on the other hand, the embeddings are the same, a graph mapping may map every edge to itself in the embedding. Since there are no intersections or overlapping vertices, this mapping is continuous in the target graph, and the distance is zero. ◀

Note that for non-plane graphs the (weak) graph distance does not fulfill the identity of indiscernibles. For example, if G_1 consists of two crossing line segment edges, and G_2 has visually the same embedding but consists of four edges and includes the intersection point as a vertex, then both, $\vec{\delta}_G(G_1, G_2) = \vec{\delta}_{wG}(G_1, G_2) = 0$ and $\vec{\delta}_G(G_2, G_1) = \vec{\delta}_{wG}(G_2, G_1) = 0$, and therefore $\delta_G(G_1, G_2) = \delta_{wG}(G_1, G_2) = 0$. Also note that we do not require graph mappings to be injective or surjective. And an optimal graph mapping from G_1 to G_2 may be very different from an optimal graph mapping from G_2 to G_1 . See Figure 2 for examples of graphs and their graph distances.

In [8], we show that the traversal distance between a graph G_1 and a graph G_2 is a lower bound for $\vec{\delta}_{wG}(G_1, G_2)$, which follows from the observation that the traversal distance captures the combinatorial structure of the graphs to a lesser extent than our graph distances. Furthermore, we apply the graph distances to measure the similarity between two polygonal paths to examine how these new definitions are generalizations of the (weak) Fréchet distance for curves to graphs.

3 Algorithms and Hardness for Embedded Graphs

Throughout this paper, let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two straight-line embedded graphs, and let $n_1 = |V_1|$, $m_1 = |E_1|$, $n_2 = |V_2|$ and $m_2 = |E_2|$.



■ **Figure 2** Examples of graph mappings $s_1 : G_1 \rightarrow G_2$ and $s_2 : G_2 \rightarrow G_1$, and the resulting graph distances. Mapped vertices are drawn with crosses and are not graph vertices. (a) $\vec{\delta}_G(G_1, G_2) = \vec{\delta}_G(G_2, G_1) = \varepsilon_1$. $s_1(u_1) = v_1, s_1(u_2) = u'_2, s_1(u_3) = v_2$ and $s_2 = s_1^{-1}$. (b) $\vec{\delta}_G(G_1, G_2) = \varepsilon_1 < \varepsilon_2 = \vec{\delta}_G(G_2, G_1)$. The mapping $s_1(u_1) = v_1$ and $s_1(u_2) = v_2$ is not surjective, and $s_2(v_1) = s_2(v_3) = u_1$ and $s_2(v_2) = u_2$ is not injective. (c) $\vec{\delta}_G(G_1, G_2) = \varepsilon_3 > \varepsilon_4 = \vec{\delta}_G(G_2, G_1)$. $s_1(u_i) = v_i$ and $s_2(v_i) = u_i$ for $i = 1, 2, 3$; $s_2(v_4) = u_1$. (a)-(c) The weak graph distances equal the strong graph distances. (d) $\vec{\delta}_G(G_1, G_2) = \vec{\delta}_{wG}(G_1, G_2) = \vec{\delta}_{wG}(G_2, G_1) = \varepsilon_5 < \varepsilon_6 = \vec{\delta}_G(G_2, G_1)$. Here, the mappings that attain the strong graph distances are $s_1(u_1) = v_1, s_1(u_2) = u'_2, s_1(u_3) = u'_3, s_1(u_4) = v_2$ and $s_2(v_1) = u_1, s_2(v_2) = u_4$, where s_2 in the limit maps u' to all points on the edge from u_2 to u_3 . The mappings attaining the weak graph distances are $s_1^w = s_1$ and $s_2^w = s_1^{-1}$.

First, we consider the decision variants for the different graph distances defined in Definition 2. Given G_1 and G_2 and a value $\varepsilon > 0$, the decision problem for the graph distances is to determine whether $\vec{\delta}_G(G_1, G_2) \leq \varepsilon$ (resp., $\vec{\delta}_{wG}(G_1, G_2) \leq \varepsilon$). Equivalently, this amounts to determining whether there exists a graph mapping from G_1 to G_2 realizing $\vec{\delta}_G(G_1, G_2) \leq \varepsilon$ (resp., $\vec{\delta}_{wG}(G_1, G_2) \leq \varepsilon$). Note that the undirected distances can be decided by answering two directed distance decision problems. As we show in Section 3.3, the value of ε can be optimized by parametric search.

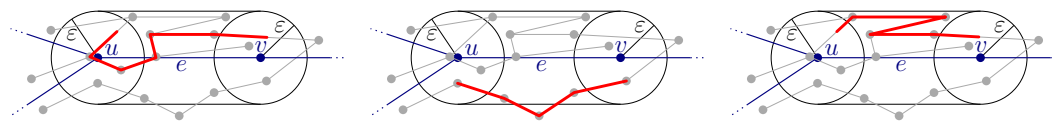
In Section 3.1 we describe a general algorithmic approach for solving the decision problems by computing *valid ε -placements* for vertices. We show that for general embedded graphs the decision problems for the strong and weak directed graph distances are NP-hard, see Section 3.2. However, we prove in Section 3.3 that our algorithmic approach yields polynomial-time algorithms for the strong graph distance if G_1 is a tree, and for the weak graph distance if G_1 is a tree or if both are plane graphs. In the latter scenario (G_1 and G_2 plane graphs), deciding if $\vec{\delta}_G(G_1, G_2) \leq \varepsilon$ remains NP-hard, see Section 4.1.

3.1 Algorithmic Approach

Recall, that a (directional) graph mapping that realizes a given distance ε maps each vertex of G_1 to a point in G_2 and each edge of G_1 to a simple path in G_2 within this distance. In order to determine whether such a graph mapping exists, we define the notion of *ε -placements* of vertices and edges; see Figures 3 and 4 (a).

► **Definition 4** (ε -Placement). *An ε -placement of a vertex v is a maximally connected part of G_2 restricted to the ε -ball $B_\varepsilon(v)$ around v . An ε -placement of an edge $e = \{u, v\} \in E_1$ is a path P in G_2 connecting placements of u and v such that $\delta_F(e, P) \leq \varepsilon$. In that case, we say that C_u and C_v are reachable from each other. An ε -placement of G_1 is a graph mapping $s : G_1 \rightarrow G_2$ such that s maps each edge e of G_1 to an ε -placement.*

A weak ε -placement of an edge $e = \{u, v\}$ is a path P in G_2 connecting placements of u and v such that $\delta_{wF}(e, P) \leq \varepsilon$. A weak ε -placement of G_1 is a graph mapping $s : G_1 \rightarrow G_2$ such that s maps each edge e of G_1 to a weak ε -placement.



(a) An ε -placement of e . (b) Not an ε -placement. (c) A weak ε -placement.

■ **Figure 3** (a) Illustration of ε -placements of an edge e . (b) Not an ε -placement because the path leaves the ε -tube around e . (c) The Fréchet distance is too large, but e can be mapped to the path if backtracking is allowed. Thus, it is a weak ε -placement.



■ **Figure 4** Illustration of valid and invalid vertex placements. (a) Placements u_3 (resp. v_3) are invalid because they are not connected to a placement of v (resp. u) by an ε -placement of the edge e . Placement v_2 is valid when considering e in isolation, but it cannot connect to a placement for the edge that leaves v to the right. Thus, it is also invalid. As a result of pruning v_2 (right), u_2 becomes invalid as well, leaving only u_1 and v_1 as potentially valid placements of u and v (b).

Note that an ε -placement of a vertex v consists of edges and portions of edges of G_2 , depending whether $B_\varepsilon(v)$ contains both, one or zero endpoint(s) of the edge, see Figure 4. Also note that each vertex has $O(m_2)$ ε -placements, since an ε -placement is defined as a connected part of G_2 of maximal size inside $B_\varepsilon(v)$. Furthermore, we consider two graph mappings s_1 and s_2 from G_1 to G_2 to be equivalent in terms of the directed (weak) graph distance if for each vertex $v \in V_1$, $s_1(v)$ and $s_2(v)$ are points on the same ε -placement of v .

General Decision Algorithm

Our algorithm consists of the following four steps, which we describe in more detail below. We assume ε is fixed and use the term *placement* for an ε -placement.

Observe that each connected component of G_1 needs to be mapped to a connected component of G_2 , and each connected component of G_1 can be mapped independently of the other components of G_1 . Hence we can first determine the connected components of both graphs, and then consider mappings between connected components only. In the following we present an algorithm for determining if a mapping from G_1 to G_2 , that realizes a given distance ε , exists, where both G_1 and G_2 are connected graphs.

■ **Algorithm 1** General Decision Algorithm.

-
- 1: Compute vertex placements.
 - 2: Compute reachability information for vertex placements.
 - 3: Prune invalid placements.
 - 4: Decide if there exists a placement for the whole graph G_1 .
-

1. Compute vertex placements

We iterate over all vertices $v \in V_1$ and compute all their placements. Each vertex has $O(m_2)$ placements, so the total number of vertex-placements is $O(n_1 \cdot m_2)$, and they can be computed in $O(n_1 \cdot m_2)$ time using standard algorithms for computing connected components.

2. Compute reachability information of vertex placements

Next, we iterate over all edges $e = \{u, v\} \in E_1$ to determine all placements of its vertices that allow a placement of the edge. That is, we search for all pairs of vertex-placements C_u, C_v that are reachable from each other according to Definition 4.

For the weak graph distance, we need to find all pairs of placements of u and placements of v that can reach one another using paths contained in the ε -tube $T_\varepsilon(e)$ around e , i.e., the set of all points with distance $\leq \varepsilon$ to a point on e , see Figure 3 (c). If we restrict G_2 to its intersection with the ε -tube, all placements in the same connected component are mutually reachable. Thus, each edge is processed in time linear in the size of G_2 using linear space per edge: For each connected component a pair of lists containing the placements of u and v in that component, respectively, is computed. So, all reachability information can be computed in $O(m_1 \cdot m_2)$ time and space. Note that the weak Fréchet distance between a straight line edge $e \in E_1$ and a simple path $s(e)$ in G_2 is the maximum of the Hausdorff distance between e and $s(e)$ and the distances of the endpoints of e and $s(e)$.

For the strong graph distance, existence of a path inside the ε -tube is not sufficient to describe the connectivity between placements. We must ensure that the Fréchet distance between e and P is at most ε , i.e., a continuous and monotone map s must exist from e to P such that $\delta_F(t, s(t)) \leq \varepsilon$ for all $t \in e$. This can be decided in $O(|P|)$ time using the original dynamic programming algorithm for computing the Fréchet distance [10]. In order to determine whether such a path P exists, every placement of u stores a list of all placements of v that are reachable. The connectivity information can be computed by running a graph exploration, starting from each placement, which prunes a branch if the search leaves the ε -tube or backtracking on e is required to map it. This method runs a search for every placement of the start vertex and thus needs $O(m_2^2)$ time per edge of G_1 . Since the connectivity is explicitly stored as pairs of placements that are mutually reachable, it also needs $O(m_2^2)$ space per edge. Hence, in total over all edges, $O(m_1 \cdot m_2^2)$ time and space are needed. Summing up, we have:

► **Lemma 5.** *To run step 1 and step 2 of Algorithm 1, we need $O(m_1 \cdot m_2)$ time and space for the weak graph distance and $O(m_1 \cdot m_2^2)$ time and space for the strong graph distance.*

3. Prune invalid placements

Now, after having processed all vertices and edges, it still needs to be decided whether G_1 as a whole can be mapped to G_2 . To this end, we delete *invalid* placements of vertices.

► **Definition 6 (Valid Placement).** *An ε -placement C_v of a vertex v is (weakly) valid if for every neighbor u of v there exists an ε -placement C_u of u such that C_v and C_u are connected by a (weak) ε -placement of the edge $\{u, v\}$. Otherwise, C_v is (weakly) invalid.*

See Figure 4 for an illustration of (in)valid placements. As shown in the Figure, deleting an invalid placement possibly sets former valid placements to be invalid. Thus, we need to process all placements recursively until all invalid placements are deleted and no new invalid placements occur. Note that the ordering of processing the placements does not affect the final result. To decide which placements of vertices u and v incident to an edge e are valid, we use the reachability information computed in Step 2.

Initially there are $O(n_1 \cdot m_2)$ vertex-placements, each of which may be deleted once. For the weak graph distance, connectivity is stored using connected components inside the ε -tube surrounding an edge $\{u, v\}$. On deleting a placement C_v of v , it is removed from the list containing placements of v . If a component no longer contains placements of v (i.e. its list becomes empty), then all placements of u in that component become invalid. A placement

C_v is deleted at most once and upon deletion it must be removed from one list for every edge incident to v . Thus, the time for pruning C_v is $O(\deg(v))$. Since the sum of all degrees is $2m_1$, all invalid placements can be pruned in $O(m_1 \cdot m_2)$ time. For the strong graph distance, every placement has a list of placements to which it is connected. On deleting C_v , it must be removed from the lists of all placements C_u to which C_v is connected. Each vertex has $O(m_2)$ placements which have to be removed from a list for each neighbor of v . Thus, pruning a placement runs in $O(\deg(v) \cdot m_2)$ time and pruning all invalid placements in $O(m_1 \cdot m_2^2)$ time.

► **Lemma 7.** *Pruning all invalid placements takes $O(m_1 \cdot m_2)$ time for the weak graph distance and $O(m_1 \cdot m_2^2)$ time for the strong graph distance.*

Note that after the pruning step all remaining vertex placements are (weakly) valid. However, the existence of a (weakly) valid placement for each vertex is not a sufficient criterion for $\vec{\delta}_G(G_1, G_2)$ ($\vec{\delta}_{wG}(G_1, G_2)$) in general, see Figure 6.

4. Decide if there exists a placement for the whole graph G_1

After pruning all invalid placements, we want to decide if the remaining valid vertex-placements allow a placement of the whole graph G_1 . The complexity of this step depends on the graph and the distance measure: for plane graphs we show that we can concatenate weakly valid placements of two adjacent faces (Lemma 12), whereas this is not possible for the directed strong graph distance in this setting (Theorem 15) or for general graphs for both distances (Theorem 10). Although deciding the directed (weak) graph distance is NP-hard for general graphs, there are two settings which may occur after running steps 1-3 of Algorithm 1, making step 4 of the algorithm trivial. Clearly $\vec{\delta}_G(G_1, G_2) > \varepsilon$ ($\vec{\delta}_{wG}(G_1, G_2) > \varepsilon$) if there is a vertex that has no (weakly) valid ε -placement. Furthermore, we have the following:

► **Lemma 8.** *If, after running steps 1-3 of Algorithm 1, each internal vertex (degree at least two) has exactly one valid ε -placement (resp., weakly valid ε -placement) and each vertex of degree one has at least one valid ε -placement (resp., weakly valid ε -placement), then G_1 has an ε -placement (resp., weak ε -placement). Thus, $\vec{\delta}_G(G_1, G_2) \leq \varepsilon$ (resp., $\vec{\delta}_{wG}(G_1, G_2) \leq \varepsilon$).*

Lemma 5, Lemma 7 and Lemma 8 imply the following Theorem.

► **Theorem 9.** *If there is a vertex that has no valid ε -placement or if each vertex has exactly one valid ε -placement after running steps 1-3 of Algorithm 1, the directed strong graph distance can be decided in $O(m_1 \cdot m_2^2)$ time and space. Analogously, if there is a vertex that has no weakly valid ε -placement or if each vertex has exactly one weakly valid ε -placement after running steps 1-3 of Algorithm 1, the directed weak graph distance can be decided in $O(m_1 \cdot m_2)$ time and space.*

3.2 NP-Hardness for the General Case

Notwithstanding the special cases in Theorem 9, deciding the (weak) graph distance is not tractable for general graphs.

► **Theorem 10.** *Deciding whether $\vec{\delta}_G(G_1, G_2) \leq \varepsilon$ and deciding whether $\vec{\delta}_{wG}(G_1, G_2) \leq \varepsilon$ for two graphs G_1 and G_2 embedded in \mathbb{R}^2 is NP-hard.*

Proof Sketch. We show NP-hardness reducing from binary constraint satisfaction problem (CSP) by identifying each variable x_i with a vertex v_i of G_1 and each constraint on two variables x_i, x_j with an edge incident to v_i and v_j . Furthermore, for every possible value

for a variable, we add one vertex to G_2 and embed the vertex inside an ε -ball around the variable. We connect two of such vertices corresponding to two different variables with an edge iff the two values satisfy the constraint. Now, deciding the CSP is equivalent to decide if every edge of G_1 can be mapped to a path of G_2 consisting of a single edge. ◀

3.3 Efficient Algorithms for Plane Graphs and Trees

Here, we show that that Algorithm 1 yields polynomial-time algorithms for deciding the strong graph distance if G_1 is a tree (Theorem 14), and the weak graph distance if G_1 is a tree or if both are plane graphs (Theorem 13). More precisely, we show that the existence of at least one (weakly) valid placement for each vertex is a sufficient condition for $\vec{\delta}_G(G_1, G_2) \leq \varepsilon$ or $\vec{\delta}_{wG}(G_1, G_2) \leq \varepsilon$.

► **Lemma 11.** *If G_1 is a tree and every vertex of G_1 has at least one (weakly) valid ε -placement after running steps 1-3 of Algorithm 1, then G_1 has a (weak) ε -placement. Thus, $\vec{\delta}_G(G_1, G_2) \leq \varepsilon$ (or $\vec{\delta}_{wG}(G_1, G_2) \leq \varepsilon$).*

Proof Sketch. We view G_1 as a rooted tree, selecting an arbitrary vertex as the root. Now we can greedily map all vertices of G_1 from the root outwards because all placements are valid and no cycle exists where we need to ensure that we start and end in the same valid placement when traversing the cycle. ◀

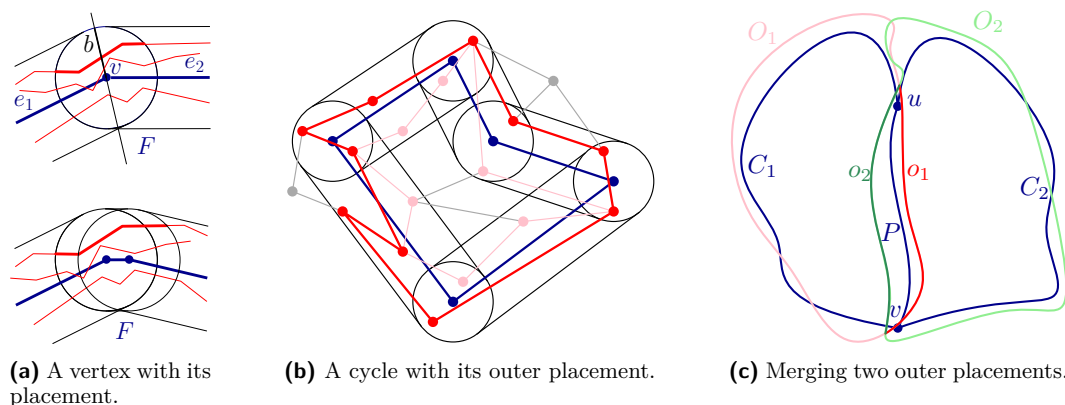
► **Lemma 12.** *If G_1 and G_2 are plane graphs and every vertex of G_1 has at least one weakly valid ε -placement after running steps 1-3 of Algorithm 1, then G_1 has a weak ε -placement. Thus, $\vec{\delta}_{wG}(G_1, G_2) \leq \varepsilon$.*

Proof. A *tree-substructure* of G_1 is a tree $T = (V_T, E_T)$ induced by the vertex set $V_T \subset V_1$ with a root vertex $r \in V_T$, such that for all vertices $v \in V_T$, $v \neq r$, v is not an endpoint of an edge $e \in E_1 \setminus E_T$ and such that T is maximal, in the sense that when adding one additional vertex, T contains a cycle. We first remove all tree-substructures of G_1 and map these as in the proof of Lemma 11. Next, we consider all faces of the remainder of G_1 and show how to iteratively map them.

Consider a cycle C bounding a face F and let e_1 and e_2 be two edges of C incident to a vertex v . Let b be the line segment of the bisector of e_1 and e_2 inside $B_\varepsilon(v)$. We define the *outermost placement* of v as the placement which intersects b at maximum distance to the endpoint of b inside F , see Figure 5 (a). Furthermore, we define an *outermost path* in G_2 of an edge $e = \{u, v\}$ of G_1 as the path P_{out} with maximum distance to F connecting the outermost placements of u and v . That is, no subpath Q of P_{out} can be replaced by a path R such that $\delta_H(R, B) \leq \delta_H(Q, B)$, where δ_H is the Hausdorff distance and B is the boundary of the tube $T_\varepsilon(e)$ which lies inside the face F . Note that if an edge is shorter than 2ε , and hence the ε -balls around the vertices overlap, then so possibly do the placements. In particular, in this case the outer placements may overlap, in which case the edge placements degenerate, see Figure 5 (a). Finally, we define an *outer placement* O of C in G_2 as the concatenation of all outermost paths of edges of C .

Note that if C is sufficiently convex the outer placement is simply the cycle that bounds H . See Figure 5 (b) for an example, where the red outer placement bounds the outer face of G_2 restricted to red and pink vertices and edges. The outer placement of C is a weak ε -placement of C .

Now, consider two cycles C_1 and C_2 bounding adjacent faces of G_1 , which share a single (possibly degenerate) path P between vertices u and v . Let O_1 and O_2 be the outer placements of C_1 and C_2 , respectively. By definition of an outermost placement, O_1 and O_2



■ **Figure 5** Illustration of outer placements and how to merge them. In (c) the outer placements of cycles C_1 and C_2 can be merged by mapping the shared path P through o_1 .

must intersect inside the intersection of the ε -tubes of C_1 and C_2 . Let o_1 and o_2 of O_1 and O_2 be the parts between the intersections of O_1 and O_2 containing the respective images of P . Again, by definition of an outermost placement, it holds that o_1 is completely inside O_2 and o_2 is completely inside O_1 .

This is illustrated in Figure 5 (c). By planarity there must be a vertex at the intersections of O_1 and O_2 . Thus, we can construct a mapping O'_2 of C_2 that consists of o_1 and $O_2 \setminus o_2$. This is a weak ε -placement of C_2 for which the image of the shared path P is identical to its image in O_1 . Thus, we can merge O_1 and O'_2 to obtain a weak ε -placement of these two adjacent cycles. Note that the mapping of C_1 is not modified in this construction. Additionally, the image of the cycle bounding the outer face is its outer placement. The same argument can be applied iteratively when C_1 and C_2 share multiple paths.

If there are two cycles C_1 and C_2 which are connected by a path P such that one endpoint u of P lies on C_1 , the other endpoint v of P lies on C_2 and all other vertices of P are no vertices of C_1 or C_2 , we can still construct a common placement for C_1 , C_2 and P : Let C_u , C_v be the outermost placements of u and v , respectively and let D_v be a valid placement of v which is connected by a path Q in G_2 to C_u such that $\delta_{wF}(Q, P) \leq \varepsilon$. Such a placement D_v must exist as C_u is a valid placement. If $D_v = C_v$ we have found a common valid placement for C_1 , C_2 and P . If $D_v \neq C_v$, by definition of an outermost placement, the path Q must intersect the outermost placement O of C_2 inside the intersection of the tubes $T_\varepsilon(P)$ and $T_\varepsilon(C_2)$. As G_2 is plane, there is a vertex w at the intersection and the resulting path $R = Q_{C_u \rightarrow w} + O_{w \rightarrow C_v}$ with $\delta_{wF}(R, P) \leq \varepsilon$ connects C_u and C_v .

Now, we iteratively map the cycles bounding faces of G_1 until G_1 is completely mapped. Let $\langle F_1, F_2, \dots, F_k \rangle$ be an ordering of the faces of G_1 such that each F_i , for $i \geq 2$ is on the outer face of the subgraph $\mathbb{G}_{i-1} := C_1 \cup C_2 \cup \dots \cup C_{i-1}$ of G_1 , where C_j is the cycle bounding face F_j . Thus, let F_1 be an arbitrary face of G_1 and subsequently choose faces adjacent to what has already been mapped. Hence when adding a cycle C_i , we have already mapped \mathbb{G}_{i-1} such that the cycle bounding its outer face is mapped to its outer placement. Thus, we can treat \mathbb{G}_{i-1} as a cycle, ignoring the part of it inside this cycle, and merge its mapping with C_i using the procedure described above. This leaves the mapping of \mathbb{G}_{i-1} unchanged, hence this is still a weak ε -placement of \mathbb{G}_{i-1} . However, the mapping of C_i is now modified to be identical to that of \mathbb{G}_{i-1} in the parts where they overlap. Thus, we can merge these mappings to obtain a weak ε -placement of \mathbb{G}_i . After mapping F_k we have completely mapped G_1 . ◀

55:12 Distance Measures for Embedded Graphs

Lemma 5 and Lemma 7 together with Lemma 11 and Lemma 12 directly imply the following theorems. Note, that $m_1 = O(n_1)$ for plane graphs and trees, in particular.

► **Theorem 13** (Decision Algorithm for Weak Graph Distance). *Let $\varepsilon > 0$. If G_1 is a tree, or if G_1 and G_2 are plane graphs, then Algorithm 1 decides whether $\vec{\delta}_{wG}(G_1, G_2) \leq \varepsilon$ in $O(n_1 \cdot m_2)$ time and space.*

► **Theorem 14** (Decision Algorithm for Graph Distance). *Let $\varepsilon > 0$. If G_1 is a tree, then Algorithm 1 decides whether $\vec{\delta}_G(G_1, G_2) \leq \varepsilon$ in $O(n_1 \cdot m_2^2)$ time and space.*

Computing the Distance

To compute the graph distance, we proceed as for computing the Fréchet distance between two curves: We search over a set of critical values and employ the decision algorithm in each step. The following types of critical values can occur:

1. A new vertex-placement emerges: An edge in G_2 is at distance ε from a vertex in G_1 .
2. Two vertex-placements merge: The vertex in G_2 where they connect is at distance ε from a vertex in G_1 .
3. The (weak) Fréchet distance between a path and an edge is ε : these are described in [10]. There are exponentially many paths in G_2 , but each value the Fréchet distance may attain is defined by either a vertex and an edge, or two vertices and an edge.

There are $O(n_1 \cdot m_2)$ critical values of the first two types, and $O(m_1 \cdot n_2^2)$ of type three. Parametric search can be used to find the distance as described in [10], using the decision algorithms from Theorems 13 and 14. This leads to a running time of $O(n_1 \cdot m_2 \cdot \log(n_1 + n_2))$ for computing the weak graph distance if G_1 is a tree or both are plane graphs. And the total running time for computing the graph distance if G_1 is a tree is $O(n_1 \cdot m_2^2 \cdot \log(n_1 + n_2))$.

4 Hardness Results and Algorithms for Plane Graphs

Lemma 12 does not hold for plane graphs and the directed strong graph distance because in general outer placements of cycles cannot be combined to a placement of G_1 as shown in the proof of Lemma 12, see Figure 6 for a counterexample. In fact we show that deciding the directed strong graph distance for plane graphs is NP-hard.

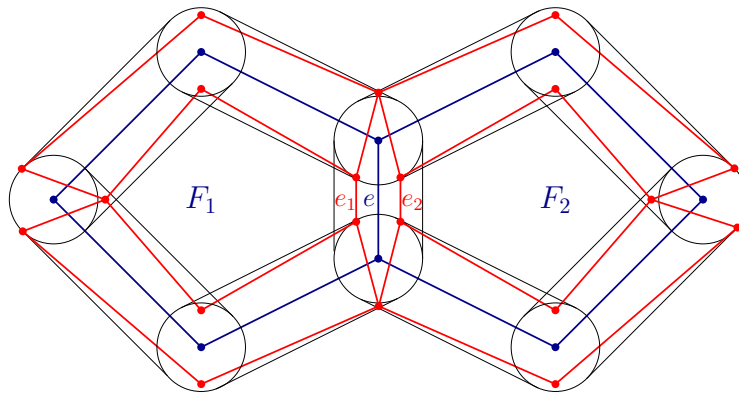
4.1 NP-Hardness for the Strong Distance for Plane Graphs

► **Theorem 15.** *For plane graphs G_1 and G_2 , deciding whether $\vec{\delta}_G(G_1, G_2) \leq \varepsilon$ is NP-hard.*

Proof Sketch. We prove the NP-hardness by a reduction from MONOTONE-PLANAR-3-SAT. In this 3-SAT variant, the associated graph with edges between variables and clauses is planar and each clause contains only positive or only negative literals. We construct two graphs G_1 and G_2 where each vertex of G_1 has at least two valid placements. The equivalence of a MONOTONE-PLANAR-3-SAT solution and a valid mapping is obtained by zig-zag shapes of G_2 inside the ε -Ball of some of the vertices of G_1 . See Figure 6 for an illustration. ◀

The following stronger result follows from the observation that characteristics of the subgraphs we constructed in the proof of Theorem 15 still hold for a slightly larger ε value.

► **Theorem 16.** *It is NP-hard to approximate $\vec{\delta}_G(G_1, G_2)$ within a 1.10566 factor.*



■ **Figure 6** An example of plane graphs G_1 (blue) and G_2 (red) where every vertex of G_1 has two valid placements, but there is no ε -placement of G_1 : If the central edge e is mapped to a path through e_1 , there is no way to map the cycle bounding face F_2 on the right, and if e is mapped to a path through e_2 , the cycle bounding F_1 cannot be mapped.

4.2 Deciding the Strong Graph Distance in Exponential Time

A brute-force method to decide the directed strong graph distance is to iterate over all possible combinations of valid vertex placements, which takes $O(m_1 \cdot m_2^{n_1})$ time. Another approach is to decompose G_1 into faces and merge the substructures bottom-up. This approach is exponential in the number of faces. For more details, see [8].

► **Theorem 17.** *For plane graphs, the strong graph distance can be decided in $O(Fm_2^{2F-1})$ time and $O(m_2^{2F-1})$ space, where F is the number of faces of G_1 .*

Thus, this method is superior to the brute-force method if $2F - 1 \leq n_1$.

4.3 Approximation for Plane Graphs

For plane graphs, Algorithm 1 yields an approximation depending on the angle between the edges for deciding the strong graph distance. The decision is based on the existence of valid placements. Therefore, the runtime is the same as stated in Theorem 14.

► **Theorem 18.** *Let $G_1 := (V_1, E_1)$ and $G_2 := (V_2, E_2)$ be plane graphs. Assume that for all adjacent vertices $v_1, v_2 \in V_1$, $B_\varepsilon(v_1)$ and $B_\varepsilon(v_2)$ are disjoint. Let α_v be the smallest angle between two edges of G_1 incident to vertex v with $\deg(v) \geq 3$, and let $\alpha := \frac{1}{2} \min_{v \in V_1}(\alpha_v)$. If there exists at least one valid ε -placement for each vertex of G_1 , then $\vec{d}_G(G_1, G_2) \leq \frac{1}{\sin(\alpha)}\varepsilon$.*

Proof Sketch. For $\hat{\varepsilon} := \frac{1}{\sin(\alpha)}\varepsilon$ the union of all ε -placements of a vertex v with $\deg(v) \geq 3$ form a single connected component of G_2 inside $B_{\hat{\varepsilon}}$. Thus, when mapping two adjacent cycles separated by a path P , we ensure that for both mappings the same $\hat{\varepsilon}$ -placements of the start and endpoints of P are used. ◀

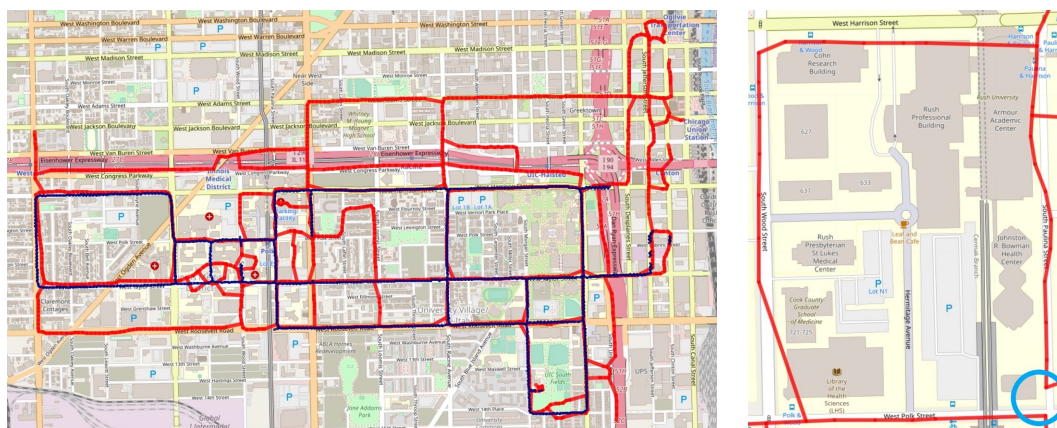
5 Experiments on Road Networks

In the last decade several algorithms have been developed for reconstructing maps from the trajectories of entities moving on the network [4, 5]. This naturally asks to assess the quality of such reconstruction algorithms. Recently, Duran et al [16] compared several of

these algorithms on hiking data, and found that inconsistencies often arise due to noise and low sampling of the input data, for example unmerged parallel roads or the addition of short off-roads.

When assessing the quality of a network reconstruction from trajectory data, several aspects have to be taken into account. Two important aspects are the *geometric* and *topological error* of the reconstruction. Another important aspect is the *coverage*, i.e., how much of the network is reconstructed from the data. We believe our measures to be well suited for assessing the geometric error while still maintaining connectivity information.

We have used the weak graph distance for measuring the distance between different reconstructions and a ground truth of a part of the road network of Chicago. Figure 7 (a) shows two reconstructed road map graphs R (red) and B (blue), overlaid on the underlying ground truth road map G from OpenStreetMap. The reconstruction R in red resulted from Ahmed et al.'s algorithm [6], whereas the reconstruction B in blue from Davies et al.'s [15] algorithm. Our directed graph distance from B to G is 25 meters, and from R to G it is 90 meters. This reflects the local geometric error of the reconstructions (note that it does not evaluate the difference in coverage). Figure 7 (b) shows an example where the topology of R and G differs (blue circle), affecting for instance navigation significantly. This is captured by our distance. Although the reconstruction approximates the geometry well, our measure computes a directed distance of 200 m from G (restricted to the part covered by R) to R .



(a) Two partial map reconstructions of Chicago.

(b) Different topology.

■ **Figure 7** Two reconstructed road map graphs R (red) and B (blue), overlaid on the underlying ground truth road map G from OpenStreetMap.

6 Conclusion

We developed new distances for comparing straight-line embedded graphs and presented efficient algorithms for computing these distances for several variants of the problem, as well as proving NP-hardness for other variants. Our distance measures are natural generalizations of the Fréchet distance and the weak Fréchet distance to graphs, without requiring the graphs to be homeomorphic. Although graphs are more complicated objects than curves, the runtimes of our algorithms are comparable to those for computing the Fréchet distance between polygonal curves. A large-scale comparison of our approach with existing graph similarity measures is left for future work.


References

- 1 Pankaj K. Agarwal, Kyle Fox, Abhinandan Nath, Anastasios Sidiropoulos, and Yusu Wang. Computing the Gromov-Hausdorff Distance for Metric Trees. *ACM Trans. Algorithms*, 14(2):24:1–24:20, April 2018. doi:10.1145/3185466.
- 2 Mahmuda Ahmed, Brittany T. Fasy, Kyle S. Hickmann, and Carola Wenk. Path-based distance for street map comparison. *ACM Transactions on Spatial Algorithms and Systems*, 28 pages, 2015.
- 3 Mahmuda Ahmed, Brittany Terese Fasy, and Carola Wenk. Local Persistent Homology Based Distance Between Maps. In *22nd ACM SIGSPATIAL GIS*, pages 43–52, 2014.
- 4 Mahmuda Ahmed, Sophia Karagiorgou, Dieter Pfoser, and Carola Wenk. A comparison and evaluation of map construction algorithms using vehicle tracking data. *GeoInformatica*, 19(3):601–632, 2015.
- 5 Mahmuda Ahmed, Sophia Karagiorgou, Dieter Pfoser, and Carola Wenk. *Map Construction Algorithms*. Springer, 2015.
- 6 Mahmuda Ahmed and Carola Wenk. Constructing Street Networks from GPS Trajectories. In *Proceedings of the 20th Annual European Conference on Algorithms*, ESA'12, pages 60–71, Berlin, Heidelberg, 2012. Springer-Verlag.
- 7 Hugo Akitaya, Maike Buchin, and Bernhard Kilgus. Distance Measures for Embedded Graphs - Revisited. In *35th European Workshop on Computational Geometry (EuroCG)*, 2019.
- 8 Hugo A. Akitaya, Maike Buchin, Bernhard Kilgus, Stef Sijben, and Carola Wenk. Distance Measures for Embedded Graphs. *CoRR*, abs/1812.09095, 2018. arXiv:1812.09095.
- 9 Helmut Alt, Alon Efrat, Günter Rote, and Carola Wenk. Matching planar maps. *Journal of Algorithms*, 49(2):262–283, 2003.
- 10 Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5(1&2):75–91, 1995.
- 11 Ayser Armiti and Michael Gertz. Geometric graph matching and similarity: A probabilistic approach. *ACM International Conference Proceeding Series*, June 2014.
- 12 James Biagioni and Jakob Eriksson. Inferring Road Maps from Global Positioning System Traces: Survey and Comparative Evaluation. *Transportation Research Record: Journal of the Transportation Research Board*, 2291:61–71, 2012.
- 13 Maike Buchin, Stef Sijben, and Carola Wenk. Distance Measures for Embedded Graphs. In *Proc. 33rd European Workshop on Computational Geometry (EuroCG)*, pages 37–40, 2017.
- 14 Otfried Cheong, Joachim Gudmundsson, Hyo-Sil Kim, Daria Schymura, and Fabian Stehn. Measuring the similarity of geometric graphs. In *International Symposium on Experimental Algorithms*, pages 101–112, 2009.
- 15 Jonathan J. Davies, Alastair R. Beresford, and Andy Hopper. Scalable, Distributed, Real-Time Map Generation. *IEEE Pervasive Computing*, 5(4):47–54, 2006.
- 16 David Duran, Vera Sacristán, and Rodrigo I. Silveira. Map Construction Algorithms: An Evaluation Through Hiking Data. In *Proceedings of the 5th ACM SIGSPATIAL International Workshop on Mobile Geographic Information Systems*, MobiGIS '16, pages 74–83, 2016.
- 17 David Eppstein. Subgraph Isomorphism in Planar Graphs and Related Problems. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '95, pages 632–640, Philadelphia, PA, USA, 1995. Society for Industrial and Applied Mathematics.
- 18 Sophia Karagiorgou and Dieter Pfoser. On vehicle tracking data-based road network generation. In *20th ACM SIGSPATIAL GIS*, pages 89–98, 2012.

Online Algorithms for Warehouse Management

Philip Dasler 

Department of Computer Science, University of Maryland, College Park, USA
daslerpc@cs.umd.edu

David M. Mount 

Department of Computer Science, University of Maryland, College Park, USA
mount@umd.edu

Abstract

As the prevalence of E-commerce continues to grow, the efficient operation of warehouses and fulfillment centers is becoming increasingly important. To this end, many such warehouses are adding automation in order to help streamline operations, drive down costs, and increase overall efficiency. The introduction of automation comes with the opportunity for new theoretical models and computational problems with which to better understand and optimize such systems.

These systems often maintain a warehouse of standardized portable storage units, which are stored and retrieved by robotic workers. In general, there are two principal issues in optimizing such a system: where in the warehouse each storage unit should be located and how best to retrieve them. These two concerns naturally go hand-in-hand, but are further complicated by the unknown request frequencies of stored products. Analogous to virtual-memory systems, the more popular and oft-requested an item is, the more efficient its retrieval should be. In this paper, we propose a theoretical model for organizing portable storage units in a warehouse subject to an online sequence of access requests. We consider two formulations, depending on whether there is a single access point or multiple access points. We present algorithms that are $O(1)$ -competitive with respect to an optimal algorithm. In the case of a single access point, our solution is also asymptotically optimal with respect to density.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Warehouse management, online algorithms, competitive analysis, robotics

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.56

Funding *David M. Mount*: Research supported by NSF grant CCF-1618866.

1 Introduction

Online shopping has grown rapidly in recent years and, as such, the efficiency of the warehouses and fulfillment centers that support it plays an increasingly important role. Several companies have developed automated systems to help streamline operations in these warehouses, drive down the costs of order fulfillment, and increase overall efficiency. The introduction of automation comes with the opportunity for new theoretical models and computational problems with which to better understand and optimize such systems.

These systems often maintain a warehouse of standardized portable storage units, which are stored and retrieved by robots [12, 14]. For example, Amazon’s Kiva robots and Alibaba’s Quicktron robots help to streamline the order-fulfillment process. The Amazon robots are 16 inches tall, weigh almost 145 kilograms, can travel at 5 mph, and carry a payload weighing up to 317 kilograms. These robots maneuver themselves under standardized shelving units, lift them from below, and carry them to a location in the warehouse where a human waits to complete an order with items from the shelf.

The frequency with which each storage unit is accessed varies, and so, intuitively, units that are accessed more often should be placed closer to the access points than those that are less frequently accessed. As access probabilities vary over time, there is a natural question of



© Philip Dasler and David M. Mount;

licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 56; pp. 56:1–56:21

Leibniz International Proceedings in Informatics



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

how to dynamically organize the warehouse's placement of storage units in order to guarantee efficient access at any time. In this paper we will develop a simple computational model for a "self-organizing warehouse," and we present online algorithms for solving them. We demonstrate that our algorithms are competitive with optimal algorithms in our model. Our work can be viewed as a geometric variant of online algorithms for self-organizing lists and virtual memory management systems [1, 19].

From a practical perspective there are many ways in which to model objects residing in a warehouse. In order to obtain meaningful theoretical results without imposing irrelevant technical details, we propose a very simple and general model, which encapsulates the most salient aspects of efficient self-organizing behavior. We model storage units, or *boxes*, as movable unit squares on a grid in the plane. In addition to the boxes, there are designated fixed points, called *access points*, where boxes are brought on demand. The input consists of a sequence of *access requests*, each specifying that a particular box in the system be moved to a given access point.

There are two natural ways in which to move boxes in a planar setting, picking them up (like cargo containers by an overhead crane) and sliding them along the ground (like the aforementioned robotic systems). The former is simpler to describe and analyze. The latter is more realistic and is consistent with other motion-planning models [11, 10]. Another issue is the geometrical configuration of the warehouse and the locations of the access points. We present clean and simple models based on infinite and semi-infinite grids and show how to generalize our solutions to rectangular warehouses.

We consider two versions of the problem: the *attic problem*, where there is a single access point and the *warehouse problem*, where there are multiple access points. In each version and for each motion type, we present an online algorithm that is competitive with respect to an optimal solution that has knowledge of the entire access sequence. Details of the problem formulations and results are given in the next section.

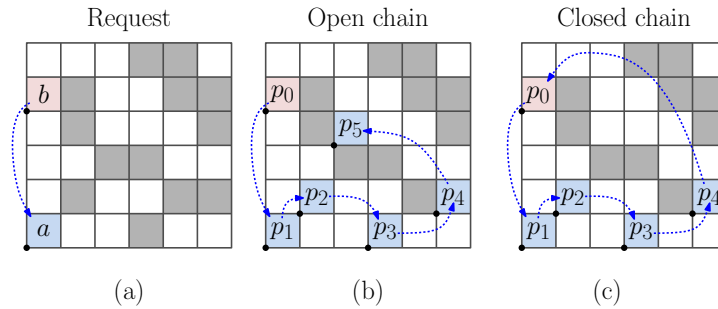
1.1 Model and Results

We model a warehouse as a rectangular subset Ω of \mathbb{Z}^2 , the square grid in the plane. Throughout, distances are measured in the ℓ_1 metric (the sum of absolute differences in x and y coordinates). We are given a finite set $A = \{a_1, \dots, a_m\} \subseteq \Omega$ of stationary *access points* and a (significantly larger) finite set $B = \{b_1, \dots, b_n\}$ of portable storage units, called *boxes*. Each box is a unit square. At any time, its lower left corner coincides with a grid point in Ω , called its *location*. A point of Ω that contains a box is said to be *occupied*, and otherwise it is *unoccupied*. No two boxes may occupy the same location at the same time.

The initial layout of the boxes is specified in the input. This is followed by a sequence of *access requests*, each being a pair (b, a) , which involves moving box $b \in B$ from its current location to access point $a \in A$. Access requests are processed *sequentially*, meaning that each request is completed before the next one is started. Since the access point may already be occupied, it will be necessary to reorganize box locations. This reorganization should be performed with care, keeping frequently accessed boxes near the access point and moving less frequently accessed boxes to the periphery. The challenge is that we do not know the future access sequence, and yet we wish to be competitive with an optimal algorithm that does.

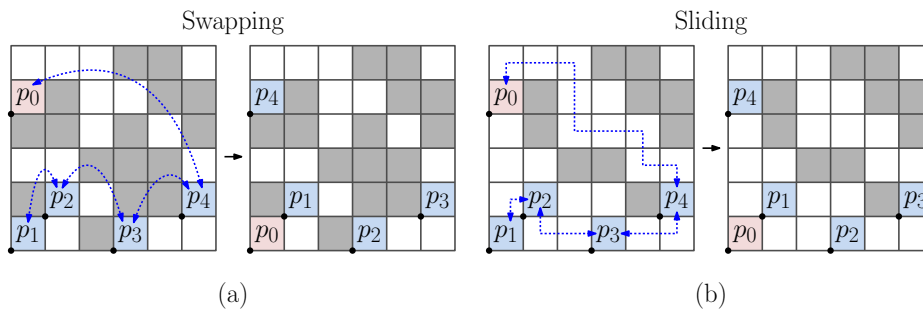
In general, the reorganization following each access request will involve a sequence of box movements. The box at the access point is displaced to a nearby location, the box at this location is then displaced to a new location, and so on. This chain of box movements continues until the last box in the chain arrives at an unoccupied square of the grid, possibly the original location of the requested box. More formally, let p_0 denote the original location

of b , and let p_1 denote the location of a . If a is not occupied, b is simply moved here, and we are done. Otherwise, the algorithm determines a chain p_2, \dots, p_k of locations, where p_2, \dots, p_{k-1} are occupied and p_k is unoccupied (see Fig. 1(a)). (Note that p_0 is considered to be unoccupied, because its box has been moved to the access point.) We call this a *reorganization chain*. If $p_k \neq p_0$, this is an *open chain* (see Fig. 1(b)), and otherwise it is a *closed chain* (see Fig. 1(c)).



■ **Figure 1** Processing a request.

For the sake of presenting our algorithms, it will be useful to describe the relocation process in terms of a sequence of *motion primitives*. In the case where boxes can be picked up (as by an overhead crane), the primitive operation is a *swap*, which exchanges the contents of two grid squares. The *cost* of the operation is the ℓ_1 distance between the two locations. The aforementioned reorganization involving a chain $\langle p_0, \dots, p_k \rangle$ (whether open or closed) can be executed by swapping boxes in reverse order along the chain, that is, $p_k \leftrightarrow p_{k-1} \leftrightarrow \dots \leftrightarrow p_0$ (see Fig. 2(a)).



■ **Figure 2** Motion primitives.

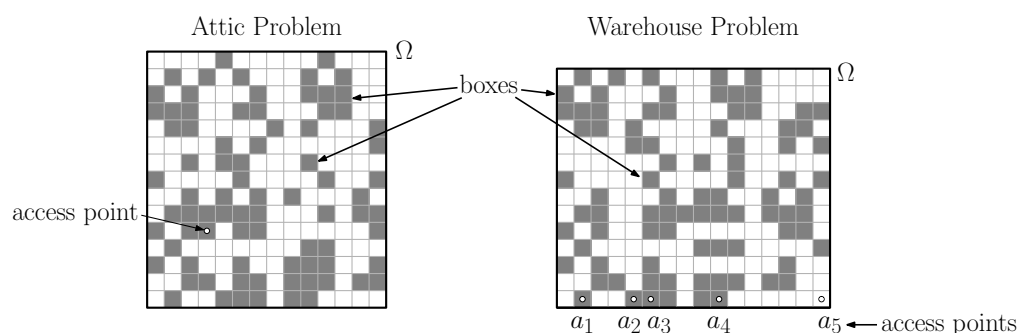
Alternatively, when boxes are moved along the ground the associated primitive operation is called *sliding*. As with swapping, the contents of two grid locations are swapped, but the boxes are moved along a rectilinear path of unoccupied grid locations (see Fig. 2(b)). The *cost* of the operation is the ℓ_1 length of the path, which may generally be much higher than the ℓ_1 distance between the two locations.

Sliding motion is more relevant in contexts where the boxes are being moved by robots, but it is complicated by the need to create empty space in which to move boxes. Our solutions will be based on first presenting a simple swapping-based solution and then showing how to adapt this to sliding motion without significantly increasing the cost. These two primitives provide a conceptually clear and simple model of motion costs. Of course, in practice, many other realistic issues would need to be considered.

Our problem formulations involve a *problem instance*, which consists of a specification of the domain Ω and the locations of the m access points A . An input to a given instance consists of the initial locations of the n boxes followed by a sequence S of access requests. For each access request, the output consists of the sequence $\langle p_0, \dots, p_k \rangle$ along which motion primitives are applied (either swapping or sliding, depending on the model). Since our focus is on reorganization strategies, we ignore a number of issues needed for a complete model, such as how to coordinate the movement of multiple robots. We focus on two versions of the problem depending on the number of access points (see Fig. 3):

Attic Problem: Ω is an axis-aligned rectangle containing a single access point.

Warehouse Problem: Ω is an axis-aligned rectangle with access points on its bottom side.



■ **Figure 3** Problem versions.

We consider the above problems in an *online* setting, which means that each access request is processed without knowledge of future requests. In contrast, in an *offline* setting the entire access sequence is known in advance. An online algorithm is said to be *c-competitive* for a constant $c \geq 1$, called the *competitive ratio*, if for all sufficiently long access sequences S , the total cost of this algorithm is at most a factor c larger than the cost of an optimal offline solution for the same sequence. We say that an algorithm is *competitive* if it is *c-competitive* for some constant c , independent of m , n , the size of the domain, and the length of the access sequence. (The competitive ratios that result from our analyses are relatively high, and we suspect that they are far from tight. Reducing them will involve establishing better lower bounds on the optimum algorithm, and this seems to be quite challenging.) The notion of “sufficiently long access sequence” allows us to ignore start-up issues, such as the initial locations of the boxes.

Our main results are competitive online algorithms for these two problems in both the swapping- and sliding-motion models (presented in Theorems 1, 9, 10, and 12). Our result for the attic problem has the additional feature of being asymptotically optimal with respect to box density. (The precise definition will be given in Section 2.3.) These online algorithms exploit an intriguing connection between these problems and the task of maintaining hierarchical memory systems [1]. Hierarchical memory systems are linear in nature, and the geometric context of the our problems introduces novel challenges, since the reorganization must take into account the 2-dimensional locations of the boxes. Also, when sliding is involved, it is necessary to manage the set of unoccupied squares to guarantee short access paths.

1.2 Prior Work

There have been a number of papers devoted to the problem of organizing storage units in warehouses. Much of the prior work has focused on solving the various engineering challenges involved.

For example, Amato et al. [2] study control algorithms for the warehouse robots, assuming a continuous distribution of item locations throughout the warehouse and ignoring the benefits of intelligent item placement. In a similar vein, Chang et al. [5] attempt to minimize unnecessary task repetition using genetic algorithms, thus shortening robot travel times, but assume a fixed storage scheme regardless of differing access frequencies. Sarrafzadeh and Maddila [17] use a discrete grid-based model, as we will, but their focus is still an engineering one, concerned primarily with robot path-finding and constructing clearings through which to move. Closer to our work, Pang and Chan [16] address the question of where certain items should be stored in the warehouse, proposing a data-mining approach to determine the relationships between products and co-locating those that are often purchased together. Experimental analysis shows that their methodology outperforms a simple greedy policy, but they do not present any proofs on the performance of their approach.

The word “warehouse” has been used for various optimization problems. In the context of operations research, the *warehouse problem* was proposed by Cahn [3] and later refined and extended by Charnes and Cooper [6] and Wolsey and Yaman [20]. This work may sound related to ours, but its focus is on the logistics of managing a warehouse’s stock in the face of changing demand. The word is also used in the context of coordinated motion planning under the name of the *warehouseman’s problem*. This is a multi-agent motion planning problem amidst obstacles. It has been shown to be PSPACE-hard [11, 10], but efficient solutions exist for restricted versions (see, e.g., [18]).

While our approach is theoretical in nature, we avoid the high complexity of the warehouseman’s problem by restricting shapes of boxes (to unit squares) and the allowed layout of boxes (by introducing additional empty working space throughout to facilitate easy motion). The problems we study are less focused on motion planning and more on how to organize the warehouse’s contents to ensure efficient processing of a series of access requests.

More closely related to our work is the *dial-a-ride problem* [7]. In this problem, a set of users must be conveyed from source locations to specified destinations in a metric space. The goal is to plan a route (or routes, in the case of multiple vehicles or the more general k -server problem [13]) that satisfies all transportation requests while minimizing total distance traveled. One key difference is that the source locations are fully specified by the problem input, whereas in the warehouse problem the location of requested boxes can be adjusted according to need, and how best to do so is central to the problem.

As mentioned earlier, our work is similar in spirit to online algorithms for self-organizing memory structures [1, 19]. Another example is the work of Fekete and Hoffmann [8], who consider the online problem of packing variously sized squares into a dynamically sized square container.

2 Online Solution to the Attic Problem

In this section we present an online algorithm for the attic problem (single access point). We will show that the resulting scheme is competitive with respect to an optimal algorithm. As mentioned above, we exploit ideas from hierarchical memory systems. In such systems, memory consists of objects called *pages*, which are organized into blocks, called *caches*. Successive caches have higher storage capacity but higher access times. A common method

for organizing such memory structures involves a block-based version of the least-recently used (LRU) policy, called *Block-LRU* of Aggarwal et al. [1]. In this policy, whenever a page is accessed it is brought to the lowest level cache, and the page that has resided in this cache for the longest time is evicted to the next higher level cache. The process is repeated until reaching the lowest cache that has space to hold this page, possibly the cache that contained the originally requested page. We next describe how the Block-LRU algorithm can be adapted to our geometric setting.

2.1 Hierarchical Model

In hierarchical memory systems, the cost of accessing an object is purely a function of each cache's speed. In our geometric context, the cost depends on the total cost of the motion primitives, which depends on the ℓ_1 distances between the locations of the boxes in the reorganization chain. The principal challenge is adapting the cache-based cost to the geometric setting. Our approach to the attic problem is based on surrounding the access point by collection of nested regions, called *containers*. Analogous to caches in the hierarchical memory systems, containers that are closer to the access point provide faster access but have lower storage capacity compared with those farther out.

It will simplify matters to describe the solution first for the infinite grid. We define a *hierarchical model*, which is based on an infinite sequence of nested *containers*, C_0, C_1, \dots , where C_0 consists only of the origin (the access point), and for $k \geq 1$, C_k consists of the points of \mathbb{Z}^2 that whose ℓ_1 distance from the origin varies from $2^{k-1} + 1$ to 2^k (see Fig. 4 below). Whenever a box b is requested, it is first moved to the access point, and then a series of *evictions* takes place, where, for $k = 0, 1, \dots$ a box from container C_k is moved to container C_{k+1} . The precise manner in which this is done for swapping and sliding motions is explained in Sections 2.2 and 2.3, respectively.

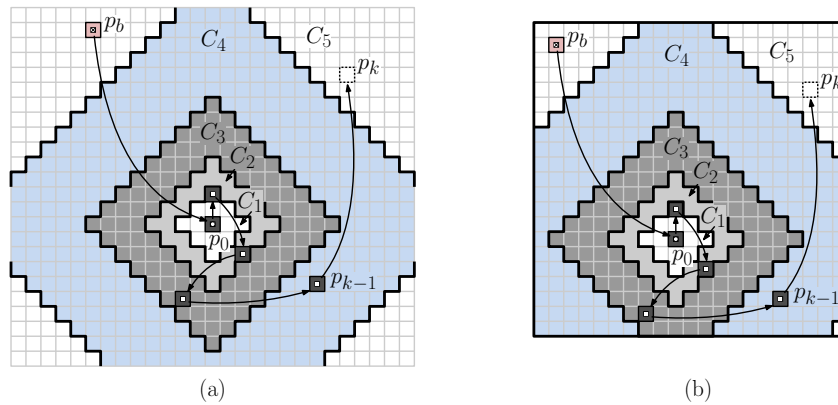
2.2 Online Algorithm for Swapping Motion

In this section we present an online algorithm solving the attic problem in the case of swapping motion, called Block-LRU_A . Consider a request for a box b . If the access point is unoccupied, we simply move the box there. Otherwise, in order to make space for b , we evict the least-recently accessed box from C_0 , C_1 , and so on until we encounter the first container C_k that has at least one unoccupied location (including possibly b 's location at the time of the request). More formally, let p_b denote b 's location, let p_0 denote the access point (origin), and let p_1, \dots, p_{k-1} denote the locations of the least-recently used boxes of containers C_1 through C_{k-1} , respectively. Finally, let $p_k \in C_k$ denote the final unoccupied location (possibly the former location of b). As described in Section 1.1, we achieve this by performing swaps in reverse order $p_k \leftrightarrow p_{k-1} \leftrightarrow \dots \leftrightarrow p_0 \leftrightarrow p_b$ (see Fig. 4(a)). The cost is the sum of the ℓ_1 distances between consecutive pairs.

In order to apply this for a rectangular domain Ω , we simply clip the boundary of the containers at the limits of Ω (see Fig. 4(b)). We show next that this is competitive.

► **Theorem 1.** *For any instance of the attic problem and any sufficiently long access sequence R , the cost of $\text{Block-LRU}_A(S)$ is within a constant factor of the cost of an optimal solution, assuming swapping motion.*

Due to space limitations, the full proof and competitive analysis appear in Appendix A.1. In essence, the containers are treated as the caches of a memory hierarchy and then the standard LRU analysis of [19] and the Block-LRU analysis of [1] are adapted to our case.



■ **Figure 4** (a) Nested containers for the attic problem and (b) restriction to a rectangular domain.

2.3 Online Algorithm for Sliding Motion

In order to accommodate the added constraints involved in sliding boxes around the space, we constrain the manner in which boxes are arranged throughout the domain in order to retrieve them efficiently. An obvious solution would be to arrange the boxes in rows connected by empty corridors, as in typical warehouses. However, this is not efficient asymptotically, because it implies that the number of unoccupied squares in any region of space is at least a constant fraction of the available space. We will adopt a more space-efficient approach by packing distant boxes more densely. While these distant boxes will require more cost to access, this cost can be amortized against the cost incurred by their distance from the access point.

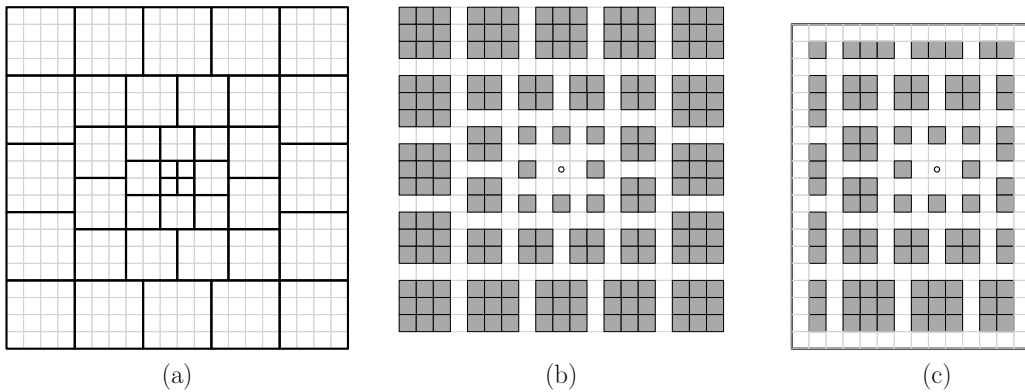
To make this formal, we define a *layout scheme* to be a subset of the integer grid \mathbb{Z}^2 , which we will think of as a subset of the unit squares. For each integer s , define $n(s)$ to be the number of squares of the layout that lie within an $s \times s$ square that is centered about origin. Define the *asymptotic density* to be the limiting ratio of the fraction of squares in the layout lying within such origin-centered squares, that is, $\lim_{s \rightarrow \infty} n(s)/s^2$. For example, the layout that places boxes at every point of the grid has an optimal asymptotic density of 1, and a layout that places boxes only on the white squares of an infinite chessboard has an asymptotic density of $1/2$.

In this section, we describe a layout that achieves the optimal asymptotic density of 1 and show how to convert our swapping-based Block-LRU_A algorithm to the sliding context at the expense of an additional constant factor in cost.

2.3.1 The Nicomachus Layout

Our layout scheme is inspired by a well-known visual proof of Nicomachus’s Theorem [15], which is shown in Fig. 5(a).¹ The grid is partitioned into expanding concentric *rings* of square regions, denoted r_1, r_2, \dots . The innermost ring, r_1 , consists of 4 unit squares. Ring r_2 consists of eight copies of a 2×2 square region surrounding r_1 . In general, r_k consists of $4k$ copies of a $k \times k$ square region surrounding r_{k-1} .

¹ Nicomachus’s Theorem states that $\sum_{k=1}^n k^3 = \left(\sum_{k=1}^n k\right)^2$. If both sides of the equation are multiplied by 4, the layout of Fig. 5(a) provides a proof, where the left side arises by summing the number of blocks ring-by-ring (the k th ring has $4k$ blocks, each with k^2 squares) and the right side comes from the overall area (since the side length of the n th ring is $n(n+1) = 2\sum_{k=1}^n k$).



■ **Figure 5** (a) A geometric tiling based on Nicomachus's Theorem, (b) the associated layout scheme, and (c) restricted to a rectangular domain.

Our layout for the warehouse problem, called the *Nicomachus layout*, is constructed as follows. For each ring r_k of the aforementioned structure and for each $k \times k$ square region of this ring, we include the $(k - 1) \times (k - 1)$ unit squares in the upper left corner in the layout (shaded in Fig. 5(b).) Each of these is called a *block*. We designate the upper-left cell of ring r_1 to be the access point. Finally, to accommodate a rectangular domain Ω , we clip the layout to the boundary of the rectangle and remove the layout squares touching the domain's boundary, thus creating corridors along the domain walls (see Fig. 5(c)). Observe that each block is surrounded by corridors that are one square wide. We show next that this layout achieves an optimal asymptotic density.

► **Lemma 2.** *The Nicomachus layout achieves an asymptotic density of 1.*

Proof. It suffices to show that the *asymptotic wastage*, that is, the asymptotic density of the complement of the Nicomachus layout is equal to zero. To see this, consider the first $\ell \geq 1$ rings of the layout. Each ring r_k , $1 \leq k \leq \ell$, consists of $4k$ blocks, each of size $(k - 1) \times (k - 1)$. The unused space per block is $k^2 - (k - 1)^2 = 2k - 1$. Thus, the total wasted space for ring k is $4k(2k - 1)$. Summing over all rings, the total wastage is $\sum_{k=1}^{\ell} 4k(2k - 1) = 8\ell^3/3 + O(\ell^2)$. The first ℓ rings fill an origin-centered square of side length $\ell(\ell + 1)$, which yields a total area of at least ℓ^4 . Therefore, ignoring lower-order terms, the wastage for these rings is at most $(8\ell^3/3)/\ell^4 = 8/3\ell$. Clearly, this tends to zero in the limit. (Expressed as a function of n , the asymptotic density is the limit of $1 - 8/(3n^{1/4})$.) ◀

2.3.2 Accessing a Box

In order to access a box in the warehouse a robot must first travel to the block in which that box resides, retrieve it from the block, and then return it to the access point. The *depth* d of a box is defined to be the minimum number of boxes between it and the boundary of the block that contains it. So, a box on the perimeter of a block has depth $d = 0$, while one at the center of a block in ring r_i has depth $d = \lfloor \frac{i-2}{2} \rfloor$. (When the domain Ω is bounded, this is an upper bound since peripheral blocks may be clipped.)

In the Nicomachus layout, the cost of reaching a box in the arrangement and retrieving it from a block are both a function of the ring in which it resides. Let $M(r_i)$ denote the maximum cost of moving the robot from the access point to any cell adjacent to a block of ring r_i , and let $C(r_i)$ be the maximum cost of retrieving a box from a block in ring r_i . First, let us consider the travel cost of reaching a cell on the perimeter of a block of boxes.

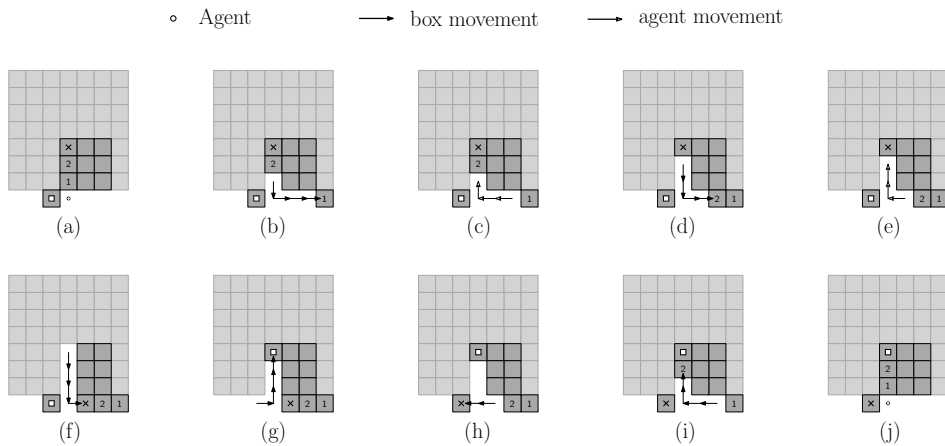
► **Lemma 3.** *Travelling from the access point to any cell adjacent to a block in ring r_i requires at most $i^2 + i$ steps.*

Proof. To reach a box on the perimeter of a block in ring r_i from the access point a robot must traverse each ring $k \leq i$ by circumnavigating one of its blocks. It is easy to see that a robot can move between any two cells adjacent to a $(k - 1) \times (k - 1)$ block of ring r_k in $2k$ steps, from which we conclude that the total travel time is

$$M(r_i) \leq \sum_{k=1}^i 2k = i(i + 1) \leq i^2 + i. \quad \blacktriangleleft$$

An equivalent distance is traveled to return the requested box to the access point.

Next, let us define a primitive $\text{Replace}(d)$ that allows for the swapping of a box b_i placed in the aisle adjacent to a block B with a box $b_j \in B$ at depth d . For now we will use this primitive to establish an upper bound on the cost of accessing a box, while the need for actually swapping boxes will not become apparent until later. Conceptually, the Replace primitive must unbury the target box by moving the d boxes in the way. It does so by moving them each $d + 1$ spaces away, retrieving the target box, and then replacing them for a total cost $O(d^2)$. A more careful analysis yields the following.



■ **Figure 6** Swapping a pair of boxes, where the original box is at depth $d = 2$ within a 7×7 block in ring r_8 .

► **Lemma 4.** *The cost of $\text{Replace}(d)$ is at most $4d^2 + 8d + 6$, where d is the depth of box b_j .*

Proof. First, number the boxes inward from box b_j 's nearest boundary from 1 to d . We assume that the robot begins adjacent to box 1 and that box b_i is adjacent to the robot. Next, we iteratively move each of the $d + 1$ boxes (the d labeled boxes plus b_j) to a location that is $d + 2$ units away along the side of the block (see Fig. 6). Accounting for the time to reach each box, pick it up, move it, put it down, and return to a position adjacent to the next box to be moved, each iteration has a total cost of $2d + 3$, except the last which does not require moving to the next box and so only costs $d + 2$. In total, moving these boxes costs $d(2d + 3) + (d + 2) = 2d^2 + 4d + 2$. Next, we reverse the process at the same cost, replacing box b_j with box b_i and restoring boxes 1 through d to their original positions. This process is briefly interrupted to move box b_j out of the way, adding a cost of 2 (Fig. 6(h)). Thus, in total, swapping a new box with an interior box comes at a cost of $2(2d^2 + 4d + 2) + 2 = 4d^2 + 8d + 6$. ◀

The depth of a box is bounded by the radius of the block in which it resides. Specifically, a box in ring r_i has a depth $d \leq \frac{i-2}{2}$ and so, along with Lemma 4, we have the following corollary:

► **Corollary 5.** *Retrieving a box from a block in ring r_i has a cost of $C(r_i) \leq i^2 + 2$.*

Combining this corollary and Lemma 3, the total cost to move to a box in ring r_i , retrieve it, and return to the access point is at most

$$(i^2 + i) + (i^2 + 2) + (i^2 + i) = 3i^2 + 2i + 2 \quad (1)$$

Next, let us consider retrieval cost as a function of distance from the access point.

► **Lemma 6.** *If a box is at ℓ_1 distance δ from the access point then it lies in a ring r_i , such that $i \leq \sqrt{3\delta}$.*

Proof. To reach the highest ring level possible at a distance δ , travel orthogonally in a straight line, traversing each ring's width in turn. As ring r_i has width i , the farthest ring that can be reached is the first ring r_i such that

$$\delta \leq \sum_{j=0}^i j = \frac{i^2 + i}{2} \quad (2)$$

Solving for i yields $i \geq \sqrt{2\delta + \frac{1}{4}} - \frac{1}{2}$.

It is easily seen that for all $\delta \geq 1$, $\sqrt{3\delta} \geq \sqrt{2\delta + \frac{1}{4}} - \frac{1}{2}$, thus $i = \sqrt{3\delta}$ suffices as an upper bound for the greatest ring index at a distance no more than δ . ◀

By combining Eq. (1) and Lemma 6, we obtain the following.

► **Lemma 7.** *In the Nicomachus layout, retrieving a box at ℓ_1 distance δ from the access point is $O(\delta)$.*

Proof. Eq. (1) shows that retrieving a box in ring r_i has a maximum total cost of $3i^2 + 2i + 2$ and Lemma 6 shows that a box at distance δ will be in some ring r_i , where $i \leq \sqrt{3\delta}$. So, retrieving a box at distance δ incurs at most a cost of $3(\sqrt{3\delta})^2 + 2\sqrt{3\delta} + 2 = 9\delta + 2\sqrt{3\delta} + 2$, which is $O(\delta)$. ◀

From this we find that trading the positions of two boxes can be done at a cost proportional to the sum of their ℓ_1 distances from the access point. A simple, naive algorithm could use the access point as an intermediary, accessing both boxes at cost $O(\delta)$, and returning them to their opposing rather than original positions. Thus, we have the following:

► **Corollary 8.** *If two boxes b_i and b_j are at ℓ_1 distances δ_i and δ_j from the access point, respectively, then the cost of swapping them is no more than $c(\delta_i + \delta_j)$, for some constant c .*

Given this corollary, we can now show that Block-LRU_A is competitive in the sliding model. From the proof of Theorem 1 and the structure of Block-LRU_A, it suffices to bound the cost of evictions from each of the containers. For any $k \geq 0$, consider an eviction from container C_k to C_{k+1} . The contribution of this eviction to $W_{\text{lrn}}(S)$ is 2^k . By Corollary 8, the cost of sliding one to the other is at most $c(2^{k-1} + 2^k) < 2c2^k$, implying that the sliding cost is within a constant factor of the eviction cost (roughly 4). From the proof of Theorem 1 the eviction cost can be used as a proxy for its actual cost, and therefore the sliding cost is

at most a constant factor more than the actual cost of Block-LRU_A in the case of swapping motion. This implies that the cost of Block-LRU_A in the sliding motion model is competitive with the optimum solution in the swapping motion model. The actual cost of the optimum algorithm in the sliding model cannot be lower than the actual cost of the optimum algorithm in the swapping model. With a roughly factor-4 cost ratio between the sliding and swapping models, the overall ratio is roughly 128. While this competitive ratio may be rather high, the analysis thus far has assumed worst case scenarios across multiple factors and the focus has been to prove the general competitiveness rather than finding the best competitive ratio. We are confident that an empirical experiment would likely show that the average case scenario has a much more favorable competitive ratio. Regardless, as a consequence of the above discussion, we have:

► **Theorem 9.** *For any instance of the attic problem and any sufficiently long access sequence S , the cost of Block-LRU_A(S) is within a constant factor of the cost of an optimal solution, assuming sliding motion.*

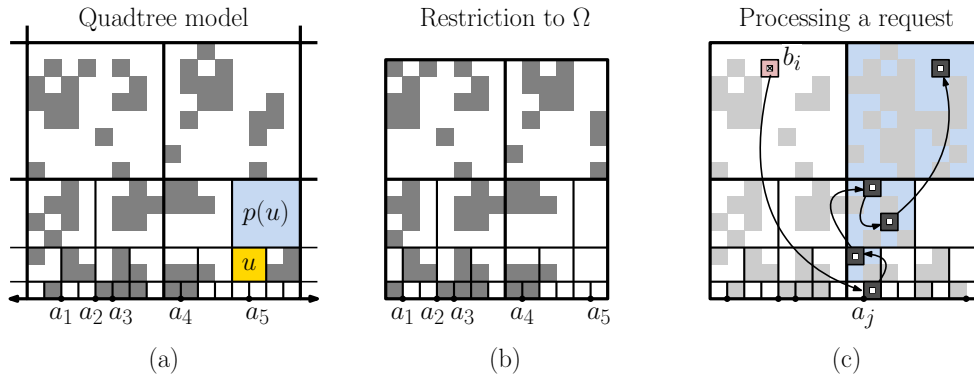
3 Online Solution to the Warehouse Problem

In this section we present an online algorithm for the warehouse problem. As before, we will present the algorithm for swapping motion and then generalize to sliding motion. Recall that the warehouse problem differs from the attic problem in that there are multiple access points, all of which lie on the bottom side of the rectangular domain Ω , which we may assume lies on the x -axis. Our algorithm, which we call Block-LRU_W, will be similar in spirit to online algorithms for hierarchical memory systems, but the combination of spatial locations and multiple access points adds considerable complexity. As with the attic problem, it will simplify matters to describe the algorithm first in an infinite context, where boxes may be placed anywhere above the x -axis, and then adjust the solution to the case of a rectangular domain. Our approach will be to define containers based on a quadtree-like structure above the x -axis, and to evict boxes up the quadtree from child to parent. We will treat each quadtree cell as if it were a cache in the memory hierarchy, with the least-recently used box evicted whenever more space is needed.

3.1 Quadtree Model

As mentioned above, our online solution to the warehouse problem employs a quadtree subdivision over the positive- y halfspace. The leaves of the quadtree, or *level 0*, consist of the unit squares whose lower left corners are the grid points on the x -axis, that is, $(x, 0)$ for $x \in \mathbb{Z}$. Level 1 consists of the 2×2 squares lying immediately above whose lower left corners are located at $(2x, 1)$ for $x \in \mathbb{Z}$. In general, for $k \geq 0$, level- k consists of the $2^k \times 2^k$ squares whose lower left corners lie on $(2^k x, 2^k - 1)$, for $x \in \mathbb{Z}$. Each level- k node u has a parent $p(u)$ of twice the side length lying immediately above on level $k + 1$ (see Fig. 7(a)), and two children each of half the side length lying immediately below on level $k - 1$. The set of unit squares associated with each node of the quadtree is called its *cell*. This structure covers the infinite grid lying above the x -axis. Given a rectangular domain Ω whose lower side lies along the x -axis, we clip the above structure to this rectangle (see Fig. 7(b)).

To simplify the analysis of our solution, we first define a variant of the warehouse problem with an alternate cost function based on this quadtree structure, which we call the *quadtree model*. Of course, an optimal solution does not need to follow this model, and later, we will



■ **Figure 7** Quadtree layout.

relate the cost of the standard solution to this variant. The processing of requests in this model differs from the standard model (described in Section 1.1) in that, after moving the box to the desired access point, the reorganization chain is allowed to move a box within its current quadtree cell, or it may move the box to the quadtree cell of an ancestor, but no other movements are allowed (see Fig. 7(c)).

More formally, consider a request for a box b to access point a . Let $Q_0(a)$ denote the quadtree cell containing a , and let $Q_1(a), Q_2(a), \dots$ denote the successive quadtree ancestor cells of $Q_0(a)$. If a is unoccupied, we simply move the box there. Otherwise, in order to make space for b_i , we perform a chain of swaps along some locations p_0, p_1, \dots, p_k such that $p_0 = a$, p_k is either unoccupied (possibly the former location of b), and if $p_i \in Q_j(a)$, then p_{i+1} is the same cell or an ancestor, that is, $p_{i+1} \in Q_{j'}(a)$ for $j' \geq j$. As described in Section 1.1, we perform swaps (in reverse order) along the resulting chain. Each swap that moves a box out of its current quadtree cell is called *eviction*.

Costs are defined as follows in this model. A box may be moved within its quadtree cell free of charge, but when it is moved to an ancestor cell, it is charged 2^k , where k is the level of the quadtree cell into which the box is moved. (The analogy with hierarchical memory systems should be evident, where we think of each quadtree cell as a cache, and eviction to an ancestor is analogous to moving a page to a larger cache in slower memory.)

3.2 Online Algorithm for Swapping Motion

Let us now present our algorithm for the warehouse problem, which we call Block-LRU_W . Consider a request (b, a) to bring box b to access point a . If this access point is unoccupied, we simply move the box there. Otherwise, in order to make space for b , we will perform a sequence of evictions from $Q_0(a)$, $Q_1(a)$, and so on until we encounter the first quadtree ancestor $Q_k(a)$ that has at least one unoccupied location (possibly b 's location at the time of the request). More formally, let p_b denote b 's location, let $p_0 = a$ denote the access point, and let p_1, \dots, p_{k-1} denote the locations of the least-recently used boxes of quadtree cells $Q_0(a)$ through $Q_{k-1}(a)$, respectively. Finally, let $p_k \in Q_k(a)$ denote the final unoccupied location (or former location of b). As described in Section 1.1, we perform swaps (in reverse order) along the chain $\langle p_b, p_0, \dots, p_k \rangle$. The main result of this section is showing that this algorithm is competitive.

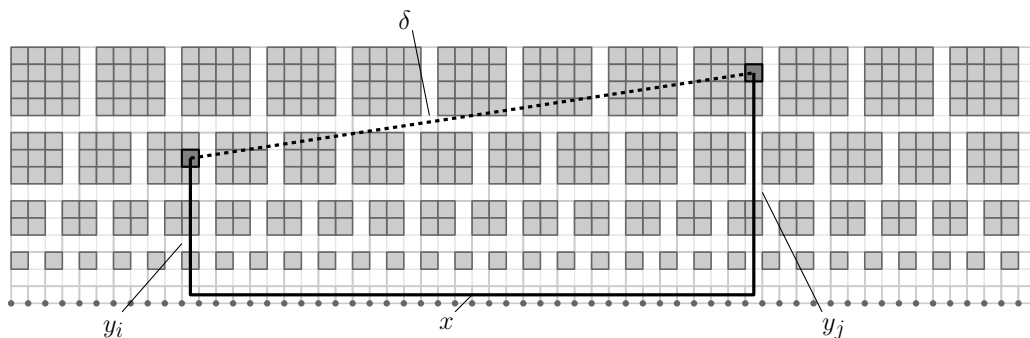
► **Theorem 10.** *For any instance of the warehouse problem and any sufficiently long access sequence S , the cost of $\text{Block-LRU}_W(S)$ is within a constant factor of the cost of an optimal solution, assuming swapping motion.*

Due to space limitations, the full proof and competitive analysis appear in Appendix A.2. It is a nontrivial extension of the single-container structure from the attic problem to a hierarchical container structure based on the quadtree, and showing how a general solution in the standard model can be transformed competitively into the quadtree model.

3.3 Online Algorithm for Sliding Motion

In this section, we show that the competitiveness of Block-LRU_W in the case of swapping motion can be used to prove that the sliding version of the same algorithm is competitive. As in the attic problem, our approach will be to describe a layout of boxes that is amenable to efficient sliding motion.

We make use of a Nicomachus-like box layout. Rather than rings centered about the access point, we flatten these rings into layers stacked above the x -axis. As before, we begin with a layer of 1×1 cell regions. Above this is a row of 2×2 regions, then 3×3 , and so on, with each $i \times i$ region containing a block of $(i - 1) \times (i - 1)$ boxes (see Fig. 8). We call this the *flattened Nicomachus layout*.



■ **Figure 8** A flattened version of the Nicomachus layout for the warehouse problem, with a conceptual example of swapping two boxes. Pathfinding is ignored in this illustration, but accounted for in the supporting lemma.

Once again, we make use of a simple naive algorithm that can efficiently trade the positions of two boxes in the sliding model. More formally, we prove the following:

► **Lemma 11.** *If two boxes b_i and b_j are at ℓ_1 distances δ from each other and at vertical distances y_i and y_j from the x -axis, respectively, then the cost of swapping them in the flattened Nicomachus layout is no more than $c(\delta + y_i + y_j)$, for some constant c .*

Proof. A naive algorithm can swap the two boxes b_i and b_j by: (1) bringing them to the x -axis, (2) swapping their positions along the x -axis, and (3) returning them to their new vertical positions. Notice that the cost of retrieving/replacing a box and bringing it to the x -axis is equivalent to the retrieval cost of a box positioned directly above the access point in the Attic Problem with Sliding Motion. As per Lemma 7, this access cost in both contexts is $O(y)$, where y is the distance to the x -axis or singular access point, respectively. Given this, both steps (1) and (3) of the algorithm occur at a constant factor of $(y_i + y_j)$. Clearly the horizontal distance traveled along the x -axis $x \leq \delta$, therefore, the total cost of swapping the two boxes must be no greater than $c(\delta + y_i + y_j)$, for some constant c . ◀

We can use this lemma to related the cost of swapping two elements in the swapping and sliding models. The following summarizes our main result.

► **Theorem 12.** *For any instance of the warehouse problem and any sufficiently long access sequence S , the cost of $\text{Block-LRU}_W(S)$ is within a constant factor of the cost of an optimal solution, assuming sliding motion.*

Proof. From Theorem 10 and the structure of Block-LRU_W , it suffices to bound the cost of evictions from one quadtree node to its parent. Assuming that the node is at quadtree level $k - 1$, and its parent is at level k , this swap incurs a cost of 2^k in the quadtree model. Letting y_1 and y_2 denote the vertical distances of these locations from the x -axis, we have $y_1 \leq 2^k$ and $y_2 \leq 2^{k+1}$. Also, they are separated from each other by an ℓ_1 distance of $\delta \leq 2^{k+2}$. By Lemma 11, the cost of sliding one to the other is at most $c(\delta + y_i + y_j) \leq c(2^{k+2} + 2^k + 2^{k+1}) = 7c2^k$, implying that sliding cost is within a constant factor of the quadtree cost. From the proof of Theorem 10 and the structure of Block-LRU_W , the quadtree cost of Block-LRU_W can be used as a proxy for its actual cost, and therefore the sliding cost is at most a constant factor more than the actual cost of Block-LRU_W assuming swapping motion. This implies that the cost of Block-LRU_W in the sliding motion model is competitive with the optimum solution in the swapping motion model. The actual cost of the optimum algorithm in the sliding model cannot be lower than the actual cost of the optimum algorithm in the swapping model. With a roughly factor-7 cost ratio between the sliding and swapping models, the overall ratio is roughly 112. As before, this is based on many worst-case assumptions and can likely be improved upon. ◀

4 Concluding Remarks

In this paper we have presented a model for an automated warehouse management system containing a set of standardized portable storage units or boxes, a robot that moves these boxes around the warehouse in one of two ways (swapping or sliding), and a set of access points where requested boxes must be delivered. We then presented online algorithms for two natural instances of the warehouse problem, one involving a single access point within a rectangular domain and the other involving a sequence of access points along the bottom side of a rectangular domain. We prove that our algorithms are competitive with respect to an optimal (offline) algorithm with full knowledge of the access sequence. Our competitive ratios are relatively high, and we suspect that they are far from tight, but tightening these bounds will involve either significantly more complex algorithms or better lower bounds.

We leave for future work some interesting open problems. Recall that our model assumes that access requests are processed sequentially. This simplifying assumption allowed us to ignore the extremely difficult issue of motion coordination, which arises when multiple robots are present [11, 10, 18]. Clearly, any realistic solution should consider an environment with multiple robots where requests are processed concurrently. Because we control the layout of boxes in the domain, it may be possible insert additional *slack space* into the layout to facilitate efficient motion coordination. Another interesting question in this vein is how to handle the insertion/deletion of boxes from the collection. Perhaps we could further leverage memory management schemes such as [9], which efficiently handle the reallocation of 2D memory.

Also, how does the competitiveness of our schemes change, if at all, when the model becomes less uniform. In our current model, all actions taken by the robot are of unit cost, regardless of factors like whether or not the robot is laden or what sort of path a robot takes to retrieve a box. Çelik and Süral [4], for example, show that the number of turns a robot makes in a parallel-aisle warehouse can have a significant impact on retrieval efficiency. Fekete and Hoffmann [8] look at the online problem of packing differently sized squares into

a dynamically sized square container, and applying this work to a warehouse which does not use standardized containers would be a natural continuation of the work presented here. Further generalizing our model to account for differing action costs and box dimensions would increase its real-world applicability and may lead to some interesting insights.

References

- 1 A. Aggarwal, B. Alpern, A. Chandra, and M. Snir. A Model for Hierarchical Memory. In *Proc. 19th Annu. ACM Sympos. Theory Comput.*, STOC '87, pages 305–314, New York, NY, 1987. ACM. doi:10.1145/28395.28428.
- 2 F. Amato, F. Basile, C. Carbone, and P. Chiacchio. An approach to control automated warehouse systems. *Control Eng. Pract.*, 13(10):1223–1241, October 2005. doi:10.1016/j.conengprac.2004.10.017.
- 3 A. S. Cahn. The summer meeting in Madison. *Bull. Amer. Math. Soc.*, 54(11):1073, November 1948. doi:10.1090/S0002-9904-1948-09093-0.
- 4 M. Çelik and H. Süral. Order picking in a parallel-aisle warehouse with turn penalties. *Internat. J. Production Res.*, 54(14):4340–4355, July 2016. doi:10.1080/00207543.2016.1154624.
- 5 F.-L. Chang, Z.-X. Liu, Z. Xin, and D.-D. Liu. Research on Order Picking Optimization Problem of Automated Warehouse. *Sys. Eng. - Theory & Pract.*, 27(2):139–143, February 2007. doi:10.1016/S1874-8651(08)60015-0.
- 6 A. Charnes and W. W. Cooper. Generalizations of the Warehousing Model. *OR: Oper. Research Quarterly*, 6(4):131–172, 1955. doi:10.2307/3006550.
- 7 J.-F. Cordeau and G. Laporte. The dial-a-ride problem: Models and algorithms. *Ann. Oper. Res.*, 153(1):29–46, 2007. doi:10.1007/s10479-007-0170-8.
- 8 S. P. Fekete and H.-F. Hoffmann. Online Square-into-Square Packing. *Algorithmica*, 77(3):867–901, 2017. doi:10.1007/s00453-016-0114-2.
- 9 S. P. Fekete., J.-M. Reinhardt, and C. Scheffer. An Efficient Data Structure for Dynamic Two-dimensional Reconfiguration. *J. Syst. Archit.*, 75(C):15–25, April 2017. doi:10.1016/j.sysarc.2017.02.004.
- 10 R. A. Hearn and E. D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theo. Comp. Sci.*, 343(1-2):72–96, 2005. doi:10.1016/j.tcs.2005.05.008.
- 11 J. E. Hopcroft, J. T. Schwartz, and M. Sharir. On the Complexity of Motion Planning for Multiple Independent Objects: PSPACE-Hardness of the “Warehouseman’s Problem”. *Internat. J. Robotics Res.*, 3(4):76–88, 1984. doi:10.1177/027836498400300405.
- 12 D. Jain. Adoption of next generation robotics: A case study on Amazon. *Perspectiva: A Case Research Journal*, III:15, 2017.
- 13 Elias Koutsoupias. The k-server problem. *Computer Science Review*, 3(2):105–118, 2009. doi:10.1016/j.cosrev.2009.04.002.
- 14 C. K. M. Lee. Development of an Industrial Internet of Things (IIoT) based Smart Robotic Warehouse Management System. In *CONF-IRM 2018 Proceedings*, page 14, 2018.
- 15 R. B. Nelsen. *Proofs without words: Exercises in visual thinking*. Number no. 1 in Classroom resource materials. The Mathematical Association of America, Washington, D.C, 1993.
- 16 K.-W. Pang and H.-L. Chang. Data mining-based algorithm for storage location assignment in a randomised warehouse. *Internat. J. Production Res.*, 55(14):4035–4052, July 2017. doi:10.1080/00207543.2016.1244615.
- 17 M. Sarrafzadeh and S. R. Maddila. Discrete warehouse problem. *Theo. Comp. Sci.*, 140(2):231–247, April 1995. doi:10.1016/0304-3975(94)00192-L.
- 18 R. Sharma and Y. Aloimonos. Coordinated motion planning: The warehouseman’s problem with constraints on free space. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(1):130–141, February 1992. doi:10.1109/21.141317.

- 19 D. D. Sleator and R. E. Tarjan. Amortized Efficiency of List Update and Paging Rules. *Commun. ACM*, 28(2):202–208, February 1985. doi:10.1145/2786.2793.
- 20 L. Wolsey and H. Yaman. Convex hull results for the warehouse problem. *Disc. Optimization*, 30:108–120, 2018. doi:10.1016/j.disopt.2018.06.002.

A Full Proofs

A.1 Competitiveness of Block-LRU_A (Attic Problem) with Swapping

► **Theorem 1.** *For any instance of the attic problem and any sufficiently long access sequence R , the cost of Block-LRU_A(S) is within a constant factor of the cost of an optimal solution, assuming swapping motion.*

Proof. Consider an input S consisting of the initial box placement and a sequence of access requests. Let $T_{\text{opt}}(S)$ and $T_{\text{lr}}(S)$ denote the total cost of the optimum and Block-LRU_A solutions, respectively, on this input. We will show that there exists a constant c and quantity $f(S)$ that does not grow with the length of the access sequence, such that $T_{\text{lr}}(S) \leq cT_{\text{opt}}(S) + f(S)$. Since $f(S)$ does not grow with the length of the access sequence, for all sufficiently long access sequences its impact on the total cost will be negligible compared to $T_{\text{opt}}(S)$.

Our analysis will be based on an auxiliary statistic. Given any container C_k , define an *eviction* to be an event in which a box lying within this container is moved to a location in an enclosing container $C_{k'}$, for $k' > k$. For the given access request sequence S , define $E_{\text{lr}}(S, k)$ to be the total number of evictions from container C_k performed by Block-LRU_A. Let $W_{\text{lr}}(S) = \sum_{k \geq 0} 2^k E_{\text{lr}}(S, k)$ denote the weighted cost of these evictions. We will show that there exist constants c_1 and c_2 and quantities $f_1(S)$ and $f_2(S)$ that do not grow with the length of the access sequence, such that the following two inequalities hold:

$$(1) T_{\text{lr}}(S) \leq c_1 W_{\text{lr}}(S) + f_1(S) \quad \text{and} \quad (2) W_{\text{lr}}(S) \leq c_2 T_{\text{opt}}(S) + f_2(S).$$

We first prove inequality (1). Observe that the cost of processing a request involving a box b in Block-LRU_A consists of two parts, the cost of moving b to the access point (that is, the ℓ_1 distance of b to access point) plus the cost of performing the evictions caused by this move. We assert that it suffices to bound only the latter quantity. To see why, consider two consecutive requests to b . Just after the first request, b is located at the access point. When the second request occurs, if b is not at the access point, it has been moved away due to various evictions involving b that have occurred due to intervening access requests. By the triangle inequality, the sum of the costs of these evictions involving b is at least as large as the ℓ_1 distance of b from the access point at the time of the second request. Thus, the cost of moving b to the access point for the second request is not greater than cost of evictions involving b due to intervening requests. This allows us to account for all the requests for b except the first. Define $f_1(S)$ to be the sum of the ℓ_1 of every box's initial location to the access point. Clearly, $f_1(S)$ depends only on the initial box placements.

It remains to bound the cost needed to process the evictions. Each time Block-LRU_A evicts a box from some container C_k to the enclosing container C_{k+1} , the cost is bounded above by the maximum distance between any point of C_k to any point in C_{k+1} . Clearly, this is not greater than the diameter of C_{k+1} , which is 2^{k+2} . Summing over all accesses and all containers, it follows that the total cost of Block-LRU_A evictions is at most $\sum_{k \geq 0} 2^{k+2} E_{\text{lr}}(S, k) = 4W_{\text{lr}}(S)$. By our earlier observation that the cost of bringing boxes back to the access point is bounded above by the sum of $f_1(S)$ and the total eviction cost, it follows that $T_{\text{lr}}(S) \leq c_1 W_{\text{lr}}(S) + f_1(S)$, where $c_1 = 2 \cdot 4 = 8$, thus establishing (1).

To prove inequality (2), we will apply a technique similar to one given by Sleator and Tarjan [19] and Aggarwal et al. [1] for hierarchical memory systems. For any $k \geq 0$, define $\overline{C}_k = \bigcup_{j \leq k} C_j$ (that is, the set of points within distance 2^k of the origin). Also define $m_k = |C_k|$ and $\overline{m}_k = |\overline{C}_k|$ denote the total capacities of these sets. For each $k \geq 2$, we will relate the weighted eviction cost of Block-LRU_A on container C_k with respect to the cost of box movements by the optimal solution within container C_k . The overall analysis comes about by summing over all container levels.

Fix any $k \geq 2$. Partition the access request sequence into contiguous *segments*, such that within any segment (except possibly the last), Block-LRU_A performs \overline{m}_k evictions from container C_k . (The last segment will not be analyzed, but since there is only one such segment for each k from which an eviction was performed, it follows that for all sufficiently long access segments, the impact on the overall cost of these segments be negligible. See [19] for more details.) Consider any complete segment. The contribution of the evictions of this segment from C_k to the weighted eviction cost $W_{\text{iru}}(S)$ is $2^k \overline{m}_k$. In Block-LRU_A every container C_j for $j \leq k$ evicts the least recently accessed box, and this implies that any box evicted from container C_k is the least recently accessed box not only from C_k , but from \overline{C}_k as well. We assert that during this segment, the number of distinct boxes accessed must be at least \overline{m}_k . To see why, observe that either all of the boxes evicted during this segment are distinct, or some box was evicted twice during the sequence. If there are \overline{m}_k distinct evictions, then there are at least \overline{m}_k distinct boxes requested. On the other hand, if a box is evicted twice, then by the nature of Block-LRU_A, between these two evictions, every one of the \overline{m}_k boxes in \overline{C}_k must have been accessed in order for this box to transition from the most recent to the least recent.

Now, let us consider how the optimum algorithm deals with the \overline{m}_k distinct box requests that have occurred during this segment. Intuitively, because of the exponential increase in container sizes, most of the \overline{m}_k distinct accessed boxes cannot fit within \overline{C}_{k-1} , and hence they must spill out into the surrounding region. We will charge for the work needed for the spillover but limited to C_k (to avoid double counting).

It will simplify matters to ignore boundary issues for now and consider the unbounded case where $\Omega = \mathbb{Z}^2$. Define \widehat{C}_k to be the set of points of the infinite grid that lie within ℓ_1 distance $(3/4)2^k$ of the access point. Since $k \geq 2$, we have $\overline{C}_{k-1} \subset \widehat{C}_k \subset \overline{C}_k$. Let $\widehat{m}_k = |\widehat{C}_k|$. We have $\widehat{m}_k \leq c' \overline{m}_k$, where $c' \approx (3/4)^2 \leq 2/3$. Thus, a fraction of $1 - c'$ or roughly one-third of the \overline{m}_k distinct boxes accessed during this sequence must spill out from \overline{C}_{k-1} to an ℓ_1 distance of at least $(3/4)2^k - 2^{k-1} = (1/2)2^{k-1} = 2^{k-2}$ beyond \overline{C}_{k-1} 's outer boundary. It follows that the contribution of to the cost of $T_{\text{opt}}(S)$ of these boxes is at least $(\overline{m}_k/3)2^{k-2} = 2^k \overline{m}_k/12$. Because all of these box motions are contained within C_k , there is no double counting of this cost between containers.

The generalization to the case of a bounded rectangular domain Ω is straightforward but tedious. The key difference is that, due to the bounded nature of Ω , the sizes of consecutive containers may grow only linearly, not quadratically with the ℓ_1 radius of the container. (This happens, for example, if the domain is a long, thin strip.) Further, the size of the last container may even be smaller than its predecessor as we approach the outer edges of the domain. However, the key is that, since the radius value grows exponentially, consecutive container sizes differ by a constant factor for all but a constant number containers, and this is all that the above analysis requires.

Let s_k denote the number of complete segments for level k . Summing all the segments and all the levels of the hierarchy, we obtain

$$T_{\text{opt}}(S) \geq \sum_{k \geq 2} s_k 2^{k-2} \overline{m}_k.$$

Adding in a term $f_2(S)$ to account for the final (incomplete) segments, noting that \bar{m}_0 and \bar{m}_1 are both constants, and combining with our earlier bound on $W_{\text{lr}}(S)$, we obtain the following, for a suitable constant c_3 .

$$\begin{aligned} W_{\text{lr}}(S) &\leq \sum_{k \geq 0} s_k 2^k \bar{m}_k + f_2(S) = s_0 \bar{m}_0 + s_1 2 \bar{m}_1 + \sum_{k \geq 2} s_k 2^k \bar{m}_k + f_2(S) \\ &\leq c_3(s_0 + s_1) + 4T_{\text{opt}}(S) + f_2(S). \end{aligned}$$

The term $c_3(s_0 + s_1)$ is just a constant times the total number of access requests and is not dominant. It follows that there is a constant c_2 such that $W_{\text{lr}}(S) \leq c_2 T_{\text{opt}}(S) + f_2(S)$, which establishes inequality (2). Note that $f_2(S)$ does not grow with the length of the access sequence.

Finally, by combining inequalities (1) and (2), we obtain

$$\begin{aligned} T_{\text{lr}}(S) &\leq c_1 W_{\text{lr}}(S) + f_1(S) \leq c_1(c_2 T_{\text{opt}}(S) + f_2(S)) + f_1(S) \\ &\leq c_1 c_2 T_{\text{opt}}(S) + (c_1 f_2(S) + f_1(S)) \leq c T_{\text{opt}}(S) + f(S), \end{aligned}$$

for some constant $c \geq c_1 c_2 \geq 32$ and quantity $f(S)$ that does not grow with the length of the access sequence. For all sufficiently long access sequences, this final term will be negligible. This completes the proof. \blacktriangleleft

A.2 Competitiveness of Block-LRU_W (Warehouse) with Swapping

► **Theorem 10.** *For any instance of the warehouse problem and any sufficiently long access sequence S , the cost of Block-LRU_W(S) is within a constant factor of the cost of an optimal solution, assuming swapping motion.*

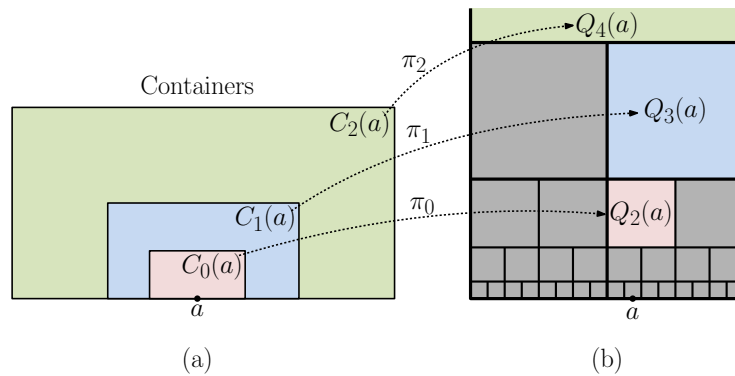
Observe that Block-LRU_W satisfies the requirements in quadtree model. For the sake of the above theorem, its cost is computed in the standard manner, as the sum of the ℓ_1 distances of all swaps performed. Later, we will show that this is proportional to its cost in the quadtree model.

The remainder of this section is devoted to proving this theorem. First, let us consider how we can simulate the behavior of a general solution to the warehouse problem in the quadtree model. Rather than focusing on individual access requests, we will do this on a box-by-box basis. Consider input sequence S and any box b . Let S' denote a contiguous segment of S , which starts and ends at two consecutive access requests involving b . Let us denote these access points by a_1 and a_2 , respectively. (For the segment prior to b 's first access, set a_1 the closest access point to b 's initial location, and for the segment following b 's last access, a_2 can be set arbitrarily to any access point.)

When the standard solution completes the processing of the first access request, b will reside at a_1 . As a result of subsequent access requests in S' , b may be moved to new locations in the domain as a result of swap operations. Let $\langle p_0, \dots, p_k \rangle$ denote the sequence of locations through which b moves during S' , so that $p_0 = a_1$, and p_k is the location of b just prior to the upcoming access request at a_2 . Since this is in the standard model, the points of this sequence are arbitrary. To perform the simulation, we will define a function π that maps the location of b at any time to the cell of some quadtree ancestor of a_1 in a manner such that, under this function, b will move in accordance with the quadtree model. We present this mapping in the next section.

A.2.1 Container Structure for the Warehouse Problem

Before giving the details of the aforementioned mapping, let us start with an intuitive explanation. For each access point a let $Q_k(a)$ denote the quadtree cell associated with a 's ancestor at level k . We define a collection of nested regions of exponentially increasing sizes called *containers* surrounding a , denoted $C_0(a) \subset C_1(a) \subset \dots$ (see Fig. 9(a)). (Note that, unlike the containers of Section 2.1, which were pairwise disjoint, here each container includes all the squares of its predecessors.)



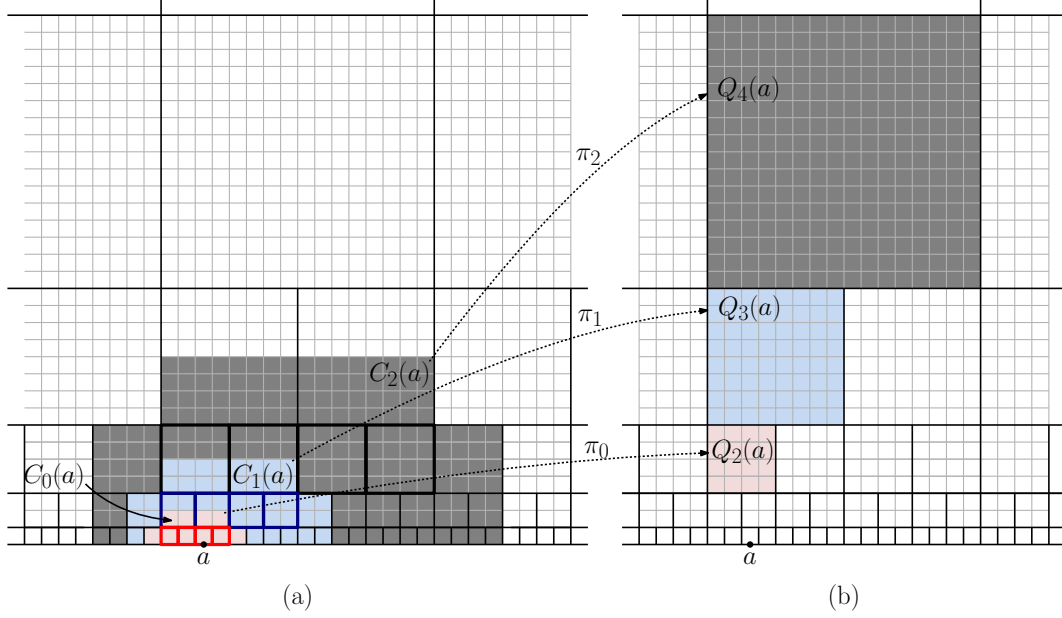
■ **Figure 9** Intuitive structure of containers for the warehouse quadtree model.

For each container $C_k(a)$ we will define a 1–1 function π_k that maps each of point in $C_k(a)$ to a point within the cell of some quadtree ancestor of a . (For example, in Fig. 9(a), π_k maps boxes from $C_k(a)$ to $Q_{k+2}(a)$.) In order to simulate the movement of a box that has been accessed most recently by a , we will track its movement through these containers. On first entering a container $C_k(a)$ at some point p , we map the box to the associated point $\pi_k(p)$ in the quadtree cell. When the box moves to a new point p' within the same container, we move the box to $\pi_k(p')$. Observe that because the containers are nested, even if the box moves into a location in a smaller container, it will still be considered as lying within C_k and so will remain in the same quadtree cell in the simulation. Recall that in the quadtree model, movements within the same quadtree cell are free of charge, and hence there is no need to account for movements within a given container. Whenever the box is first moved into a new larger container $C_{k'}$, it will be charged the eviction cost of $2^{k'}$, where $Q_{k'}(a)$ is the associated quadtree cell.

Let us now define the containers and the associated functions more formally. One complication that arises is that the functions π_k associated with two nearby access points may map locations to the same quadtree cell. When this happens, we must guarantee that two distinct locations in their containers are not mapped to the same location in this quadtree cell. To handle this, we will design our container structure carefully so that access points that map to the same quadtree cell will share the same container and the same mapping function.

To make this precise, consider any access point a and any quadtree ancestor of a at level k . The function π_k for a will map points from a 's container $C_k(a)$ to $Q_{k+2}(a)$. This implies that the four grandchildren of $Q_{k+2}(a)$ at level k will do the same. So, we will give them all a common container and a common function. (In Fig. 10(a), the container $C_2(a)$ is shared by four 4×4 quadtree cells drawn in heavy black lines.) The associated container is defined as follows. First, imagine a square grid of side length 2^k covering the plane that is aligned with the quadtree cells. The container consists of the 16 grid cells that are ℓ_1 neighbors of the four grandchildren. (In Fig. 10(a), this container $C_2(a)$ is shaded in dark gray and

includes the squares of $C_0(a)$ and $C_1(a)$. Note that the lowest tier of these grid squares falls one unit below the x -axis, but we simply ignore these nonexistent squares in our mapping.) The number of squares is at most $16 \cdot 2^k = 2^{k+2}$, and so there is sufficient space to map the squares of the container into $Q^{k+2}(a)$ (see Fig. 10(b)). We define π_k for this container to be any such function. (We do not require that this function preserve distances because, according to the quadtree model, movements within a quadtree cell are free.)



■ **Figure 10** Actual structure of containers for the warehouse quadtree model.

A.2.2 Proving Competitiveness

In this section, we present a proof of Theorem 10. Given an access sequence S , define $T_{\text{opt}}(S)$, $T_{\text{lru}}(S)$ to be the (standard) costs for Opt and Block-LRU_W, respectively. Define $W_{\text{lru}}(S)$ to be the cost of Block-LRU_W in the quadtree cost model, and define $W_{\text{opt}}(S)$ to be the cost of the quadtree-simulated version of Opt in the quadtree cost model.

The analysis follows a similar structure to the one given in Theorem 1, and so we will focus on just the major differences. The analysis is based on three inequalities, where c_1 , c_2 , and c_3 are constants and $f_2(S)$ and $f_3(S)$ are quantities that do not grow with the length of the access sequence:

$$(1) T_{\text{lru}}(S) \leq c_1 W_{\text{lru}}(S) \quad (2) W_{\text{lru}}(S) \leq c_2 W_{\text{opt}}(S) + f_2(S) \quad (3) W_{\text{opt}}(S) \leq c_3 T_{\text{opt}}(S) + f_3(S)$$

- $T_{\text{lru}}(S) \leq c_1 W_{\text{lru}}(S)$: Block-LRU_W is running in the quadtree model, but it uses the standard (ℓ_1) costs, not the eviction costs. Also, it evicts from child to parent, never skipping ancestors. When moving a box from quadtree cell Q_{k-1} to Q_k the actual cost is at most the worst-case ℓ_1 distance between these cells, which is at most $2 \cdot 2^k = 2^{k+1}$, and the quadtree model assesses a charge of 2^k . Thus, setting $c_1 = 2$ yields the desired bound.
- $W_{\text{lru}}(S) \leq c_2 W_{\text{opt}}(S) + f_2(S)$: Let $m_k = 2^{2k}$ denote the number of boxes in a quadtree cell Q_k at level k . Let \bar{m}_k the sum of m_j for a quadtree cell and all its descendants (which is roughly $2m_k$). Let us focus on a single quadtree cell at level k , call it Q_k . Consider the two child cells at level $k-1$, Q'_{k-1} and Q''_{k-1} . Let A' and A'' denote the subsets of

access points descended from these two quadtree nodes, respectively. Now, break up the access sequence into contiguous segments, such that Q_k witnesses \bar{m}_k evictions in the running of Block-LRU $_W$. Let us consider a single segment S' . Observe that, with respect to access points $A' \cup A''$, Block-LRU $_W$ is effectively running an LRU algorithm on the union of Q_k and the cells of all its children. (To see why, observe that the least-recently used boxes of each descendent are evicted to their parents and eventually up to Q_k , and the least-recently used box within Q_k is evicted.)

We assert that over segment S' , at least \bar{m}_k distinct box accesses have been processed by the access points $A' \cup A''$ combined. Now, let us consider how $W_{\text{opt}}(S)$ handles the same requests, but from the perspective of Q'_{k-1} and Q''_{k-1} . These two together (and their descendant cells) have a total capacity of $\bar{m}_{k-1} + \bar{m}_{k-1} \approx \bar{m}_k/2$. Thus, the remaining roughly $\bar{m}_k/2$ boxes must be evicted from these children by Opt. They may be evicted up one level to Q_k or up multiple levels. For the sake of simplicity, let us consider the case where they are evicted up just one level to Q_k . (The other case involves splitting the charge among the nodes along the path according to a geometric series.) Each evicted box is assessed a charge of 2^k , for a total of roughly $2^k \bar{m}_k/2 = 2^{k-1} \bar{m}_k$. Therefore, the total charge assessed to $W_{\text{opt}}(S)$ during this segment is at least $2^{k-1} \bar{m}_k$, while the total charge assessed to Q_k in $W_{\text{lr}}(S)$ is $2^{k+1} \bar{m}_k$. Summing over all the levels (and letting $f_2(S)$ account for the charges in the partial segment at the end of S) we have $W_{\text{lr}}(S) \leq c_2 W_{\text{opt}}(S) + f_2(S)$, where c_2 is roughly 4.

- $W_{\text{opt}}(S) \leq c_3 T_{\text{opt}}(S) + f_3(S)$: We focus on the activity involving a single box b between two consecutive accesses to a and a' , say. (The additional $f_3(S)$ term handles the cost prior to the initial request for b and after the final request.) Observe that $W_{\text{opt}}(S)$ does not charge for movements within a quadtree cell, and (since we are in the quadtree model) it never demotes a box to a lower level of the quadtree. It charges an eviction cost of 2^k whenever the box enters a quadtree cell at level k . This event corresponds to an event in standard Opt when this box enters $C_k(a) \setminus C_{k-1}(a)$ for the first time. Let k^* denote the highest container index into which Opt moves this box (formally, the highest k such that the box enters $C_k(a) \setminus C_{k-1}(a)$). Since this box might be evicted into all the containers from level 1 up to k^* , this box contributes at most $\sum_{k=1}^{k^*} 2^k \leq 2^{k^*+1}$ to $W_{\text{opt}}(S)$. On the other hand, Opt has to move this box from the access point to some point in $C_{k^*}(a) \setminus C_{k^*-1}(a)$. It is easy to see that this involves a distance of at least $2^{k^*} + 1$. It follows that this box contributes more than 2^{k^*} to $T_{\text{opt}}(S)$ and at most 2^{k^*+1} to $W_{\text{opt}}(S)$. Therefore, setting $c_3 = 2$ yields the desired result.

Together, the three inequalities imply that

$$\begin{aligned} T_{\text{lr}}(S) &\leq c_1 W_{\text{lr}}(S) \leq c_1 (c_2 W_{\text{opt}}(S) + f_2(S)) \\ &\leq c_1 (c_2 (c_3 T_{\text{opt}}(S) + f_3(S)) + f_2(S)) \leq c T_{\text{opt}}(S) + f(S), \end{aligned}$$

where $c = c_1 c_2 c_3 = 16$ and $f(S) = c_1 c_2 f_3(S) + c_1 f_2(S)$. This completes the proof of Theorem 10.

On Approximate Range Mode and Range Selection

Hicham El-Zein

Cheriton School of Computer Science, University of Waterloo, Canada
helzein@uwaterloo.ca

Meng He

Faculty of Computer Science, Dalhousie University, Canada
mhe@cs.dal.ca

J. Ian Munro

Cheriton School of Computer Science, University of Waterloo, Canada
imunro@uwaterloo.ca

Yakov Nekrich

Department of Computer Science, Michigan Technological University, USA
yakov@mtu.edu

Bryce Sandlund

Cheriton School of Computer Science, University of Waterloo, Canada
bcsandlund@uwaterloo.ca

Abstract

For any $\varepsilon \in (0, 1)$, a $(1 + \varepsilon)$ -approximate range mode query asks for the position of an element whose frequency in the query range is at most a factor $(1 + \varepsilon)$ smaller than the true mode. For this problem, we design a data structure occupying $O(n/\varepsilon)$ bits of space to answer queries in $O(\lg(1/\varepsilon))$ time. This is an encoding data structure which does not require access to the input sequence; the space cost of this structure is asymptotically optimal for constant ε as we also prove a matching lower bound. Furthermore, our solution improves the previous best result of Greve et al. (Cell Probe Lower Bounds and Approximations for Range Mode, ICALP'10) by saving the space cost by a factor of $\lg n$ while achieving the same query time. In dynamic settings, we design an $O(n)$ -word data structure that answers queries in $O(\lg n / \lg \lg n)$ time and supports insertions and deletions in $O(\lg n)$ time, for any constant $\varepsilon \in (0, 1)$; the bounds for non-constant $\varepsilon = o(1)$ are also given in the paper. This is the first result on dynamic approximate range mode; it can also be used to obtain the first static data structure for approximate 3-sided range mode queries in two dimensions.

Another problem we consider is approximate range selection. For any $\alpha \in (0, 1/2)$, an α -approximate range selection query asks for the position of an element whose rank in the query range is in $[k - \alpha s, k + \alpha s]$, where k is a rank given by the query and s is the size of the query range. When α is a constant, we design an $O(n)$ -bit encoding data structure that can answer queries in constant time and prove this space cost is asymptotically optimal. The previous best result by Krizanc et al. (Range Mode and Range Median Queries on Lists and Trees, Nordic Journal of Computing, 2005) uses $O(n \lg n)$ bits, or $O(n)$ words, to achieve constant approximation for range median only. Thus we not only improve the space cost, but also provide support for any arbitrary k given at query time. We also analyse our solutions for non-constant α .

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases data structures, approximate range query, range mode, range median

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.57



© Hicham El-Zein, Meng He, J. Ian Munro, Yakov Nekrich, and Bryce Sandlund;
licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 57; pp. 57:1–57:14

Leibniz International Proceedings in Informatics



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

1 Introduction

The mode and median of a data set are important statistics, widely used across many disciplines. Thus, they are frequently computed in applications for data mining, information retrieval and data analytics. The range mode and median problems further aim at speeding up the computation of the mode and median in an arbitrary subrange of the given sequence of elements, and thus have been studied extensively [17, 1, 19, 20, 12, 13, 16, 2, 15, 4, 11, 5, 14, 7]. In these problems, we preprocess a sequence of elements c_1, c_2, \dots, c_n to answer queries. Given two indices a and b with $1 \leq a \leq b \leq n$, a *range mode query* asks for a position of the most frequent element in $c_{a..b}$ ($c_{a..b}$ denotes c_a, \dots, c_b), while a *range median query* asks for the position of the median element in $c_{a..b}$. A generalization of range median is the *range selection query*, which asks for the position of the k^{th} smallest element in $c_{a..b}$ for any given k . Thus a range selection query becomes range median if $k = \lceil (b - a + 1)/2 \rceil$.

Due to the massive amounts of electronic data available, linear space data structures are often preferred by modern applications. The following are the best solutions to these query problems that use $O(n)$ words of space. In static settings, Chan et al. [4] showed how to answer a range mode query in $O(\sqrt{n/\lg n})$ time. By proving a conditional lower bound, they also gave strong evidence that, if linear space is required, this query time cannot be improved significantly using purely combinatorial methods with current knowledge. When updates to elements are allowed, El-Zein et al. [7] showed how to support both range mode queries and updates in $O(n^{2/3})$ time. For range selection, the solution of Chan and Wilkinson [5] answers queries in $O(\lg k / \lg \lg n + 1)$ time, matching the lower bound of Jørgensen and Larsen [16] under the cell probe model. He et al. [15] showed how to support range selection in $O((\lg n / \lg \lg n)^2)$ worst-case time and updates in $O((\lg n / \lg \lg n)^2)$ amortized time.

The query times for range mode in both linear space data structure solutions and conditional lower bounds are much larger than that for many other query problems, including range median. To provide faster support for queries, researchers have studied *approximate range mode* [13]. To define this query, let $F_x(c_{a..b})$ denote the frequency of an element x in $c_{a..b}$ and $F(c_{a..b})$ denote the frequency of the mode of $c_{a..b}$ ($F(c_{a..b}) = \max_x F_x(c_{a..b})$). Then a $(1 + \varepsilon)$ -approximate range mode query asks for the position of an element x in $c_{a..b}$ such that $(1 + \varepsilon) \cdot F_x(c_{a..b}) \geq F(c_{a..b})$ for some positive ε . This element is called a $(1 + \varepsilon)$ -approximate mode of $c_{a..b}$. Previously, the best result on this problem is that of Greve et al. [13], which uses $O(n/\varepsilon)$ words of space to support queries in $O(\lg(1/\varepsilon))$ time, for any $\varepsilon \in (0, 1)$.

Approximate range median can be defined similarly. We say that the i th smallest element in the query range $c_{a..b}$ has rank i . Then, for an approximation ratio $\alpha \in (0, 1/2)$, an α -approximate range median query asks for the position of an element x whose rank in $c_{a..b}$ is between $\lceil s/2 \rceil - \alpha s$ and $\lceil s/2 \rceil + \alpha s$, where $s = b - a + 1$. Bose et al. [1] studied this problem, for which they proposed a data structure occupying $O(n/\alpha)$ words of space that answers queries in constant time. An α -approximate range selection query can also be defined, which, for any given k , asks for the position of an element x whose rank in $c_{a..b}$ is between $k - \alpha s$ and $k + \alpha s$. However, this problem has not been formally studied previously.

To further improve the space efficiency of data structures, researchers have recently studied various query problems in the encoding model [8, 14]. Under this model, a data structure is not allowed to store or assume access to the original data set. Instead, it should occupy as little space as possible while providing support for queries. For example, in this model, Fischer and Heun [8] studied the range minimum query problem, which asks for the position of the smallest element in $c_{a..b}$. They proposed a data structure occupying only $2n + o(n)$ bits with constant query time. The range selection problem has also been

■ **Table 1** Static and Dynamic Range Mode Query History. In this table, δ is an arbitrary constant in $(0, 1/2)$ and $m = \min(n \lg n/\varepsilon, n/\varepsilon^2)$.

Query Type	Query Time	Update Time	Space in Bits	Source
Exact	$O(n^\delta \lg n)$	-	$O(n^{2-2\delta} \lg n)$	[17]
	$O(1)$	-	$O(n^2 \log \log n / \lg n)$	[20]
	$O(\sqrt{n/\log n})$	-	$O(n \lg n)$	[4]
	$O(n^{3/4} \log n / \log \log n)$	$O(n^{3/4} \log \log n)$	$O(n \lg n)$	[4]
	$O(n^{2/3} \log n / \log \log n)$	$O(n^{2/3} \log n / \log \log n)$	$O(n^{4/3} \lg n)$	[4]
	$O(n^{2/3})$	$O(n^{2/3})$	$O(n \lg n)$	[7]
$(1 + \varepsilon)$ - Approximation	$O(\lg \lg n + \lg(1/\varepsilon))$	-	$O(n \lg n / \varepsilon)$	[1]
	$O(\lg(1/\varepsilon))$	-	$O(n \lg n / \varepsilon)$	[13]
	$O(\lg(1/\varepsilon))$	-	$O(n/\varepsilon)$	new
	$O(\lg m / \lg \lg m)$	$O(\lg n / \varepsilon^2)$	$O(m \lg m)$	new

considered in this model: Grossi et al. [14] proposed an encoding data structure occupying $O(n \lg \kappa)$ bits for any fixed positive integer κ , using which a range selection query can be answered in $O(\lg k / \lg \lg n + 1)$ time for any k given in the query with $1 \leq k \leq \kappa$.

Naturally, encoding data structures are only relevant when their space occupancy is asymptotically less than the input data, at least for certain choices of parameters. The space costs of previous results on approximate range mode or median, however, match the size of the input sequence asymptotically when ε or α is a constant and become superlinear when ε or α is in $o(1)$. Thus, we study the problem of designing encoding data structures of approximate range mode, median and selection queries, to improve the space efficiency of previous solutions. Furthermore, previously no research has been done on dynamic approximate range mode, while the dynamic exact data structures for range mode require polynomial query and update times. Therefore, we also study approximate range mode queries under dynamic settings, to provide substantially faster support for queries and updates.

Our Results. For $(1 + \varepsilon)$ -approximate range mode, where $0 < \varepsilon < 1$, we design an encoding data structure using $O(n/\varepsilon)$ bits that can answer a query in $O(\lg(1/\varepsilon))$ time. This is an improvement upon the previous best result of Greve et al. [13], since we match their query time while saving the space cost by a factor of $\lg n$; we assume a word RAM model in which each word has $\Theta(\lg n)$ bits. We also prove a lower bound to show that any data structure supporting $(1 + \varepsilon)$ -approximate range mode must use $\Omega(n/(1 + \varepsilon))$ bits for any positive ε . This means that our space cost is asymptotically optimal for constant ε . When ε is not necessarily a constant, as long as $\varepsilon = \omega(1/\lg n)$, our data structure uses $o(n \lg n)$ bits, i.e., $o(n)$ words, which is asymptotically less than the space needed to encode the original sequence itself.

For α -approximate range selection, where $0 < \alpha < 1/2$, we design encoding data structures for two variants of this problem. If k is fixed and given in advance, either as a constant or as a function of the size, s , of the query range satisfying certain reasonable constraints (e.g., $k = \lceil s/2 \rceil$ for range median), we have a solution occupying $O(n/\alpha^2)$ bits that can answer a query in constant time. If k is not known beforehand and different values of k could be given with each query, we have another encoding structure in $O(n/\alpha^3)$ bits with constant query time. Our query time matches that of the previous best data structure of Bose et al. [1] which supports range median only, while we decrease the space cost by a factor of $\lg n$ when α is a constant. As we also show that any approximate range selection data structure must use at least $\Omega(n)$ bits, our data structures are asymptotically optimal for constant α .

■ **Table 2** Static and Dynamic Range Median and Selection Query History.

Query Type	Query Time	Update Time	Space in Bits	Source
Exact	$O(1)$	-	$O((n \lg \lg n)^2 / \lg n)$	[20]
	$O(\lg n / \lg \lg n)$	-	$O(n \log n)$	[2]
	$O(\lg k / \lg \lg n + 1)$	-	$O(n \lg n)$	[5]
	$O(\lg^2 n)$	$O(\lg^2 n)$	$O(n \lg^2 n)$	[12]
	$O((\lg n / \lg \lg n)^2)$	$O((\lg n / \lg \lg n)^2)$	$O(n \lg^2 n / \lg \lg n)$	[2]
	$O((\lg n / \lg \lg n)^2)$	$O((\lg n / \lg \lg n)^2)$	$O(n \lg n)$	[15]
α - Approximation	$O(1)$	-	$O(n \lg n / \alpha)$	[1]
(with fixed k)	$O(1)$	-	$O(n / \alpha^2)$	new
α - Approximation	$O(1)$	-	$O(n / \alpha^3)$	new

In dynamic settings, for any $\varepsilon \in (0, 1)$, we present an $O(m \lg m)$ -bit structure where $m = \min(n \lg n / \varepsilon, n / \varepsilon^2)$. It supports $(1 + \varepsilon)$ -approximate range mode in $O(\lg m / \lg \lg m)$ time and insertions/deletions in $O(\lg n / \varepsilon^2)$ time. When ε is an arbitrary constant in $(0, 1)$, this data structure uses $O(n)$ words, answers queries in $O(\lg n / \lg \lg n)$ time, and supports updates in $O(\lg n)$ time. As the best result on dynamic exact range mode [7] requires $O(n^{2/3})$ time for both queries and updates, this approximate solution is much faster for constant ε . It is also the first result on dynamic approximate range mode. Finally, we apply the technique to solve static $(1 + \varepsilon)$ -approximate three-sided range mode in two dimensions, achieving $O(\lg m)$ time query and occupying $O(m \lg m)$ words of space, where again $m = \min(n \lg n / \varepsilon, n / \varepsilon^2)$. This is another new approximate query problem.

Tables 1 and 2 compare our results to previous work to be surveyed in Section 2.

2 Previous Work

Range Mode. Krizanc et al. [17] first studied the static range mode problem and showed that, for any $\delta \in (0, 1/2)$, there is an $O(n^{2-2\delta})$ -word solution that answers queries in $O(n^\delta \log n)$ time. Setting $\delta = 1/2$ yields an $O(n)$ -word data structure supporting range mode in $O(\sqrt{n} \log n)$ time. They also presented a data structure using $O(n^2 \log \log n / \log n)$ words, or $O(n^2 \log \log n)$ bits, to support queries in constant time. Chan et al. [4] further provided a better linear word solution with $O(\sqrt{n / \log n})$ query time. They also proved a conditional lower bound to show that, with current knowledge, either the query time must be polynomial, or the construction time must be polynomially larger than n . Later, Greve et al. [13] gave an (unconditional) lower bound in the cell probe model, showing that any structure using S memory cells of w -bit words requires $\Omega(\frac{\log n}{\log(Sw/n)})$ time to answer a range mode query. On the other end of the spectrum, there has been work [19, 20] on improving the constant-time query structure of Krizanc et al., and the best solution uses $O(n^2 \lg \lg n / \lg^2 n)$ words, or $O(n^2 \lg \lg n / \lg n)$ bits [20].

In dynamic settings, Chan et al. [4] provided a tradeoff among space cost, query time and update time. This tradeoff implies two important results: using linear space in words, range mode can be supported in $O(n^{3/4} \log n / \log \log n)$ worst-case time while updates can be performed in $O(n^{3/4} \log \log n)$ amortized expected time. Alternatively, they can use $O(n^{4/3})$ words to improve the query and update efficiency to $O(n^{2/3} \log n / \log \log n)$ worst-case time and amortized expected time, respectively. They also proved a conditional lower bound to show that, with current knowledge, either queries or updates must require polynomial time. Very recently, El-Zein et al. [7] further improved these solutions by designing an $O(n)$ -word structure supporting both queries and updates in $O(n^{2/3})$ time.

Bose et al. [1] were the first to study approximate range mode. They showed how to provide constant-time support for 4-approximate mode, 3-approximate mode and 2-approximate mode using data structures occupying $O(n)$, $O(n \lg \lg n)$ and $O(n \lg n)$ words, respectively. For $(1 + \varepsilon)$ -approximation, they designed an $O(n/\varepsilon)$ -word solution that can answer a query in $O(\lg \lg_{1+\varepsilon} n) = O(\lg \lg n + \lg(1/\varepsilon))$ time. Greve et al. [13] further improved these results by using $O(n/\varepsilon)$ words of space to support queries in $O(\lg(1/\varepsilon))$ time.

Range Median and Selection. The study of range median also has a rich history. It was also Krizanc et al. [17] who initially proposed this problem. There have been several solutions with near-quadratic space and constant query time [17, 19, 20], the best of which uses $O((n \lg \lg n / \lg n)^2)$ words [20]. For linear-space solutions, following a series of earlier work [17, 10, 12, 3], Brodal et al. [2] first achieved an $O(n)$ -word solution that answers range median and selection queries in $O(\lg n / \lg \lg n)$ time. Jørgensen and Larsen [16] further improved the query time of range selection to $O(\lg \lg n + \lg k / \lg \lg n)$, where k is the specified query rank. They also proved that, under the cell probe model, $\Omega(\lg k / \lg \lg n + 1)$ time is necessary for any range selection data structure using $O(n \lg^{O(1)} n)$ space. Chan and Wilkinson [5] were then the first who designed a linear word solution with $O(\lg k / \lg \lg n + 1)$ optimal query time for range selection. More recently, Grossi et al. [14] proposed an encoding data structure occupying $O(n \lg \kappa)$ bits for any fixed positive integer κ , using which a range selection query can be answered in $O(\lg k / \lg \lg n + 1)$ time for any k given in the query with $1 \leq k \leq \kappa$. Gawrychowski and Nicholson [11] presented a space-optimal encoding of range selection which uses even less space, and proved its space cost is optimal within an $o(n)$ additive term in bits, though no support for queries is provided. All of the above results for range selection assume the selection rank k is specified at query time.

In the dynamic case, Gfeller and Sanders [12] proposed a data structure that uses $O(n \lg n)$ words of space to support range median in $O(\lg^2 n)$ time and insertions and deletions in $O(\lg^2 n)$ amortized time. The structure of Brodal et al. [2] occupies $O(n \lg n / \lg \lg n)$ words of space, answers queries in $O((\lg n / \lg \lg n)^2)$ worst-case time and supports insertions and deletions in $O((\lg n / \lg \lg n)^2)$ amortized time. Later He et al. [15] improved the space cost to $O(n)$ words while providing the same support for queries and updates. The work of Bose et al. [1] is the only work on α -approximate range median, for which they proposed a data structure occupying $O(n/\alpha)$ words of space that answers queries in constant time.

3 Approximate Range Mode

Before we proceed, we give a few preliminaries. We will at times refer to elements (of $c_{1..n}$ or otherwise) as *colors*. This is because their data type has no significance in frequency applications and thus the term color standardizes the data type. Furthermore, at times we create indexing such as a value r_i for when the mode in some range $c_{s_i..r_i}$ exceeds a given threshold. It is possible the mode never exceeds such a threshold. To avoid dealing with such corner cases in the rest of this exposition, we make the assumption that our list of elements $c_{1..n}$ is padded at the beginning and end with a sufficient number of one arbitrary color.

We allow non-constant ε . However, in our upper bounds, we make the restriction $\varepsilon \leq 1$, to allow simplification in the runtime and space analyses.

► **Theorem 1.** *Any one-dimensional $(1 + \varepsilon)$ -approximate range mode data structure requires $\Omega(n/(1 + \varepsilon))$ bits.*

Proof. Using a simple proof we show that $\Omega(n/(1 + \varepsilon))$ bits are required for any data structure that answers one-dimensional approximate range mode queries. Here we allow arbitrary ε .

Given an approximation factor $1 + \varepsilon$, divide the sequence S of size n into $\lfloor n/(2k) \rfloor$ full blocks each of size $2k$, where $k = \lceil 1 + \varepsilon \rceil + 1$, and, if n is not a multiple of $2k$, a non-full block of size $n \bmod 2k$. Denote by t_1, \dots, t_{k+1} $k + 1$ arbitrary, distinct colors. We say that S satisfies property $(*)$ if for each full block b in S one of the following two conditions hold:

- either b consists of t_1 repeated k times followed by t_2, \dots, t_{k+1} ,
- or b consists of t_2, \dots, t_{k+1} followed by t_1 repeated k times.

Clearly, the number of sequences that satisfy $(*)$ is at least $2^{\lfloor n/(2k) \rfloor}$, since there exist $\lfloor n/(2k) \rfloor$ full blocks in a sequence of size n and each of them can have one of two different values. Moreover, for any two distinct sequences S_1 and S_2 satisfying $(*)$ differing at full block b , there exists at least one approximate range mode query, namely the query that asks for an approximate mode of b , that will return different values (either a value from the first k position in the block or from the last k positions of the block). Thus, the information theoretic lower bound for storing an approximate range mode data structure is $\Omega(\lg 2^{\lfloor n/(2k) \rfloor}) = \Omega(\lfloor n/(2k) \rfloor) = \Omega(n)$ bits. ◀

We now proceed with our new upper bound. Our data structure consists of two parts. The first part answers *low* frequency queries $c_{a..b}$ with $F(c_{a..b}) \leq \lceil 1/\varepsilon \rceil$, and is exact. The second part answers *high* frequency queries $c_{a..b}$ with $F(c_{a..b}) > \lceil 1/\varepsilon \rceil$, and makes use of the approximation factor.

Low Frequencies: $O(n/\varepsilon)$ -Bits $O(\lg(1/\varepsilon))$ Query Time. Similar to the data structure of Greve et al. [13], for $k = 0, \dots, \lceil 1/\varepsilon \rceil$ let Q_k be an increasing sequence of size n such that $Q_k[i]$ is the largest integer $j \geq i$ satisfying $F(c_{i..j}) = k$. Since Q_k is an increasing sequence whose largest element is n , we store it in $2n + O(n/\lg^2 n)$ bits [18] while still accessing its i^{th} element in constant time¹. The total space used is $O(n/\varepsilon)$ bits. Given a query range $c_{a..b}$, $F(c_{a..b}) > k$ iff $b > Q_k[a]$. Thus, using binary search, we can determine if $F(c_{a..b}) < 1/\varepsilon$ and $K = F(c_{a..b})$ in that case. If $F(c_{a..b}) < 1/\varepsilon$ we return index $Q_{(K-1)}[a] + 1$; otherwise we query the high frequency structure. The total time is $O(\lg(1/\varepsilon))$.

High Frequencies: $O(n/\varepsilon)$ -Bits $O(\lg \lg n + \lg(1/\varepsilon))$ Query Time. We first present an $O(n/\varepsilon)$ -bit structure that answers high frequency $(1 + \varepsilon)$ -approximate range mode queries in $O(\lg \lg n + \lg(1/\varepsilon))$ time. We start by developing a tool to binary search the frequency of the mode, with the goal of locating a $(1 + \varepsilon)$ -approximate mode.

► **Lemma 2.** *There exists a data structure using $O(k \cdot \varepsilon \cdot n / (1 + \varepsilon)^k + n / \lg^2 n)$ bits that can find in constant time, for any query range $c_{a..b}$, one of the following that holds:*

1. $F(c_{a..b}) < (1 + \varepsilon)^k / \varepsilon$,
2. $F(c_{a..b}) > (1 + \varepsilon)^k / \varepsilon$, or
3. $((1 + \varepsilon)^{k-1/2}) / \varepsilon < F(c_{a..b}) < ((1 + \varepsilon)^{k+1/2}) / \varepsilon$.

In case 2, we find an element with frequency greater than $(1 + \varepsilon)^k / \varepsilon$ in range $c_{a..b}$. In case 3, we find an element with frequency greater than $((1 + \varepsilon)^{k-1/2}) / \varepsilon$ in range $c_{a..b}$.

When this structure is present for all k in range $0, \dots, \lceil \lg_{1+\varepsilon} n \rceil$, the above trichotomy is sufficient to binary search for an approximate mode of frequency at least $1/\varepsilon$. If we ever land in case 3, the encoding gives an approximate mode, and otherwise, we find the k satisfying

¹ We store $Q_k[1]$ and $(Q_k[i] - Q_k[i-1])$ in unary with a 0 separator between each two consecutive values in a $2n$ -bit vector ψ with rank and select structures. To access $Q_k[i]$ we count the number of 1s before the i^{th} 0 in ψ .

$(1 + \varepsilon)^k / \varepsilon < F(c_{a..b}) < (1 + \varepsilon)^{k+1} / \varepsilon$, which represents case 2 for value k and case 1 for value $k + 1$. Since case 2 provides an element with frequency greater than $(1 + \varepsilon)^k / \varepsilon$, this element is an approximate mode.

Proof. Let $1 + \Delta = \sqrt{1 + \varepsilon}$ and $f_j = (\Delta / \varepsilon) \cdot (1 + \Delta)^j$. For each integer i in $[0, n / \lceil f_{2k-1} \rceil]$ let $s_i = i \cdot \lceil f_{2k-1} \rceil + 1$ and denote by r_i the smallest value such that $F(c_{s_i..r_i}) \geq (1 + \Delta)^{2k} / \varepsilon$. Notice that c_{r_i} is the unique mode of $c_{s_i..r_i}$. Similarly, for each integer i in $[0, n / \lceil f_{2k} \rceil]$, let $s'_i = i \cdot \lceil f_{2k} \rceil + 1$ and denote by r'_i the smallest value such that $F(c_{s'_i..r'_i}) \geq (1 + \Delta)^{2k+1} / \varepsilon$.

Given a query range $c_{a..b}$, we find the biggest indices s_i, s'_j preceding or equal to a . We proceed as follows.

1. If $b < r_i$, then $F(c_{a..b}) \leq F(c_{s_i..r_i-1}) < ((1 + \Delta)^{2k} / \varepsilon) = ((1 + \varepsilon)^k / \varepsilon)$.
2. If $b \geq r'_j$, then $F_{r'_j}(c_{a..b}) > F_{r'_j}(c_{s'_j..r'_j}) - f_{2k}$, since there are at most $\lceil f_{2k} \rceil - 1 < f_{2k}$ elements between s'_j and a . Then:

$$\begin{aligned} F_{r'_j}(c_{a..b}) &> F_{r'_j}(c_{s'_j..r'_j}) - f_{2k} \geq ((1 + \Delta)^{2k+1} / \varepsilon) - (\Delta / \varepsilon) \cdot (1 + \Delta)^{2k} = (1 + \Delta)^{2k} / \varepsilon \\ &= (1 + \varepsilon)^k / \varepsilon. \end{aligned}$$

3. Suppose $b \geq r_i$ and $b < r'_j$. Since there are at most $\lceil f_{2k-1} \rceil - 1 < f_{2k-1}$ elements between s_i and a and since $b \geq r_i$, we have that

$$\begin{aligned} F_{r_i}(c_{a..b}) &> F_{r_i}(c_{s_i..r_i}) - f_{2k-1} \geq ((1 + \Delta)^{2k} / \varepsilon) - (\Delta / \varepsilon) \cdot (1 + \Delta)^{2k-1} = ((1 + \Delta)^{2k-1} / \varepsilon) \\ &= ((1 + \varepsilon)^{k-1/2} / \varepsilon). \end{aligned}$$

Finally, since $b < r'_j$, then $F(c_{a..b}) \leq F(c_{s'_j..r'_j-1}) < ((1 + \Delta)^{2k+1} / \varepsilon) = ((1 + \varepsilon)^{k+1/2} / \varepsilon)$.

To store the values $\{r_i\}$, we construct a bit vector of length $O(n)$ as follows. In the bit vector, there are n 0s. For each r_i , we insert a 1 bit after the r_i th 0 bit in the bit vector. Thus r_i is equal to the number of 0s before the i th 1 bit in the bit vector. A second bit vector of length $O(n)$ is used to encode the values $\{r'_i\}$ in a similar way. We then represent these two bit vectors in the succinct data structure of Patrascu [18]. This data structure provides constant time rank and select, which allow us to locate r_i and r'_j , and thus determine whether case 1, 2, or 3 applies, in constant time.

For a bit vector of size n with m ones, the space cost can be made $O(m \lg(n/m) + n / \lg^2 n)$ bits [18]. For vector r , $m \lg(n/m) = O((n / f_{2k-1}) \lg f_{2k-1})$, and for vector r' , $m \lg(n/m) = O((n / f_{2k}) \lg f_{2k})$. The cost is dominated by vector r . Let us first consider the $O(m \lg(n/m))$ term. We have

$$n / f_{2k-1} \lg f_{2k-1} = \frac{\varepsilon n}{\Delta (1 + \Delta)^{2k-1}} \lg((\Delta / \varepsilon) \cdot (1 + \Delta)^{2k-1}). \quad (1)$$

Rationalizing the denominator, we can show $\frac{1}{\Delta} = \frac{1}{\sqrt{1+\varepsilon}-1} = \frac{1+\sqrt{1+\varepsilon}}{\varepsilon}$ and so $\frac{1}{\Delta} = \Theta(\frac{1}{\varepsilon})$ and $\Delta / \varepsilon = O(1)$. Thus, with $\varepsilon \leq 1$, we can bound (1) with $O\left(\frac{(k-1/2)n}{(1+\varepsilon)^{k-1/2}} \lg(1+\varepsilon)\right)$. Finally, since we restrict $\varepsilon \leq 1$, we can do a Taylor series expansion to give $\lg(1+\varepsilon) = O(\varepsilon)$. Thus our final space bound is $O((n / f_{2k-1}) \lg f_{2k-1} + n / \lg^2 n) = O(k \cdot \varepsilon \cdot n / (1 + \varepsilon)^k + n / \lg^2 n)$. ◀

To make the above lemma useful, we must apply it to all k in range $0, \dots, \lceil \lg_{1+\varepsilon} \varepsilon n \rceil$. We first analyze the total space cost of all the $O(k \cdot \varepsilon \cdot n / (1 + \varepsilon)^k)$ terms. Summing up these terms, we have $O\left(\sum_{k=1}^{\lceil \lg_{1+\varepsilon} \varepsilon n \rceil} (k \cdot \varepsilon \cdot n / (1 + \varepsilon)^k)\right) = O\left(n \cdot \varepsilon \sum_{k=1}^{\infty} (k / (1 + \varepsilon)^k)\right) = O(n / \varepsilon)$ bits. The other term comes out to $O(\lg_{1+\varepsilon}(\varepsilon \cdot n) \cdot n / \lg^2 n) \subseteq O\left(\frac{n}{\lg n \lg(1+\varepsilon)}\right)$ bits. Again applying Taylor series for $1 / \lg(1 + \varepsilon) = O(1 / \varepsilon)$ gives $O(n / (\varepsilon \lg n))$ bits. Thus the total space cost is $O(n / \varepsilon)$ bits.

The time complexity of the binary search is different from a typical binary search. The number of values of k in the entire range is $O(\lg_{1+\varepsilon} n)$, so the complexity of the binary search is $O(\lg(\lg_{1+\varepsilon} n)) = O(\lg\left(\frac{\lg n}{\lg(1+\varepsilon)}\right)) = O(\lg\left(\frac{\lg n}{\varepsilon}\right)) = O(\lg \lg n + \lg(1/\varepsilon))$.

► **Lemma 3.** *There exists an $O(n/\varepsilon)$ -bit data structure that supports one-dimensional $(1 + \varepsilon)$ -approximate range mode queries in $O(\lg \lg n + \lg(1/\varepsilon))$ time.*

High Frequencies: $O(n/\varepsilon)$ -Bits $O(\lg(1/\varepsilon))$ Query Time. The bottleneck of the approach described in the previous section is the binary search on k . To speed up queries, we store an additional data structure that uses $O(n)$ bits but returns a 4-approximate range mode.

► **Lemma 4.** *There exists an $O(n)$ -bit data structure that supports one-dimensional approximate range mode queries in constant time with approximation factor 4.*

Proof. We assume n is a power of 2. We construct a network of fusion trees [9]. At the top level, we store two fusion trees $\mathcal{F}_{n/2,l}$ and $\mathcal{F}_{n/2,r}$. The tree $\mathcal{F}_{n/2,l}$ contains the values $e_1, \dots, e_{\lg n}$, where e_j is the largest index satisfying $F(c_{e_j..n/2}) = 2^j$. $\mathcal{F}_{n/2,r}$ contains the values $e_1, \dots, e_{\lg n}$, where e_j is the smallest index satisfying $F(c_{n/2..e_j}) = 2^j$. If a query crosses the middle index $n/2$, we query $\mathcal{F}_{n/2,l}$ to get p_1 , the smallest value greater than or equal to a , and we query $\mathcal{F}_{k,r}$ to get p_2 , the largest value less than or equal to b . We return p_1 if $F(c_{p_1..n/2}) > F(c_{n/2..p_2})$ and p_2 otherwise. Clearly, p_1 is a 2-approximate mode for $c_{a..n/2}$ and p_2 is a 2-approximate mode for $c_{n/2..b}$. The true mode has at least half its occurrences in one of these regions, so the value we return is a 4-approximate mode for $c_{a..b}$.

If the query does not cross the middle, it falls entirely in one of the two sides. We may therefore repeat our fusion tree scheme in a divide and conquer fashion, recursing on the two halves. Eventually, there will be a level of the recursion that intersects the query.

To analyze the total space used, we use the recurrence $S(n) = 2S(n/2) + O(\lg^2 n)$, which solves to $S(n) = O(n)$ bits.

To analyze the time complexity of the query, observe that the fusion trees on $O(\lg n)$ elements with word size $O(\lg n)$ support the necessary predecessor/successor queries in constant time. However, we must know which fusion trees to query. This involves finding the level of recursion in which the query range intersects the midpoint. This is equivalent to the highest set bit in the XOR of a and b , which can be determined in constant time in the word RAM model. With this information, we can do the necessary arithmetic to find the appropriate fusion trees to query, and thus query takes constant time. ◀

We now return to the $(1 + \varepsilon)$ -approximation. To answer a query $c_{a..b}$, we first query the 4-approximation structure of Lemma 4, which returns a corresponding frequency x . We now know $x \leq F(c_a, \dots, c_b) \leq 4x$. We have thus shrunk the number of values of k from Lemma 2 that need be tested for the $(1 + \varepsilon)$ -approximation from $\lceil \lg_{1+\varepsilon} n \rceil$ to $\lceil \lg_{1+\varepsilon}(4x/x) \rceil = \lceil \lg_{1+\varepsilon} 4 \rceil$. Thus our binary search now takes time $O(\lg\left(\frac{2}{\lg(1+\varepsilon)}\right)) = O(\lg(1/\varepsilon))$.

► **Theorem 5.** *There exists an $O(n/\varepsilon)$ -bit data structure that supports one-dimensional approximate range mode queries in $O(\lg(1/\varepsilon))$ time with approximation factor $1 + \varepsilon$.*

4 Dynamic Approximate Range Mode

In this section we consider the dynamic variant of the approximate range mode problem. We maintain our sequence $c_{a..b}$ under insertions and deletions, so that for an arbitrary query range $c_{a..b}$ an approximate range mode can be found efficiently.

The high-level approach is as follows. Similar to Section 3, for each $j \leq \lg_{1+\varepsilon} n$, our goal is to maintain a set of intervals \mathcal{I}_j such that the mode of a query range $c_{a..b}$ occurs more than $(1 + \varepsilon)^j$ times if and only if $c_{a..b}$ contains an interval in \mathcal{I}_j . Then, for all j and each interval $c_{l..r}$ in \mathcal{I}_j we maintain the points (l, r, j) in a data structure \mathcal{D} that supports the following range queries: given a query point (a, b) , return the highest j such that $a \leq l$ and $r \leq b$ for at least one point (l, r, j) in \mathcal{D} .

However, unlike the sets of intervals maintained in Section 3, our construction in this section satisfies the property that a single update affects only a *small* number of intervals in the sets \mathcal{I}_j for all j . We now proceed with the technical argument.

Let S_x denote the set of positions of the element x in the sequence $c_{1..n}$. We will denote by $S_x[i]$ the position of the i^{th} occurrence of element x . Let $I_x(l, r)$ denote the interval $^{c_{S_x[l]..S_x[r]}}$.

Now let $\delta = 1 + \varepsilon' = (1 + \varepsilon)^{1/3}$ and fix x . There are $f = F_x(c_{1..n})$ occurrences of element x in the full range $c_{1..n}$. We will maintain a subset of the $\binom{f}{2}$ possible intervals $I_x(l, r)$ in sets $\mathcal{I}_{j,x}$, $1 \leq j \leq \lceil \lg_\delta n \rceil$. We will not have need for nested intervals in $\mathcal{I}_{j,x}$; therefore, we can number each interval of $\mathcal{I}_{j,x}$ from left to right with s_k the start of interval k and e_k the end of interval k , satisfying $s_k \leq s_{k+1}$, $e_k \leq e_{k+1}$. We maintain the following two invariants on the intervals of $\mathcal{I}_{j,x}$: (1) $\delta^j \leq e_k - s_k \leq \delta^{j+1}$, and (2) $(\varepsilon'/2)\delta^j \leq s_{k+1} - s_k \leq \varepsilon'\delta^j$, and the number of positions of S_x not covered by an interval of $\mathcal{I}_{j,x}$ at either end is at most $(\varepsilon'/2)\delta^j$ (so $s_1 \leq (\varepsilon'/2)\delta^j$ and $f - r_{|\mathcal{I}_{j,x}|} \leq (\varepsilon'/2)\delta^j$). From our invariants we get the following proposition.

► **Proposition 6.** *An interval $I_x(l, r)$ intersects at most $2(r - l + 1)/(\varepsilon'\delta^j) + O(1/\varepsilon')$ intervals of $\mathcal{I}_{j,x}$.*

Proof. By Invariant (2), we have a gap size of between $(\varepsilon'/2)\delta^j$ and $\varepsilon'\delta^j$ elements between consecutive starting points of intervals of $\mathcal{I}_{j,x}$. Since each interval has size at most δ^{j+1} , the total number of intervals intersecting $I_x(l, r)$ is at most $2(r - l + 1 + 2\delta^{j+1})/(\varepsilon'\delta^j)$. ◀

For each interval $I_x(s_k, e_k)$ of $\mathcal{I}_{j,x}$, let $\text{pot}(I_x(s_k, e_k)) = e_k - s_k + 1$ denote the number of elements of S_x (and thus positions in the original sequence $c_{1..n}$) that fall between s_k and e_k . When we insert or delete an element x , by Proposition 6, we must update the pot values of $O(1/\varepsilon')$ intervals of $\mathcal{I}_{j,x}$. Across all j , $1 \leq j \leq \lceil \lg_\delta n \rceil$, $O((1/\varepsilon') \lg_\delta n)$ intervals are affected.

During the updates, each affected $\text{pot}(I_x(s_k, e_k))$ value is incremented or decremented by one. If, for an interval $I_x(s_k, e_k)$ in $\mathcal{I}_{j,x}$, Invariant (1) is violated by the update, then we rearrange the intervals in the neighborhood of $I_x(s_k, e_k)$ as follows. Consider all intervals of $\mathcal{I}_{j,x}$ that intersect with $I_x(s_k, e_{k+1})$. By Proposition 6, there are $O(1/\varepsilon')$ such intervals. We remove said intervals and create new intervals in their place with exactly $\lceil (1 + \varepsilon'/2)\delta^j \rceil$ positions of x that fall in each interval. Furthermore, we space them so that Invariant (2) holds when the new intervals are inserted into $\mathcal{I}_{j,x}$.

To build these intervals, we must be able to efficiently search for elements by rank in S_x . As this will not dominate the update cost, we can use a typical order statistic tree, with $O(\lg f) = O(\lg n)$ query and update time. We may construct the new intervals satisfying invariants (1) and (2) with a constant number of queries on S_x per interval, thus in $O((1/\varepsilon') \lg n)$ time overall.

We can analyze the total cost of rebuilds as follows. On each update, we affect $O(1/\varepsilon')$ intervals at each level. However, the affect on pot is the same for each interval, and when we rebuild, we rebuild a superset of the intervals affected on update. It follows that the total amortized cost of rebuilds is $\sum_{j=1}^{\lceil \lg_\delta n \rceil} (1/\delta^j) \cdot O((1/\varepsilon') \lg n) = O((\lg n)/\varepsilon'^2)$ per update.

57:10 On Approximate Range Mode and Range Selection

Further, in each update we must update S_x and update the `pot` values. These take time $O(\lg n)$ and $O((1/\varepsilon') \lg_\delta n) = O(\lg n/\varepsilon'^2)$, respectively. So far we pay $O(\lg n/\varepsilon'^2)$ per update, but we have yet to describe the data structure that holds each $\mathcal{I}_{j,x}$, which will also need to be updated during rebuilds.

Consider each interval $I_x(s_k, e_k)$ of $\mathcal{I}_{j,x}$ as a point (s_k, e_k, j) . We store each interval of $\mathcal{I}_{j,x}$, across all $1 \leq j \leq \lceil \lg_\delta n \rceil$ and all x , in a data structure \mathcal{D} that supports the following range queries: given a query point (a, b) , return the highest j such that $a \leq l$ and $r \leq b$ for at least one point (l, r, j) in \mathcal{D} . Associated with each point, we keep the element x from which it originated.

We first must consider the number of intervals (and thus points) stored in \mathcal{D} . As before, we assume element x occurs $f = F_x(c_{1..n})$ times in $c_{1..n}$. Then $|\mathcal{I}_{j,x}| = O(f/[\varepsilon'^{\delta^j}])$. Across all levels, we can bound the total number of intervals at $O(f \lg_\delta n) = O(f \lg n/\varepsilon')$ or $O(f/\varepsilon'^2)$. Accounting for all x , the number of intervals in \mathcal{D} will be $O(m) = O(\min(n \lg n/\varepsilon', n/\varepsilon'^2))$.

► **Lemma 7.** *Data structure \mathcal{D} can be stored in $O(n \lg n)$ bits, where n is the number of elements in \mathcal{D} . Queries and updates can be supported in $O(\lg n/\lg \lg n)$ time.*

Proof. Let P denote the set of points to be stored in our data structure. Here we use $\varepsilon > 0$ independently of the rest of the section. We start by considering the special case when the second coordinate is bounded by $\lg^\varepsilon n$, i.e., $r \leq \lg^\varepsilon n$ for all $(l, r, j) \in P$. In this case it is sufficient to store $\lg n$ points for every possible value of b : let $\max_{r,j}$ denote the biggest first coordinate of a point (l', r', j') in P with $r' = r$, $j' = j$ ($\max_{r,j} = \max\{l' \mid (l', r', j') \in P \text{ and } r' = r, j' = j\}$). The answer to a query (a, b) is the largest j that satisfies $a \leq \max_{r,j}$ for some $r \leq b$. We keep all values $\max_{r,j}$ such that P contains at least one point (l, r, j) for some l , and store them in increasing order. We group them in blocks of size $\Theta(\lg^{1-\varepsilon} n)$ and we keep a global lookup table of size $o(n)$ bits that allows answering queries within any possible block.

Also, in a local lookup table of size $O(\lg^{3\varepsilon} n)$ bits we store for each block and every possible value of r the index of the block preceding it which maximizes the value of j given r . We also store a fusion tree on the values $\max_{r,j}$ so that we can compute the rank of a within these values in constant time. Given a query, we compute in constant time the block which the predecessor of a belongs to and use table lookup on that block and one other block preceding it to get the answer. Updates also take constant time since the size of individual blocks and the local lookup table fit in a single word.

A general query can be reduced to the above described special case by using a range tree with node degree $\lg^\varepsilon n$ that splits the points on the value of their second coordinate. Although every point is stored in $O(\lg n/\lg \lg n)$ nodes, our data structure uses linear space. Let $P(u)$ denote the set of points stored in a node u . We replace the second coordinate of each point $p \in P(u)$ with the index i of the child node u_i such that $p \in P(u_i)$. We keep the above described special case data structure in every node $P(u)$, but we do not store the set $P(u)$ itself. A query interval can be fully covered by $O(\lg n/\lg \lg n)$ tree nodes. We query the data structure in each one of them and return the maximum value j in $O(\lg n/\lg \lg n)$ time. Similarly, an update affects the special case data structure in $O(\lg n/\lg \lg n)$ nodes and requires $O(\lg n/\lg \lg n)$ time.

The total space usage is $O(n \log n)$ bits because we spend $O(\min(\log^{2+\varepsilon} n, |P(u)| \lg n))$ bits in each node u of the range tree. To prove this bound, we classify nodes into low and high nodes. Low nodes are the nodes in the lowest $(1 + 2/\varepsilon)$ levels of the tree and the rest of the nodes are high nodes. We also store the set of points $P(u)$ in every low node u . Thus we spend $O(|P(u)| \lg n)$ bits in every low node, so the total space consumed by all low nodes

is $O((1/\varepsilon)n \lg n)$ bits. We spend $O(\lg^{2+\varepsilon} n) = O(|P(u)|)$ bits in every high node because $|P(u)| \geq \lg^{2+\varepsilon} n$. Since the total number of points in all $P(u)$ is $O(n \lg n / \lg \lg n)$, the total space consumed by high nodes is $O(n \lg n / \lg \lg n)$ bits. ◀

Now suppose we are given a query range $c_{a..b}$. We find the largest j such that some interval from $\mathcal{I}_{j,x}$ for some x is contained in $c_{a..b}$. Using data structure \mathcal{D} from Lemma 7, we can compute the index j in $O(\lg n / \lg \lg n)$ time. We return the element x associated with j .

► **Lemma 8.** *The element x returned is a $(1 + \varepsilon)$ -approximate mode of query range $c_{a..b}$.*

Proof. If $c_{a..b}$ contains an interval from $\mathcal{I}_{j,x}$, then x occurs at least δ^j times in $c_{a..b}$. On the other hand, we can show that if some y occurs δ^{j+3} times in $c_{a..b}$, then $c_{a..b}$ contains an interval from $\mathcal{I}_{j+1,y}$. Recall $\delta^{j+3} = (1 + \varepsilon')\delta^{j+2}$. Each interval of $\mathcal{I}_{j+1,y}$ has size at most δ^{j+2} and there is a gap of at most $\varepsilon'\delta^j$ elements of y between the start of every interval in $\mathcal{I}_{j+1,y}$. Then since $\varepsilon'\delta^{j+1} + \delta^{j+2} < (1 + \varepsilon')\delta^{j+2}$, it must be that an interval of $\mathcal{I}_{j+1,y}$ falls in the query range $c_{a..b}$. We therefore know $\delta^j \leq F(c_{a..b}) < \delta^{j+3} = (1 + \varepsilon)\delta^j$. It follows that x is a $(1 + \varepsilon)$ -approximate mode of query range $c_{a..b}$. ◀

This gives us the main theorem for the section.

► **Theorem 9.** *There exists an $O(m \lg m)$ -bit data structure, where $m = \min(n \lg n / \varepsilon, n / \varepsilon^2)$ that answers $(1 + \varepsilon)$ -approximate range mode queries in $O(\lg m / \lg \lg m)$ time. Insertions and deletions are supported in $O(\lg n / \varepsilon^2)$ time.*

Proof. We have $(1 + \varepsilon')^3 = (1 + \varepsilon)$ and $(1 + \varepsilon)^3 = \varepsilon^3 + 3\varepsilon^2 + 3\varepsilon + 1$. The smallest exponent dominates $O(1/\varepsilon')$ since $\varepsilon \leq 1$ and thus $\varepsilon' < \varepsilon \leq 1$. Thus we have $1/\varepsilon' = O(1/\varepsilon)$. As previously stated, the number of intervals in \mathcal{D} is $O(m)$, where $m = \min(n \lg n / \varepsilon, n / \varepsilon^2)$. The space bound for \mathcal{D} is thus $O(m \lg m) = \Omega(n \lg n)$ bits, which dominates the total space cost. The query time is $O(\lg m / \lg \lg m)$.

The update cost has four components: Updating \mathcal{D} , updating S_α , and updating pot values for all affected intervals. As previously mentioned, the latter three are dominated by $O(\lg n / \varepsilon'^2) = O(\lg n / \varepsilon^2)$. Via Lemma 7, the cost of updating \mathcal{D} is $O(\lg m / \lg \lg m)$. Since m is no more than n / ε^2 , $\lg m / \lg \lg m$ is dominated by $O(\lg n / \varepsilon^2)$. In total, the cost of updates is $O(\lg n / \varepsilon^2)$. ◀

We can use our dynamic data structure to obtain a result for approximate range mode queries on two-dimensional points. Our data structure can find approximate mode in the case when the query range is bounded on three sides.

► **Corollary 10.** *There exists a data structure that supports three-sided two-dimensional approximate range mode queries in $O(\log m)$ time and uses $O(m \log m)$ words of $\log n$ bits, where $m = \min(n \lg n / \varepsilon, n / \varepsilon^2)$.*

Proof. Using the technique introduced by Dietz in [6], we can transform a data structure that supports updates in $u(n)$ time and queries in $q(n)$ time into an offline partially persistent data structure that answers queries in $O(q(n) \cdot \log \log n)$ time and uses $O(n \cdot u(n))$ words of space. Using sweep line technique, we can transform an offline partially persistent data structure for one-dimensional queries into a static data structure for three-sided queries with the same time and space bounds. ◀

5 Approximate Range Median and Range Selection

In this section we present solutions to approximate range selection queries. As discussed previously, a range selection query takes two indices a, b of a sequence c_1, \dots, c_n and must return the index of an element x whose rank in $c_{a..b}$ is between $k - \alpha(b - a + 1)$ and $k + \alpha(b - a + 1)$. We study two variants. In the first variant, the rank k is supplied prior to construction of the data structure. In the second variant, we allow k to be specified at query time. Here rank is defined so the i th smallest element in the range has rank i . We also support a specific k depending on the size of the range, i.e. $f(b - a + 1) = \lceil (b - a + 1)/2 \rceil$, which is the range median problem. We make the restrictions $f(x) \leq f(x + 1) \leq f(x) + 1$ and $1 \leq f(x) \leq x$.

► **Theorem 11.** *Any one-dimensional approximate range median data structure requires $\Omega(n)$ bits.*

Proof. Assume n is even. Divide the sequence S of size n to $n/2$ blocks each of size 2. We say that S satisfies property $(*)$ if for each block b in S one of the following two conditions hold:

- either b consists of $\{1, 2\}$,
- or b consists of $\{2, 1\}$.

Clearly, the number of sequences that satisfy $(*)$ is $2^{(n/2)}$ since there exists $n/2$ blocks in a sequence of size n and each block can have one of two different values. Moreover for any two distinct sequences S_1 and S_2 satisfying $(*)$ differing at block b , the approximate range selection query must be exact on block b , and therefore must return different values. Thus, the information theoretic lower bound for storing an approximate range median data structure is $\Omega(\lg 2^{(n/2)}) = \Omega(n/2) = \Omega(n)$ bits. ◀

Fixed Rank $f(b - a + 1) = k$ Selection. We first address the range selection variant with a fixed rank $f(b - a + 1) = k$. We use a similar approach to the one in Lemma 4. We again assume n is a power of 2. At the top level, we store values $m_{n/2, i, j}$ ($1 \leq i, j \leq \lceil \lg_{1+\alpha} n \rceil$). Let $r_i = n/2 - (1 + \alpha)^i$ and $s_j = n/2 + 1 + (1 + \alpha)^j$. Then $m_{n/2, i, j}$ is the element of rank $f(s_j - r_i + 1)$ in the range $c_{[r_i]..[s_j]}$. We then build the structure recursively on the left and right halves of the full range.

Given a query range $c_{a..b}$, we find the appropriate element $m_{t, i, j}$ where $a \leq t$, $t + 1 \leq b$, and i and j are largest possible satisfying $a \leq r_i$ and $s_j \leq b$. We return $m_{t, i, j}$.

► **Lemma 12.** *The above data structure returns an α -approximate fixed-rank k element of any query range $c_{a..b}$.*

Proof. Let $x = r_i - a$ and $y = b - s_j$. Consider the size of x . If we let $z = (1 + \alpha)^i$, then $x + z < (1 + \alpha)z$. It follows $x < \alpha z$. Since $z \leq (t - a + 1)$, and applying similarly for y , we can show $x < \alpha(t - a + 1)$ and $y < \alpha(b - t)$. The elements in the ranges represented by x and y shift the true rank k element of $c_{a..b}$ at most $x + y < \alpha(b - a + 1)$ ranks from $m_{t, i, j}$. It follows that $m_{t, i, j}$ is an α -approximate rank k element for range s_a, \dots, s_b . ◀

As for Theorem 4, to find the level to query, we find the highest set bit of a XOR b , then find the appropriate index $m_{t, i, j}$ via arithmetic. In total, the query takes constant time.

We now analyze the space required. At the top level, we use $O(\lg_{1+\alpha}^2(n) \cdot \lg n)$ bits, which is equal to $O(\frac{\lg^3 n}{\lg^2(1+\alpha)}) = O(\lg^3 n / \alpha^2)$ bits. Therefore our recurrence is $S(n) = 2S(n/2) + O(\lg^3 n / \alpha^2)$. The recursion tree is leaf-heavy, with total space amounting to $O(n / \alpha^2)$ bits.

► **Theorem 13.** *There exists an $O(n / \alpha^2)$ -bit data structure that supports one-dimensional α -approximate fixed-rank $f(b - a + 1) = k$ selection queries in constant time.*

Online Rank k Selection. Our data structure from the previous section can be adapted to support queries that specify the rank k at query time. We again assume n is a power of 2. Let $\delta = 1 + \alpha/2$. At the top level we now store values $m_{n/2,i,j,l}$ ($1 \leq i, j, \leq \lceil \lg_\delta n \rceil$, $0 \leq l \leq \lfloor 1/\alpha \rfloor$). Again, we let $r_i = n/2 - \delta^i$ and $s_j = n/2 + 1 + \delta^j$. However, this time, $m_{n/2,i,j,l}$ represents the element of rank $q_l = l\alpha \cdot (s_j - r_i + 1) + 1$ in $c_{r_i..s_j}$. As q_l may be fractional, for simplicity we just store both rank $\lfloor q_l \rfloor$ and $\lceil q_l \rceil$ elements. We build this structure recursively on both halves of the full range.

Given a query $s_{a..b}$, we again find the appropriate element $m_{t,i,j,l}$ where $a \leq t$, $t + 1 \leq b$, i and j are largest possible satisfying $a \leq r_i$ and $s_j \leq b$, and l is chosen so q_l is as close to k as possible. We return $m_{t,i,j,l}$.

► **Lemma 14.** *The above data structure returns an α -approximate rank k element of any query range $c_{a..b}$ and specified rank k .*

Proof. Again let $x = r_i - a$ and $y = b - s_j$. For the same reasons as in the proof of Lemma 12, we have $x + y < \alpha(b - a + 1)/2$.

There are no more than $\alpha \cdot (s_j - r_i + 1) \leq \alpha \cdot (b - a + 1)$ ranks between each consecutive q_l and q_{l+1} . Thus our chosen q_l satisfies $|q_l - k| < \lfloor \alpha(b - a + 1)/2 \rfloor$. It follows that $m_{t,i,j,l}$ is no more than $\alpha \cdot (b - a + 1)$ ranks away from the true rank k element in range $c_{a..b}$. ◀

The query time follows as in the previous section. However, we must account for the additional space usage. Our recurrence is now $T(n) = 2T(n/2) + O(\lg^3 n/\alpha^3)$, from the additional $1/\alpha$ factor in the space cost at each level. This totals to $O(n/\alpha^3)$ bits.

► **Theorem 15.** *There exists an $O(n/\alpha^3)$ -bit data structure that supports one dimensional α -approximate online rank k selection queries in constant time.*

References

- 1 Prosenjit Bose, Evangelos Kranakis, Pat Morin, and Yihui Tang. Approximate Range Mode and Range Median Queries. In *Proceedings of the 22nd Annual Symposium on Theoretical Aspects of Computer Science*, pages 377–388, 2005.
- 2 Gerth Stølting Brodal, Beat Gfeller, Allan Grønlund Jørgensen, and Peter Sanders. Towards optimal range medians. *Theoretical Computer Science*, 412(24):2588–2601, 2011.
- 3 Gerth Stølting Brodal and Allan Grønlund Jørgensen. Data Structures for Range Median Queries. In *Proceedings of the 20th International Symposium on Algorithms and Computation*, pages 822–831, 2009.
- 4 Timothy M Chan, Stephane Durocher, Kasper Green Larsen, Jason Morrison, and Bryan T Wilkinson. Linear-space data structures for range mode query in arrays. *Theory of Computing Systems*, 55(4):719–741, 2014.
- 5 Timothy M. Chan and Bryan T. Wilkinson. Adaptive and Approximate Orthogonal Range Counting. *ACM Trans. Algorithms*, 12(4):45:1–45:15, 2016.
- 6 Paul F. Dietz. Fully Persistent Arrays (Extended Array). In *Proceedings of Workshop on Algorithms and Data Structures (WADS)*, pages 67–74, 1989. doi:10.1007/3-540-51542-9_8.
- 7 Hicham El-Zein, Meng He, J. Ian Munro, and Bryce Sandlund. Improved Time and Space Bounds for Dynamic Range Mode. In *Proceedings of the 26th Annual European Symposium on Algorithms*, pages 25:1–25:13, 2018.
- 8 Johannes Fischer and Volker Heun. Space-Efficient Preprocessing Schemes for Range Minimum Queries on Static Arrays. *SIAM Journal on Computing*, 40(2):465–492, 2011.
- 9 M. L. Fredman and D. E. Willard. BLASTING Through the Information Theoretic Barrier with FUSION TREES. In *Proceedings of the Twenty-second Annual ACM Symposium on Theory of Computing*, STOC '90, pages 1–7, 1990.

57:14 On Approximate Range Mode and Range Selection

- 10 Travis Gagie, Simon J. Puglisi, and Andrew Turpin. Range Quantile Queries: Another Virtue of Wavelet Trees. In *Proceedings of the 16th International Symposium on String Processing and Information Retrieval*, pages 1–6, 2009.
- 11 Pawel Gawrychowski and Patrick K. Nicholson. Optimal Encodings for Range Top- k Selection, and Min-Max. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming*, pages 593–604, 2015.
- 12 Beat Gfeller and Peter Sanders. Towards Optimal Range Medians. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming*, pages 475–486, 2009.
- 13 Mark Greve, Allan Grønlund Jørgensen, Kasper Dalgaard Larsen, and Jakob Truelsen. Cell Probe Lower Bounds and Approximations for Range Mode. In *Proceedings of the 37th International Colloquium on Automata, Languages and Programming*, pages 605–616, 2010.
- 14 Roberto Grossi, John Iacono, Gonzalo Navarro, Rajeev Raman, and S. Srinivasa Rao. Asymptotically Optimal Encodings of Range Data Structures for Selection and Top- k Queries. *ACM Transactions on Algorithms*, 13(2):28:1–28:31, 2017.
- 15 Meng He, J. Ian Munro, and Patrick K. Nicholson. Dynamic Range Selection in Linear Space. In *Proceedings of the 22nd International Symposium*, pages 160–169, 2011.
- 16 Allan Grønlund Jørgensen and Kasper Green Larsen. Range Selection and Median: Tight Cell Probe Lower Bounds and Adaptive Data Structures. In *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 805–813, 2011.
- 17 Danny Krizanc, Pat Morin, and Michiel H. M. Smid. Range Mode and Range Median Queries on Lists and Trees. *Nord. J. Comput.*, 12(1):1–17, 2005.
- 18 M. Patrascu. Succincter. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 305–313, 2008.
- 19 Holger Petersen. Improved Bounds for Range Mode and Range Median Queries. In *Proceedings of the 34th Conference on Current Trends in Theory and Practice of Computer Science*, pages 418–423, 2008.
- 20 Holger Petersen and Szymon Grabowski. Range mode and range median queries in constant time and sub-quadratic space. *Information Processing Letters*, 109(4):225–228, 2009.

External Memory Planar Point Location with Fast Updates

John Iacono

Université Libre de Bruxelles, Belgium
New York University, USA
<http://www.johniacono.com>
john@johniacono.com

Ben Karsin

Université Libre de Bruxelles, Belgium
<https://www.benkarsin.com>
bkarsin@gmail.com

Grigorios Koumoutsos

Université Libre de Bruxelles, Belgium
<http://homepages.ulb.ac.be/~gkoumout>
gregkoumoutsos@gmail.com

Abstract

We study dynamic planar point location in the *External Memory Model* or Disk Access Model (DAM). Previous work in this model achieves polylog query and polylog amortized update time. We present a data structure with $O(\log_B^2 N)$ query time and $O(\frac{1}{B^{1-\epsilon}} \log_B N)$ amortized update time, where N is the number of segments, B the block size and ϵ is a small positive constant, under the assumption that all faces have constant size. This is a $B^{1-\epsilon}$ factor faster for updates than the fastest previous structure, and brings the cost of insertion and deletion down to subconstant amortized time for reasonable choices of N and B . Our structure solves the problem of vertical ray-shooting queries among a dynamic set of interior-disjoint line segments; this is well-known to solve dynamic planar point location for a connected subdivision of the plane with faces of constant size.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis; Theory of computation → Computational geometry; Theory of computation → Models of computation

Keywords and phrases point location, data structures, dynamic algorithms, computational geometry

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.58

Funding This work was supported by the Fonds de la Recherche Scientifique-FNRS under Grant no MISU F 6001 1 and by NSF Grant CCF-1533564.

1 Introduction

The *dynamic planar point location* problem is one of the most fundamental and extensively studied problems in geometric data structures, and is defined as follows: We are given a connected planar polygonal subdivision Π with N edges. For any given query point p , the goal is to find the face of Π that contains p , subject to insertions and deletions of edges. Here we focus on subdivisions Π such that each face has constant number of edges. An equivalent formulation, which we use here is as follows: given a set S of N interior-disjoint line segments in the plane, for any given query point p , report the first line segment in S that a vertical upwards-facing ray from p intersects, subject to insertions and deletions of segments.

Dynamic planar point location has many applications in spatial databases, geographic information systems (GIS), computer graphics, etc. Moreover it is a natural generalization of the dynamic dictionary problem with predecessor queries; this problem can be seen as the one dimensional variant of planar point location.



© John Iacono, Ben Karsin, and Grigorios Koumoutsos;
licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 58; pp. 58:1–58:18

Leibniz International Proceedings in Informatics



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

In this paper we focus on the External Memory model, also known as the Disk Access Model (DAM) [2]. The DAM is the standard method of designing algorithms that efficiently execute on large datasets stored in secondary storage. This model assumes a two-level memory hierarchy, called disk and internal memory and it is parameterized by values M and B ; the disk is partitioned into blocks of size B , of which M/B can be stored in memory at any given moment. The cost of an algorithm in the DAM is the number of block transfers between memory and disk, called Input-Output operations (I/Os). The quintessential DAM-model data structure is the B-Tree [11]. See [25, 26] for surveys. Many applications of dynamic planar point location, such as GIS problems, must efficiently process datasets that are too massive to fit in internal memory, thus it is of great relevance and interest to consider the problem in the DAM and to devise I/O efficient algorithms.

1.1 Previous Work

RAM Model. In the RAM model (the leading model for applications where all data fit in the internal memory) the dynamic planar point location problem has been extensively studied [4, 10, 15, 18, 19, 21]. It is a major and long-standing open problem in computational geometry to design a data structure that supports queries and updates in $O(\log N)$ time [16, 17, 24], i.e., to achieve the same bounds as for the dynamic dictionary problem. In a recent breakthrough, Chan and Nekrich in FOCS'15 [15] presented a data structure supporting queries in $O(\log N(\log \log N)^2)$ time and updates in $O(\log N(\log \log N))$ time. They also showed the tradeoff of supporting queries in $O(\log N)$ time and updates in $O((\log N)^{1+\epsilon})$ time or vice-versa for $\epsilon > 0$.

Recently Oh and Ahn [23] presented the first data structure for a more general setting where the polygonal subdivision Π is not necessarily connected; their data structure supports queries in $O(\log N(\log \log N)^2)$ time and updates in $O(\sqrt{N} \log N(\log \log N)^{3/2})$ amortized time.

External Memory model. (See Table 1). Several data structures have been presented over the years which support queries and updates in $\text{polylog}(N)$ I/Os [1, 5, 7]. Table 1 contains a list of results of prior work. The best update bound known is by Arge, Brodal and Rao [5] and achieves $O(\log_B N)$ amortized I/Os. The query time of their data structure is $O(\log_B^2 N)$. Very recently, the first data structure that supports queries in $o(\log_B^2 N)$ I/Os was announced by Munro and Nekrich [22]. In particular they support queries in $O(\log_B N(\log \log_B N)^3)$ I/Os. However their update time is slightly worse than logarithmic, $O(\log_B N(\log \log_B N)^2)$. In all those works the bounds are obtained by solving the problem of vertical ray-shooting.

Fast Updates in External Memory. One of the most intriguing and practically relevant features of the external memory model is that it allows fast updates. For the dynamic dictionary problem with predecessor queries, the optimal update bound in the RAM model is $O(\log N)$. In external memory, however, B -trees achieve the optimal query time of $O(\log_B N)$ and typical update time of $O(\log_B N)$, although substantially faster update times are possible. Brodal and Fagerberg [14] showed that $O(\frac{1}{B^{1-\epsilon}} \log_B N)$ amortized I/Os per update can be supported, for small positive constant, ϵ , while retaining $O(\log_B N)$ -time queries; they further showed that this is an asymptotically optimal tradeoff between updates and queries. Observe that this update bound is a huge speedup from $O(\log_B N)$ and that for reasonable choices of parameters, e.g. $B \geq 1000$, $N < 10^{93}$, $\epsilon = \frac{1}{2}$, this yields a subconstant amortized number of I/Os per update. A similar update bound was later achieved for other dynamic problems like three-sided range reporting and top- k queries [13].

■ **Table 1** Overview of results on dynamic planar point location in external memory. Results marked with M are for monotone subdivisions and G for general ray-shooting among non-intersecting segments. Query bounds are worst-case and update bounds are amortized. Space usage is measured in words. Here ϵ is a constant such that $0 < \epsilon \leq 1/2$.

Reference	Space	Query Time	Insertion Time	Deletion Time	
Agarwal et al. [1]	$O(N)$	$O(\log_B^2 N)$	$O(\log_B^2 N)$	$O(\log_B^2 N)$	M
Arge and Vahrenhold [7]	$O(N)$	$O(\log_B^2 N)$	$O(\log_B^2 N)$	$O(\log_B N)$	G
Arge et al. [5]	$O(N)$	$O(\log_B^2 N)$	$O(\log_B N)$	$O(\log_B N)$	G
Munro and Nekrich [22]	$O(N)$	$O(\log_B N \log^3 \log_B N)$	$O(\log_B N \log^2 \log_B N)$	$O(\log_B N \log^2 \log_B N)$	G
This paper	$O(N)$	$O(\log_B^2 N)$	$O((\log_B N)/B^{1-\epsilon})$	$O((\log_B N)/B^{1-\epsilon})$	G

Given this progress and the fact that in the RAM model the bounds achieved for planar point location and the dictionary problem are believed to coincide, it is natural to conjecture that a similar update bound can be achieved for the dynamic planar point location problem. However, to date no result has been presented that achieves sublogarithmic insertion or deletion time.

1.2 Our Results

We consider the dynamic planar point location problem in the external memory model and present the first data structure with sublogarithmic amortized update time of $O(\frac{1}{B^{1-\epsilon}} \log_B N)$ I/Os. Prior to our work, the best update bound for both insertions and deletions was $O(\log_B N)$, achieved by Arge et al. [5]. Our main result is:

► **Theorem 1 (Main result).** *For any constant $0 < \epsilon \leq 1/2$, there exists a data structure which uses $O(N)$ space, answers planar point location queries in $O((1/\epsilon)^2 \cdot \log_B^2 N) = O(\log_B^2 N)$ I/Os and supports insertions and deletions in $O(\log_B N / (\epsilon \cdot B^{1-\epsilon})) = O((\log_B N) / B^{1-\epsilon})$ amortized I/Os. The data structure can be constructed in $O((N/B) \log_B N)$ I/Os.*

To obtain this result, several techniques are used. Our primary data structure is an augmented *interval tree* [20]. We combine both the primary interval tree and two auxiliary structures described below with the *buffering* technique [3, 14] to improve insertion and deletion bounds. In Section 2 we prove Theorem 1 using our auxiliary structures as black boxes and omit some technical details relating to rebuilding; these details are deferred to Appendix 5.

Similarly to previous work, we focus on solving the problem of vertical ray-shooting queries. Our first auxiliary structure answers vertical ray-shooting queries among non-intersecting segments whose right (left) endpoints lie on the same vertical line. This is called the *left (right) structure* (in Section 2 it will be clear why we choose this terminology and not vice-versa). Left/Right structures of Agarwal et al. [1], which support queries and updates in $O(\log_B K)$ I/Os, are used by several prior works [1, 5, 7]. Our structure improves on their result by reducing the update bound by a factor of $B^{1-\epsilon}$. We obtain the following result, the proof of which is the topic of Section 3:

► **Theorem 2 (Left/right structure).** *For a set of K non-intersecting segments whose right (left) endpoints lie in the same vertical line and any constant $0 < \epsilon \leq 1/2$, we can create a data structure which supports vertical ray-shooting queries in $O((1/\epsilon) \cdot \log_B K) = O(\log_B K)$*

I/Os and insertions and deletions in $O((\log_B K)/(\epsilon \cdot B^{1-\epsilon})) = O((\log_B K)/B^{1-\epsilon})$ amortized I/Os. This data structure uses $O(K)$ space and it can be constructed in $O((K/B) \log_B K)$ I/Os. If the segments are already sorted, it can be constructed in $O(K/B)$ I/Os.

Our second auxiliary structure answers vertical ray-shooting queries among non-intersecting segments whose endpoints lie in a set of $B^{\epsilon/2} + 1$ vertical lines. These vertical lines define $B^{\epsilon/2}$ vertical slabs, hence the structure is called a *multislab* structure. We obtain the following result, the proof of which is the topic of Section 4:

► **Theorem 3 (Multislab structure).** *For any constant $0 < \epsilon \leq 1/2$ and set of K non-intersecting segments whose endpoints lie in $B^{\epsilon/2} + 1$ vertical lines, we can create a data structure which supports vertical ray-shooting queries in $O((1/\epsilon) \cdot \log_B K) = O(\log_B K)$ I/Os and insertions and deletions in $O((\log_B K)/(\epsilon \cdot B^{1-\epsilon})) = O((\log_B K)/B^{1-\epsilon})$ amortized I/Os. This data structure uses $O(K)$ space and it can be constructed in $O((K/B) \log_B K)$ I/Os. If the segments are already sorted according to a total order, it can be constructed in $O(K/B)$ I/Os.*

A major challenge faced by previous multislab structures is how to efficiently support insertions. At a high-level, it is hard to deal with insertions in cases where a total order is maintained: each time a new segment gets inserted we need to determine its position in the total order, which cannot be done quickly. Arge and Vitter [7] developed a deletion-only multislab data structure and then used the so-called logarithmic method [12] which allowed them to handle insertions in $O(\log_B^2 K)$ I/Os. Later Arge, Brodal and Rao [5] developed a more complicated multislab structure supporting insertions in amortized $O(\log_B K)$ I/Os by performing separate case analysis depending on the value of B .

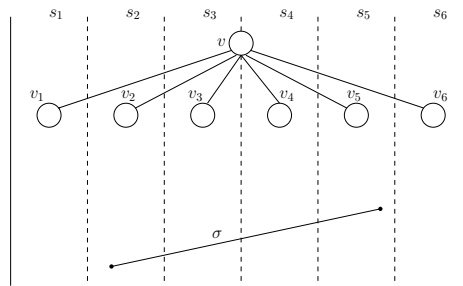
Here, we support insertions in a much simpler way by breaking each inserted segment into smaller *unit segments* whose endpoints lie on two consecutive vertical lines and can be compared easily to the segments already stored. This way, we are able to support insertions easily in $O(\log_B K)$ I/Os. Finally, we add buffering and obtain sublogarithmic update bounds.

1.3 Notation and Preliminaries

External Memory Model. Throughout this paper we focus on the external memory model of computation. N denotes the number of segments in the planar subdivision, B the block size and M the number of elements that fit in internal memory. We assume that $M \ll N$ and $2 \leq B \leq \sqrt{M}$ (the *tall cache assumption*). It is well-known that sorting K elements requires $\Theta((K/B) \log_{M/B}(K/B))$ I/Os [2]. Given that $B \leq \sqrt{M}$, this bound is $O((K/B) \log_B K)$. We use this bound for sorting in many places without further explanation.

Ray-shooting Queries. In the rest of this paper, we focus on answering vertical ray-shooting queries in a dynamic set of non-intersecting line segments. Let S be the set of segments of the polygonal subdivision Π . Given a query point p , the answer to a vertical ray-shooting query is the the first segment of S hit by a vertical ray emanating from a query point in the $(+y)$ direction. Based on standard techniques (see e.g. [7]), for connected polygonal subdivisions Π with faces of size $O(1)$, a planar point location query for a point p can be answered in $O(\log_B N)$ I/Os after answering a vertical ray-shooting query for p .

B^ϵ -Trees. All tree structures that we will use are variants of the B^ϵ -Trees [14] which are B -trees except that the internal nodes have at most B^ϵ (and not B) children; the leaves still store $\Theta(B)$ data items. For constant ϵ , this does not change the asymptotic height of the tree or the search cost, both remain $O((1/\epsilon) \cdot \log_B N) = O(\log_B N)$.



■ **Figure 1** The slab of node v of the interval tree \mathcal{I} is divided into slabs s_1, \dots, s_6 corresponding to its children v_1, \dots, v_6 . Segment σ is assigned to node v , with left subsegment in slab s_2 , right subsegment in s_5 and the middle subsegment crosses slabs s_3, s_4 .

2 Overall Structure

In this Section we prove Theorem 1, using the data structures of Theorems 2 and 3 (detailed in Sections 3 and 4, respectively). Given N non-intersecting segments in the plane and a constant $0 < \epsilon \leq 1/2$, we construct a $O(N)$ -space data structure which answers vertical ray-shooting queries in $O((1/\epsilon)^2 \cdot \log_B^2 N) = O(\log_B^2 N)$ I/Os and supports updates in $O((\log_B N)/(\epsilon \cdot B^{1-\epsilon})) = O((\log_B N)/B^{1-\epsilon})$ amortized I/Os. Throughout this section we let $\epsilon' = \epsilon/2$.

The Data Structure. As in the previous works on planar point location, our primary data structure is based on the *interval tree* (the external interval tree defined in [9]). Our interval tree \mathcal{I} is a $B^{\epsilon'}$ -tree which stores the x -coordinates of segment endpoints in its leaves. Here we assume for clarity of presentation that the interval tree is static, i.e. all new segments inserted share x -coordinates with already stored segments; in Appendix 5 we remove this assumption and extend our data structure to accommodate new x -coordinates and achieve the bounds of Theorem 1.

Each node of \mathcal{I} is associated with several secondary structures, as we explain later, and each segment is stored in the secondary structures of exactly one node of \mathcal{I} . Each node v of \mathcal{I} is associated with a vertical slab s_v . The slab of the root is the whole plane. For an internal node v , the slab s_v is divided into $B^{\epsilon'}$ vertical slabs $s_1, \dots, s_{B^{\epsilon'}}$ corresponding to the children of v , separated by vertical lines called *slab boundaries*, such that each slab s_i contains the same number of vertices of Π from slab s_v .

Let S be the set of segments that compose Π . Each segment $\sigma \in S$ is *assigned* to a node v of \mathcal{I} . This is the highest node v of \mathcal{I} such that σ is completely contained in slab s_v and intersects at least one slab boundary partitioning s_v ; if such an internal node v does not exist, then σ is assigned to a leaf v such that σ is completely contained in its slab s_v . Segments assigned to internal nodes are stored in the secondary structures of those nodes, whereas segments assigned to leaves are stored explicitly in the corresponding leaf. By construction of the slab boundaries, each leaf stores $O(B)$ segments in $O(1)$ blocks.

Consider a segment σ assigned to a node v of \mathcal{I} . Let s_ℓ and s_r be the children slabs of s_v where the left and right endpoints of σ lie. We call the segment $\sigma \cap s_\ell$ the *left subsegment* of σ , the segment $\sigma \cap s_r$ the *right subsegment* of σ and the rest of σ (which spans children slabs $s_{\ell+1}, \dots, s_{r-1}$) is its middle subsegment. See Figure 1 for an illustration. In this example, the left subsegment is $\sigma \cap s_5$, the right subsegment is $\sigma \cap s_2$, and the portion of σ in s_3 and s_4 is the middle subsegment.

Let S_v be the set of segments assigned to a node v of \mathcal{I} . To store segments of S_v , node v of \mathcal{I} contains the following secondary structures:

1. A multislab structure \mathcal{M} which stores the set of middle segments.
2. $B^{\epsilon'}$ left structures L_i , for $1 \leq i \leq B^{\epsilon'}$, storing the left (sub)segments of slab s_i .
3. $B^{\epsilon'}$ right structures R_i , for $1 \leq i \leq B^{\epsilon'}$, storing the right (sub)segments of slab s_i .

In addition, each internal node v contains an insertion buffer I_v and deletion buffer D_v , each storing up to B segments.

Construction and Space Usage. For every node v , the buffers I_v and D_v fit in $O(1)$ blocks, since they store at most B segments. By Theorems 2 and 3, a secondary structure storing K segments uses $O(K)$ space. Since each segment of S_v is stored in at most 3 secondary structures, overall secondary structures of v use $O(|S_v|)$ space. Thus each node v uses $O(|S_v|)$ space. We get that our data structure uses overall $O(\sum_{v \in \mathcal{I}} |S_v|) = O(N)$ space. The interval tree can be constructed in $O((N/B) \log_B N)$ I/Os. This can be done by sorting the segments by their endpoints' x -coordinates and then determining all slab boundaries to create a balanced interval tree. By Theorems 2 and 3, all secondary structures of a node v of \mathcal{I} can be constructed in $O((|S_v|/B) \log_B |S_v|)$ I/Os. Thus, all secondary structures of the tree can be constructed in $O((\sum_{v \in \mathcal{I}} |S_v|/B) \cdot \log_B N) = O((N/B) \log_B N)$ I/Os.

Queries. To answer a vertical ray-shooting query for a point p , we traverse the root-to-leaf path of \mathcal{I} based on the x -coordinate of p , while maintaining a segment σ (initialized to `null`) which is the answer to the query among segments assigned to nodes we have traversed so far. At each node v visited along this path, we first update buffers I_v and D_v by removing from both of them all segments (if any) of $I_v \cap D_v$. Then, we perform a vertical ray-shooting on the secondary structures of v ; in particular we ray-shoot on the multislab structure and the left and right structures L_i and R_i , for i such that the query point p is in slab s_i ¹. After checking the secondary structures, we update σ if a closer segment above p is found as a result. Next, we ray-shoot among segments stored in I_v and update σ if necessary. Finally, we determine which child v_i of v to visit, and flush any segments of D_v that are contained in the slab of v_i to D_{v_i} ; this way we make sure that information about deleted segments is updated throughout the root-to-leaf path and no deleted segment can be considered as an answer to the query. We then continue the process at v_i . Once a leaf node is reached, we simply compare the B segments it contains with p and return the closest segment above p among them and σ .

Bounding the query cost: Since any root-to-leaf path of \mathcal{I} has length $O((1/\epsilon') \cdot \log_B N)$, each secondary data structure supports ray-shooting queries in $O((1/\epsilon') \cdot \log_B N)$ I/Os (due to Theorems 2 and 3) and we check $O(1)$ secondary structures per node, we get that a query is answered in $O((1/\epsilon')^2 \cdot \log_B^2 N) = O(\log_B^2 N)$ I/Os. Note that in each node v of the root-to-leaf path visited, the operations involving I_v and D_v require $O(1)$ I/Os, thus they increase the total cost by at most a $O(1)$ factor.

¹ Minor detail: For each secondary structure considered, we first perform insertions/deletions of the corresponding segments from buffers I_v and D_v .

Insertions. To handle insertions, we use the insertion buffers stored in nodes of \mathcal{I} . When a new segment σ is inserted, we insert it in the insertion buffer of the root. Let v be an internal node with children $v_1, \dots, v_{B^{\epsilon'}}$. Whenever I_v becomes full, it is flushed. Segments of I_v that cross at least one slab boundary partitioning s_v are inserted in the secondary structures of v ; segments that are contained in the slab s_i of v_i are inserted in I_{v_i} , for $1 \leq i \leq B^{\epsilon'}$. In case I_v becomes full for some node v whose children are leaves, we insert those segments explicitly at the corresponding leaves. When a leaf becomes full, we restructure the tree using split operations on full nodes.

Bounding the insertion cost: We compute the amortized cost of an insertion by considering three components:

- (i) The cost for moving segments between insertion buffers. Whenever an insertion buffer I_v gets full, it forwards segments to the buffers of its $B^{\epsilon'}$ children performing $O(B^{\epsilon'})$ I/Os. Since a flushing occurs every B insertions in I_v , the amortized cost of such operations is $O(B^{\epsilon'}/B) = O(1/(B^{1-\epsilon'}))$. Each segment will move in at most $O((1/\epsilon') \log_B N)$ insertion buffers before it is inserted in the secondary structures of a node (or in a leaf). Thus the amortized cost for moving between buffers is $O((\log_B N)/(\epsilon' \cdot B^{1-\epsilon'}))$.
- (ii) The insertion cost in the secondary structures. By Theorems 2 and 3 we get that insertions in secondary structures require $O((\log_B N)/(\epsilon \cdot B^{1-2\epsilon'}))$ I/Os.
- (iii) The cost of restructuring the tree after insertions when a leaf becomes full. We show in Section 5 that the restructuring requires $O(\frac{\log_B N}{\epsilon' \cdot B^{1-\epsilon'}})$ amortized I/Os, by slightly modifying our primary interval tree data structure.

We conclude that our data structure supports insertions in amortized $O(\log_B N/(\epsilon' \cdot B^{1-2\epsilon'})) = O(\log_B N/B^{1-\epsilon})$ I/Os.

Deletions. To support deletions, we use the deletion buffers stored in all nodes of \mathcal{I} . To delete a segment σ , we first check whether σ is in the insertion buffer I_r of the root r and in that case we delete it; otherwise we store it in D_r . Similar to insertions, whenever D_v gets full for some internal node v with children $v_1, \dots, v_{B^{\epsilon'}}$, we flush D_v . The segments of D_v crossing at least one slab boundary partitioning s_v are deleted from the corresponding secondary structures associated with v ; the other segments of D_v are moved to buffers D_{v_i} ; in case a segment σ inserted in $D_{v_i} \cap I_{v_i}$, we delete it from both buffers. In case D_v becomes full for some v parent of leaves, we delete those segments explicitly from the corresponding leaves.

Bounding the deletion cost: The deletion cost has three components:

- (i) Moving segments between the deletion buffers. Using the same argument as for insertions, we get that this requires $O(\log_B N/(\epsilon' \cdot B^{1-\epsilon'}))$ I/Os, amortized.
- (ii) The cost of deletion in the secondary structures. By Theorems 2 and 3 we get that deletions in secondary structures require amortized $O(\log_B N/(\epsilon' \cdot B^{1-2\epsilon'}))$ I/Os.
- (iii) The cost of restructuring the tree. Every $N/2$ deletions, we rebuild the structure using $O((N/B) \log_B N)$ I/Os, to get an amortized restructuring cost of $O((\log_B N)/B)$ I/Os.

Overall deletions are supported in amortized $O(\log_B N/(\epsilon' \cdot B^{1-2\epsilon'})) = O(\log_B N/(B^{1-\epsilon}))$ I/Os.

3 Left and Right Structures

In this section we prove Theorem 2. Given K points all of whose right (left) endpoints lie on a single vertical line, we construct a data structure which answers vertical ray-shooting queries on those segments in $O(\log_B K)$ I/Os and supports insertions and deletions in $O((\log_B K)/B^{1-\epsilon})$ amortized I/Os for a constant $0 < \epsilon \leq 1/2$.

We describe the structure for the case where we are given a set \mathcal{L} of K segments whose right endpoints have the same x -coordinate (left structure)². The case where the left endpoints of the segments have the same x -coordinate (right structure) is completely symmetric. For a segment σ , we will refer to the y -coordinate of its right endpoint as the y -coordinate of σ . Conversely we define the x -coordinate of σ to be the x -coordinate of its left endpoint.

Total Order. We assume that the segments in \mathcal{L} are ordered according to their y -coordinates. We can always order the segments according to this total order in $O((K/B)\log_B K)$ I/Os.

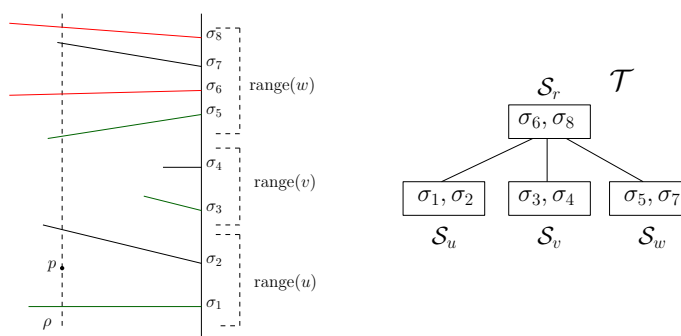
The Data Structure. We store all segments of \mathcal{L} in an augmented B^ϵ -tree \mathcal{T} which supports vertical ray-shooting queries, insertions and deletions. The degree of each node is between $B^\epsilon/2$ and B^ϵ , except the root which might have degree in the range $[2, B^\epsilon]$, and leaves store $\Theta(B)$ elements. For a node $v \in \mathcal{T}$, let \mathcal{T}_v be the subtree rooted at v . Since the segments are sorted according to their y -coordinates, each subtree \mathcal{T}_v corresponds to a range of y -coordinates, which we call the y -range of node v . Let v be an internal node of \mathcal{T} with children $v_1, \dots, v_{B^\epsilon}$. Node v stores the following information:

1. A buffer of segments \mathcal{S}_v of capacity B which contains segments in the y -range of v whose left endpoints have the smallest x -coordinates (i.e., segments that extend the farthest from the vertical line) and are not stored in any buffer \mathcal{S}_w for an ancestor w of v . In other words, \mathcal{T} together with segments of buffers \mathcal{S}_v form an external memory priority search tree [6].
2. An insertion buffer I_v and a deletion buffer D_v , each storing up to B segments.
3. A list \mathcal{M}_v that contains, for each child v_i , the segment with minimum x -coordinate stored in \mathcal{S}_{v_i} . We call this the *minimal* segment for child v_i .

The data structure satisfies the following invariants: For each node $v \in \mathcal{T}$, either $|\mathcal{S}_v| \geq B/2$ or if $|\mathcal{S}_v| < B/2$, then I_v and D_v are empty and all buffers stored in descendants v are empty. Also, for each node v , buffers \mathcal{S}_v, I_v and D_v are disjoint. Finally, for a leaf v , I_v and D_v are empty.

Construction and Space Usage. Overall buffers and lists of each node contain $O(B)$ segments, i.e. they can be stored in $O(1)$ blocks. Thus \mathcal{T} can be stored in $O(K/B)$ blocks, i.e. it requires $O(K)$ space. Construction of \mathcal{T} requires $O(\frac{K}{B}\log_B K)$ I/Os, since we need to sort all K segments according to their y -coordinates. If the segments are already sorted according to their y -coordinate, then \mathcal{T} can be created in $O(K/B)$ I/Os.

² Recall from Section 2 that we call left structures the ones storing the left subsegment of a segment σ , thus all subsegments stored in a left structure have the same x -coordinate of right endpoints.



■ **Figure 2** Example of the query algorithm in the left structure: Left column shows the segments stored in \mathcal{T} , the query point p and the vertical ray ρ emanating from p . Right column shows buffers \mathcal{S} of the nodes of \mathcal{T} . Red segments are stored in the root. For nodes u, v, w , the green segment is their minimal segment, i.e., the one stored in list \mathcal{M}_r . By ray-shooting on ρ among green segments, the first segment hit upwards is σ_5 , which is stored in \mathcal{S}_w , thus we set $v_+ = w$. Note that σ_2 (the correct answer for the query) is not stored in \mathcal{S}_w , i.e., maintaining only v_+ produces an incorrect answer. Thus, our algorithm ray-shoots downwards as well, hitting σ_1 , which is stored in u , and setting $v_- = u$. Then, by ray-shooting on ρ among \mathcal{S}_u and \mathcal{S}_w , the first segment we hit upwards of p is σ_2 .

Queries in the static structure.

To get a feel for how our structure supports queries, we first show how to perform queries in the static case, i.e., assuming there are no insertions and deletions and all buffers I_v and D_v are empty. Later we will give a precise description of performing queries in the fully dynamic structure.

Let ρ^+ be the ray emanating from p in the $(+y)$ direction and ρ^- the ray emanating from p in the $(-y)$ direction. We query the structure by finding the first segment hit by both ρ^+ and ρ^- . We keep two pointers, v_+ and v_- , initialized at the root. We also keep the closest segments σ_+ and σ_- seen so far in the $(+y)$ and $(-y)$ direction respectively (initialized to $+\infty$ and $-\infty$). At each step, we update both v_+ and v_- to move from a node of depth i to a node of depth $i + 1$. While at level i , v_- and v_+ might coincide, or one of them might be undefined (set to null).

We now describe the query algorithm. We start at the root of \mathcal{T} and advance down, while updating v_+ , v_- , and σ_+, σ_- . When at depth i , we find the first segment σ_i hit by ρ^+ among \mathcal{S}_{v_-} and \mathcal{S}_{v_+} and update σ_+ if necessary (i.e. if σ_i is the first segment hit by ρ^+ among all segments seen so far). Similarly, we ray-shoot on ρ^- among \mathcal{S}_{v_-} and \mathcal{S}_{v_+} and update σ_- if necessary. To determine in which nodes of depth $i + 1$ to continue the search, we ray-shoot on ρ^+ among \mathcal{M}_{v_-} and \mathcal{M}_{v_+} and also ray-shoot on ρ^- among \mathcal{M}_{v_-} and \mathcal{M}_{v_+} (i.e., all minimal segments of children of v_- and v_+). Let σ_{m+} be the first segment in $\mathcal{M}_{v_+} \cup \mathcal{M}_{v_-}$ hit by ρ^+ (if such a segment exists) and v_s be the node containing σ_{m+} (if σ_{m+} exists). If the y -range of v_s is higher than the y -coordinate of σ_+ or if σ_{m+} does not exist, we leave v_+ undefined for level $i + 1$. Otherwise, we set $v_+ = v_s$. Similarly, call σ_{m-} the first minimal segment of $\mathcal{M}_{v_+} \cup \mathcal{M}_{v_-}$ hit by ρ^- and v_p be the node containing σ_{m-} (if such a segment exists). If the y -range of v_p is lower than the y -coordinate of σ_- or if σ_{m-} does not exist, we leave v_- undefined for level $i + 1$. Otherwise we set $v_- = v_p$.

If both v_+ and v_- are undefined for the next level $i + 1$, we stop the procedure and output σ_+ as the result to the vertical ray-shooting query. Otherwise we repeat the same procedure in the next level. When we reach a leaf level, we find the first segment hit by ρ^+ among \mathcal{S}_{v_-} and \mathcal{S}_{v_+} , update σ_+ if necessary, and output σ_+ as the result of the query.

Remark: The reader might wonder why we answer vertical ray-shooting queries in both directions and keep two pointers v_- and v_+ . Isn't it sufficient to answer queries in one direction and keep one pointer at each step? Figure 2 shows an example where this is not true and maintaining only the v_+ pointer would result in an incorrect answer.

The formal proof of correctness of this query algorithm is deferred to Appendix A.

Bounding the query cost: To count the cost, observe that in each step we move down the tree by one level and perform operations that require $O(1)$ I/Os, as we check $O(B)$ segments stored in the current nodes v_- and v_+ . Since the height of the tree is $O((1/\epsilon) \log_B K)$, a query is answered in $O((1/\epsilon) \log_B K) = O(\log_B K)$ I/Os.

Insertions. Assume we want to insert a segment σ into the left structure \mathcal{L} . If the x -value of σ is smaller than the maximum x -value of a segment stored in the buffer of the root \mathcal{S}_r , we insert σ into \mathcal{S}_r . Otherwise we store σ in the insertion buffer of the root I_r . Note that insertion of σ in \mathcal{S}_r might cause \mathcal{S}_r to overflow (i.e., $|\mathcal{S}_r| = B + 1$); in that case we move the segment of \mathcal{S}_r with the maximum x -value into the insertion buffer of the root I_r .

Let v be an internal node with children $v_1, \dots, v_{B^\epsilon}$. Whenever the insertion buffer I_v becomes full, we *flush* it, moving the segments to buffers of the corresponding children. For a segment σ that should be stored in child v_i , we repeat the same procedure as in the root: Check whether σ has smaller x -value than the maximum x -value of a segment stored in \mathcal{S}_{v_i} and if yes, store σ in \mathcal{S}_{v_i} , otherwise store it in I_{v_i} . If \mathcal{S}_{v_i} overflows, we move its last segment (i.e. the one with maximum x -value) into I_{v_i} . Also, if σ gets stored in \mathcal{S}_{v_i} and its x -value is smaller than all previous segments of \mathcal{S}_{v_i} , we update the minimal segment of v_i , \mathcal{M}_v .

When \mathcal{S}_v overflows for some leaf v , we split v into two leaves v_1 and v_2 , as in standard B -trees. Note that this might cause recursive splits of nodes at greater height.

Bounding the insertion cost: To flush a buffer I_v and forward segments to buffers \mathcal{S}_{v_i} and I_{v_i} , for $1 \leq i \leq B^\epsilon$ we perform $O(B^\epsilon)$ I/Os. Since I_v becomes full after at least B insertions, the amortized cost of moving a segment from I_v to buffers of a child of v is $O(B^\epsilon/B) = O(1/B^{1-\epsilon})$. Each inserted segment moves between buffers in a root-to-leaf path of length $O((1/\epsilon) \log_B K)$, thus the total amortized cost for moves between buffers is $O(\log_B K / (\epsilon \cdot B^{1-\epsilon}))$ I/Os. The restructuring of \mathcal{T} due to splitting nodes requires amortized $O(1/B)$ I/Os, as in standard B -trees. Thus, insertions are supported in $O(\log_B K / (\epsilon \cdot B^{1-\epsilon}))$ amortized I/Os.

Deletions. To delete a segment σ , we first check whether it is stored in the buffers of the root \mathcal{S}_r or I_r ; in this case we delete it. Otherwise, we insert σ in the deletion buffer of the root D_r .

Let v be an internal node with children $v_1, \dots, v_{B^\epsilon}$. Whenever D_v becomes full we flush it and move the segments to the corresponding children and repeat the same procedure: For a segment σ which moves to child v_i , we check whether it is stored in \mathcal{S}_{v_i} or I_{v_i} : if yes, we delete it and update the minimal segment of v_i in \mathcal{M}_v if necessary. Otherwise, we store σ in the deletion buffer D_{v_i} . If segment buffer \mathcal{S}_v underflows (i.e., $|\mathcal{S}_v| < B/2$), we refill it using segments stored in buffers \mathcal{S}_{v_i} ; the segments moved to \mathcal{S}_v are deleted from \mathcal{S}_{v_i} and all necessary updates in \mathcal{M}_v are performed. This might cause underflowing segment buffers \mathcal{S}_{v_i} for children of v_i ; we handle those in the same way. In case all buffers \mathcal{S}_{v_i} become empty and $|\mathcal{S}_v| < B$, we move the segments from I_v to \mathcal{S}_v until either $|\mathcal{S}_v| = B$ or $|I_v| = 0$.

Bounding the deletion cost: Deletion cost consists of three components:

- (i) Cost for moving segments between buffers: Using the same analysis as for insertions we get that this requires $O(\log_B K / (\epsilon \cdot B^{1-\epsilon}))$ amortized I/Os.

- (ii) Cost due to refilling of buffers \mathcal{S}_v : For a node v with children v_i , while refilling buffer \mathcal{S}_v from \mathcal{S}_{v_i} we perform $O(B^\epsilon)$ I/Os and we move $\Theta(B)$ segments one level higher. Thus the amortized cost of moving a segment up by one level is $O(1/B^{1-\epsilon})$. Since the tree has height $O((1/\epsilon) \cdot \log_B K)$, over a sequence of K deletions the total number of moves of segments by one level is $O((1/\epsilon) \cdot K \cdot \log_B K)$. Thus the total cost due to refilling is at most $O((1/\epsilon B^{1-\epsilon})K \cdot \log_B K)$, which implies that the amortized cost is $O(\log_B K / (\epsilon \cdot B^{1-\epsilon}))$.

A corner case that we did not take into account above is when the total number of segments stored in buffers \mathcal{S}_{v_i} are less than $B/2$. In this case it is not valid that the amortized cost of updating \mathcal{S}_v is $O(B^\epsilon/B)$. To take care of this, we use a simple amortization trick: we double charge all I/Os performed relating to insertions. This way, for each buffer \mathcal{S}_{v_i} there is a saved I/O from the time when segments move from I_v to node v_i . We use this additional saved I/O when \mathcal{S}_{v_i} gets emptied due to the refilling of \mathcal{S}_v .

- (iii) Restructuring requires $O(\frac{\log_B K}{B})$ amortized I/Os, by rebuilding the structure after $K/2$ deletions.

Overall, the amortized deletion cost is $O(\log_B K / (\epsilon \cdot B^{1-\epsilon})) = O(\log_B K / B^{1-\epsilon})$ I/Os.

Queries in the dynamic structure. We now describe how to extend our query algorithm to the dynamic case. In order to ensure that all nodes visited are up-to-date and we do not miss any updates in the insertion/deletion buffers, when moving a pointer from a node u to its child v_i , we flush any deletes in D_u to v_i , i.e. delete segments of D_u that are stored in \mathcal{S}_{v_i} , store the other segments in D_{v_i} and update \mathcal{M}_u if necessary. We then delete any segments found in both I_{v_i} and D_{v_i} . Finally, we compare segments in I_{v_i} with σ_+ (recall this is the first segment hit by ρ^+ among segments considered so far) and, if any segment in I_{v_i} would be hit by ρ^+ before σ_+ we replace σ_+ with it. Clearly this increases the total cost by at most a $O(1)$ factor compared to the static case, thus the query cost is $O((1/\epsilon) \log_B K)$ I/Os.

4 Multislab Structure

In this section we prove Theorem 3. Assume that we are given a set of K non-intersecting segments with endpoints on at most $B^{\epsilon/2} + 1$ vertical lines $l_1, \dots, l_{B^{\epsilon/2}+1}$, for some constant $0 < \epsilon \leq 1/2$. We show that those segments can be stored in a data structure which uses $O(K)$ space, supports vertical ray-shooting queries in $O(\log_B K)$ I/Os, and updates in $O(\log_B K / B^{1-\epsilon})$ amortized I/Os, for $0 < \epsilon \leq 1/2$. This data structure can be constructed in $O((K/B) \log_B K)$ I/Os. We call this data structure a *multislab* structure.

For notational convenience we set $\epsilon' = \epsilon/2$. This way endpoints of the segments lie on at most $B^{\epsilon'} + 1$ vertical lines $l_1, \dots, l_{B^{\epsilon'}+1}$. For $1 \leq i \leq B^{\epsilon'}$, let s_i denote the vertical slab defined by vertical lines l_i and l_{i+1} . We will show that queries are supported in $O(\log_B K)$ I/Os and updates in $O((\log_B K) / B^{1-2\epsilon'})$ I/Os. Theorem 3 then follows.

Total Order. In order to implement the multislab structure we need to maintain an ordering of the segments based on their y -coordinates. Using standard approaches (see e.g. [5, 7]) we can define a partial order for segments that can be intersected by a vertical line. Arge et. al. [8] showed how to extend a partial order into a total order on K segments (not necessarily all intersecting the same vertical line) in $O((K/B) \log_{M/B} \frac{K}{B}) = O((K/B) \log_B K)$ I/Os. We use this total order to create our multislab structure.

The Data Structure. We store the ordered segments in an augmented B-tree \mathcal{T} which supports queries, insertions and deletions. The degree of each node is between $B^{\epsilon'}/2$ and $B^{\epsilon'}$, except the root which might have degree in the range $[2, B^{\epsilon'}]$. Leaves store $\Theta(B)$ elements. For a node $v \in \mathcal{T}$, let \mathcal{T}_v be the subtree rooted at v . Let $v_1, \dots, v_{B^{\epsilon'}}$ be the children of an internal node v . Node v stores the following information:

1. A buffer \mathcal{S}_v of capacity B which contains the highest (according to the total order) segments stored in \mathcal{T}_v which are not stored in any buffer \mathcal{S}_w for an ancestor w of v . In other words, \mathcal{T} together with segments of buffers \mathcal{S}_v form an external memory priority search tree [6].
2. An insertion buffer I_v and a deletion buffer D_v , both storing up to B segments.
3. A list L_v which contains, for each slab s_i , $1 \leq i \leq B^{\epsilon'}$, and each child v_j , $1 \leq j \leq B^{\epsilon'}$, the highest segment (according to the total order) $t_{i,j}$ crossing slab s_i stored in \mathcal{T}_{v_j} .

The data structure satisfies the following invariants: i) for each node $v \in \mathcal{T}$, either $|\mathcal{S}_v| \geq B/2$ or if $|\mathcal{S}_v| < B/2$, then I_v and D_v are empty and all buffers of descendants w of v are empty, ii) for each node v , buffers \mathcal{S}_v, I_v and D_v are disjoint, and iii) for every leaf v , I_v and D_v are empty.

Construction and Space Usage. Overall buffers of each node contain $O(B)$ segments and list L_v contains at most $B^{2\epsilon'} = O(B)$ segments, i.e., they can be stored in $O(1)$ blocks. Thus \mathcal{T} can be stored in $O(K/B)$ blocks, i.e. it requires $O(K)$ space. The structure can be constructed in $O(\frac{K}{B} \log_B K)$ I/Os. If segments are already sorted according to a total order, construction requires $O(K/B)$ I/Os.

Insertions. To insert a new segment σ we need to determine its position in the total order. Clearly, we can not afford to produce a new total order from scratch, as this costs $O((K/B) \log_B K)$ I/Os. Thus, we break σ into at most $B^{\epsilon'}$ *unit segments*, where each segment crosses exactly one slab. In particular, if σ crosses slabs s_ℓ, \dots, s_r , we break it into unit segments $\sigma_\ell, \dots, \sigma_r$, where segment σ_i crosses slab s_i . We call all such unit segments stored in \mathcal{T} *new segments*. The rest of the segments stored in \mathcal{T} are called the *old segments* of \mathcal{T} . Now we can easily update the total order: segment σ_i needs to be compared only with segments crossing slab s_i ; if σ_p and σ_s are the predecessor and successor of σ_i within slab s_i , we locate σ_i in an arbitrary position between σ_p and σ_s in the total order. This way a valid total order is always maintained.

We now describe the insertion algorithm. When segment σ needs to be inserted, we first break it into unit segments $\sigma_\ell, \dots, \sigma_r$. For each segment σ_j , $\ell \leq j \leq r$, we first check whether it should be inserted in the buffer \mathcal{S}_r of the root: if this is the case we store it there; otherwise we store it in the insertion buffer of the root I_r . In case \mathcal{S}_r overflows (i.e. $|\mathcal{S}_r| = B + 1$) we move its last segment (according to the total order) to I_r . Let v be an internal node with children $v_1, \dots, v_{B^{\epsilon'}}$. Each time I_v becomes full, we *flush* it and move the segments to its children v_i , for $1 \leq i \leq B^{\epsilon'}$. For a segment moving from v to v_i , we first check whether it is greater (according to the total order) than the minimum segment stored in \mathcal{S}_{v_i} and if so we store it in \mathcal{S}_{v_i} ; otherwise we store it in buffer I_{v_i} . In case \mathcal{S}_{v_i} overflows (i.e. $|\mathcal{S}_{v_i}| = B + 1$) we move its last segment to I_{v_i} . Also we update information in list L_v if necessary. In case I_{v_i} becomes full, we repeat the same procedure recursively.

When \mathcal{S}_v overflows for some leaf v , we split v into two leaves v_1 and v_2 , as in standard B-trees. Note that this might cause recursive splits of nodes at greater height.

Bounding the insertion cost: To flush a buffer I_v and move segments to buffers of child nodes \mathcal{S}_{v_i} and I_{v_i} , we need to perform $O(B^{\epsilon'})$ I/Os. Since each segment breaks into at most $B^{\epsilon'}$ unit segments, a buffer of size B becomes full after at least $B/B^{\epsilon'} = B^{1-\epsilon'}$ insertions. Thus the amortized cost of moving a segment from a buffer of depth i to depth $i + 1$ is $O(B^{\epsilon'}/B^{1-\epsilon'}) = O(1/B^{1-2\epsilon'})$. Since each segment will be eventually stored in a node of depth $O((1/\epsilon') \cdot \log_B K)$, the amortized cost until it gets inserted is $O(\log_B K/(\epsilon' \cdot B^{1-2\epsilon'}))$. The restructuring of \mathcal{T} due to splitting full nodes requires amortized $O(1)$ I/Os, as in standard B-trees. Overall insertions require $O(\log_B K/(\epsilon \cdot B^{1-2\epsilon'})) = O((\log_B K)/B^{1-\epsilon})$ amortized I/Os.

Linear space usage: To avoid increases in space usage due to unit segments, whenever there are $K/B^{\epsilon'}$ new segments, we rebuild the structure. This way the space used is $O(K + (K/B^{\epsilon'}) \cdot B^{\epsilon'}) = O(K)$. This rebuilding requires $O((K/B) \log_B K)$ I/Os, i.e., $O(\log_B K/B^{1-\epsilon'})$ amortized I/Os, thus it does not violate the insertion time bound.

Deletions. The process of deleting a segment, σ , is similar to insertion: we break σ into at most $B^{\epsilon'}$ unit segments $\sigma_\ell, \dots, \sigma_r$ where s_ℓ and s_r are the leftmost and rightmost slabs spanned by σ and apply the deletion procedure for each of those unit segments separately.

The deletion algorithm for a unit segment σ_i is analogous to the one of the left (right) structure of Section 3. For completeness we describe it here. To delete a unit segment σ_i , we first check whether it is stored in the buffers of the root \mathcal{S}_r or I_r ; in this case we delete it. Otherwise, we insert σ_i in the deletion buffer of the root D_r . Let v be an internal node with children $v_1, \dots, v_{B^{\epsilon'}}$. Whenever D_v becomes full we flush it and forward the segments to the corresponding children and repeat the same procedure: For a segment σ which moves to child v_i , we check whether it is stored in \mathcal{S}_{v_i} or I_{v_i} and if this is the case, we delete it and update list L_v if necessary. Otherwise, we store σ_i in the deletion buffer D_{v_i} .

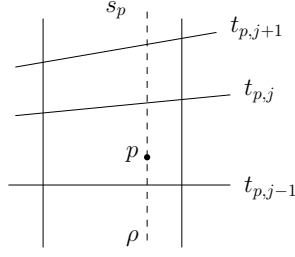
In case segment buffer \mathcal{S}_v underflows (i.e., $|\mathcal{S}_v| < B/2$), we refill it using segments from buffers \mathcal{S}_{v_i} ; segments moved to \mathcal{S}_v are deleted from \mathcal{S}_{v_i} and L_v gets updated (if needed). This might cause underflowing segment buffers \mathcal{S}_{v_i} ; we handle those in the same way. In case all buffers \mathcal{S}_{v_i} become empty and $|\mathcal{S}_v| < B$, we move to \mathcal{S}_v the segments from I_v until either $|\mathcal{S}_v| = B$ or $I_v = 0$. After $K/B^{\epsilon'}$ deletions we rebuild our data structure.

Remark: Note that here we split all segments σ into unit segments $\sigma_\ell, \dots, \sigma_r$. However, the old segments σ are not unit segments and are stored manually in the data structure. However this does not affect our algorithm: whenever the first unit segment σ_i which is a part of σ reaches the node v such that $\sigma \in \mathcal{S}_v$, we delete σ from \mathcal{S}_v and remove σ_i from deletion buffers. The remaining segments σ_j will eventually reach node v and realize that σ is already deleted from \mathcal{S}_v ; at this point σ_j gets deleted.

Bounding the deletion cost: The analysis of the deletion cost is identical to the analysis of deletions in the structure of Section 3. Since each segment breaks into at most $B^{\epsilon'}$ unit segments, we get an amortized deletion cost of $O(\log_B K/B^{1-2\epsilon'}) = O(\log_B K/B^{1-\epsilon})$.

Linear space usage: Similar to insertions, we need to make sure that the total space used is not increasing asymptotically due to the use of at most $B^{\epsilon'}$ unit segments in deletion buffers for each deleted segment σ . The total capacity of deletion buffers is $O(K)$. Since we rebuild the structure after $K/B^{\epsilon'}$ deletions, there are at most $O(K)$ segments stored in deletion buffers, i.e., deletion buffers never get totally full and total space used is $O(K)$

Queries. Let p be the query point and ρ^+ be the the vertical ray emanating from p in the $(+y)$ direction. Let also s_p be the slab containing p . We can find s_p in $O(1)$ I/Os by storing all slab boundaries in a block. We perform a root-to-leaf search and we keep the first segment



■ **Figure 3** Vertical ray-shooting queries in the multislab structure: Query point p is in slab s_p . ρ is the vertical ray emanating from p . While being at node v of \mathcal{T} , to decide in which child to continue our search we examine all minimal segments $t_{p,1}, \dots, t_{p,B^{\epsilon'}}$ stored in list L_v . Among them, the first one hit by ρ is $t_{p,j}$. Thus the search continues at child v_j of v .

σ hit by ρ^+ among segments seen so far. While visiting a node v we do the following: (i) perform a vertical ray-shooting query from p among segments stored in buffers \mathcal{S}_v and I_v , and update σ if necessary (ii) move to the child v_i which contains the successor segment $t_{p,j}$ of p in list L_v (see Figure 3) and (iii) find in I_v (resp. D_v) the segments crossing slab s_p and should be stored (according to the total order) in \mathcal{T}_{v_i} and move them to \mathcal{S}_{v_i} or I_{v_i} (resp. delete them from \mathcal{S}_{v_i} or store it in D_{v_i}). If a segment inserted in D_{v_i} is also stored in I_{v_i} , we delete it from both buffers. Once we reach a leaf v , we first delete from \mathcal{S}_v the segments that are in the deletion buffer of its parent and then we perform ray-shooting query among the segments stored in \mathcal{S}_v and update σ if necessary.

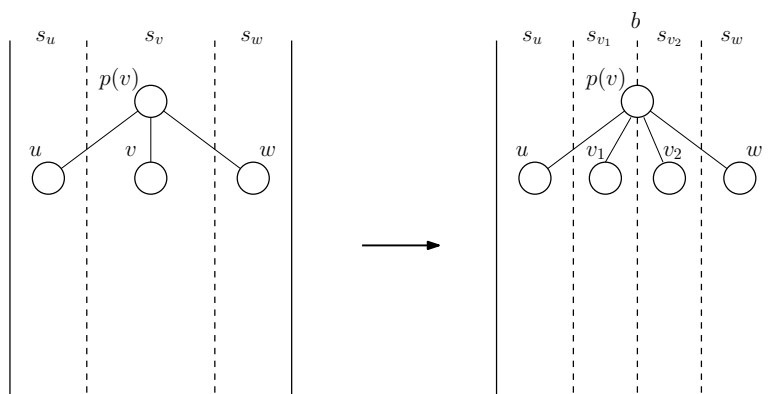
Bounding the query cost: Since we follow a root-to-leaf path, and at each level we need to perform $O(1)$ I/Os, a ray-shooting query is answered in $O((1/\epsilon') \cdot \log_B K)$ I/Os.

5 Counting the Restructuring Cost

In Section 2 we proved the Theorem 1 (query and update bounds of the overall structure) without taking into account the cost of restructuring the interval tree \mathcal{I} due to insertions that cause leaves to become full. In this section we show that Theorem 1 holds while taking into account the restructuring of \mathcal{I} as well.

When a leaf becomes full we need to split it. This split in turn might cause the split of the parent and possibly continue up the tree, thus causing some part of the tree \mathcal{I} to need rebalancing. While rebalancing, we need to perform updates in the secondary structures so that they are adjusted with the updated nodes of the interval tree \mathcal{I} . In this section, we show that we can slightly modify our data structure such that all updates in secondary structures can be performed in $O(\frac{\log_B N}{B^{1-\epsilon}})$ amortized I/Os. This implies that Theorem 1 holds.

Our Approach. We use a variant of the weight-balanced B^ϵ -tree of [9]. Each leaf stores at most B segment endpoints. Let v be a node at height $h - 1$ with parent $p(v)$. Node $p(v)$ stores $w_v = \Theta(B \cdot B^{\epsilon h})$ elements in its subtree $\mathcal{I}_{p(v)}$. We will show that if node v splits, then we can perform all updates needed in the secondary structures in $O(w_v/B^{1-\epsilon})$ I/Os. This implies that a split requires amortized $O(1/B^{1-\epsilon})$ I/Os, since after a restructuring, there should be at least $\Omega(w_v)$ insertions in $\mathcal{I}_{p(v)}$ until the next split is needed. Since each insertion can cause $O(\log_B N)$ splits, we get an amortized restructuring cost of $O(\frac{\log_B N}{B^{1-\epsilon}})$ I/Os for insertion.

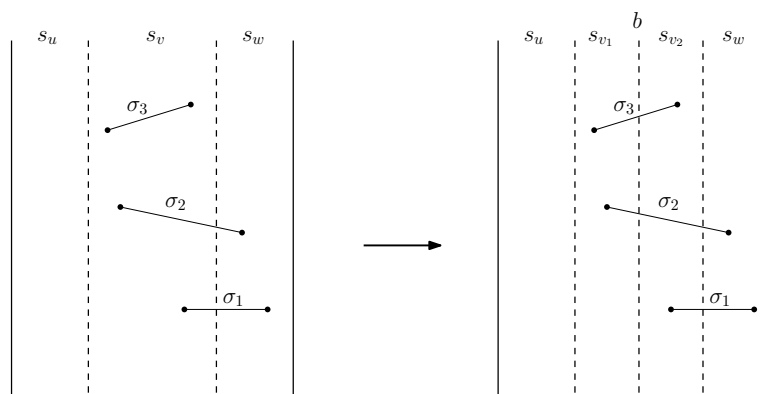


■ **Figure 4** Splitting a node v into v_1 and v_2 : slab s_v is divided into slabs s_{v_1} and s_{v_2} with boundary b .

Splitting a node. Node v splits into two new nodes v_1 and v_2 . The slab s_v of v is divided into two slabs s_{v_1}, s_{v_2} with slab boundary b ; see Figure 4. To capture this change and update our data structure, we need to perform updates in the secondary structures of $p(v)$ and construct the secondary structures for v_1, v_2 . We describe these updates in detail and show that they can be performed in $O(w_v/B^{1-\epsilon})$ I/Os. In our analysis we use the fact that all secondary structures (multislab and left/right) storing K segments can be scanned in $O(K/B)$ I/Os.

Updates in secondary structures of $p(v)$. We begin with the construction of left/right structures for v_1 and v_2 using the previous left/right structures for v . We describe the creation of left structures L_{v_1} and L_{v_2} for v_1 and v_2 , respectively, and the right structures are symmetric. Segments that were stored in L_v and do not cross b (like segment σ_1 in Figure 5) are stored in L_{v_2} ; segments of L_v that cross b (see segment σ_2 in Figure 5) are stored in L_{v_1} . To identify if a segment is stored in L_{v_1} or L_{v_2} we just need to scan L_v , which takes $O(w_v/B)$ I/Os. Moreover, there are some additional segments that need to be stored in left/right structures of $p(v)$: the segments that are strictly inside the slab of v (i.e. they were stored in secondary structures of v) and cross b ; see e.g. segment σ_3 in Figure 5. For those segments, their left subsegments are stored in L_{v_1} and their right subsegments in R_{v_2} . To find such segments we need to scan all secondary structures stored at v . Since each secondary structure can be scanned in $O(w_v/B)$ I/Os and there are $O(B^\epsilon)$ structures stored in each node, all this takes $O((w_v/B) \cdot B^\epsilon) = O(w_v/B^{1-\epsilon})$ I/Os.

We now proceed to the updates of the multislab structure of $p(v)$. Here, we just need to add some segments to the previous multislab structure. The new segments are the segments of L_v that cross b which are not already stored in the multislab (and symmetrically, the segments of R_v that cross b and are not yet in the multislab). For an example, see segment σ_2 in Figure 5; before it was not stored in the multislab and now we store its middle subsegment. Note that the middle subsegment is a unit segment (i.e. crosses exactly one slab) thus we don't need to compute a new total order; we can find its position in the total order by comparing it only with segments that cross slab s_{v_2} . All those segments that need to be added can be found by scanning L_v and R_v in $O(w_v/B)$ I/Os. Insertions in the multislab of $p(v)$ require $O((\log_B w_v)/B^{1-\epsilon}) = O(w_v/B)$ I/Os. Also, all information stored in nodes of the multislab structure can be updated in $O(w_v/B)$ I/Os. Overall, all updates in the multislab structure of v are performed in $O(w_v/B)$ I/Os.



■ **Figure 5** Example of segments that get stored in different secondary structures after a split. Segment σ_1 was stored in L_v and, after the split, gets stored in L_{v_2} . Segment σ_2 was stored in L_v ; following the split its left subsegment is stored in L_{v_1} and its middle subsegment in the multislab structure of $p(v)$. Segment σ_3 was previously stored in secondary structures of v , and after the split it should be stored in structures L_{v_1} and R_{v_2} of $p(v)$.

Construct secondary structures for v_1 and v_2 . The left and right structures for each child slab of v_1 and v_2 will be based on the left/right structure of the same slab in v just by removing the segments that cross b (which are assigned to $p(v)$ as we explained above). Similarly, segments that cross b are excluded from the multislab structure.

We start with the construction of left/right structures of v_1 and v_2 . We describe the left and the right is symmetric. For each slab s_k of v , $1 \leq k \leq B^\epsilon$ we scan the left list L_k ; the segments that do not cross b remain in L_k and the others are deleted. All this takes $O((w_v/B) \cdot B^\epsilon) = O(w_v/B^{1-\epsilon})$ I/Os.

Finally we create the multislab structures for v_1 and v_2 . Again, we need to scan the multislab of v and delete the segments that cross b , which takes $O(w_v/B)$ I/Os. Then we need to build the multislabs of v_1 and v_2 out of the remaining segments. Since all segments are already sorted according to a total order, this can be done in $O(w_v/B)$ I/Os.

References

- 1 Pankaj K. Agarwal, Lars Arge, Gerth Stølting Brodal, and Jeffrey Scott Vitter. I/O-Efficient Dynamic Point Location in Monotone Planar Subdivisions. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 11–20, 1999. URL: <http://dl.acm.org/citation.cfm?id=314500.314525>.
- 2 Alok Aggarwal and Jeffrey Scott Vitter. The Input/Output Complexity of Sorting and Related Problems. *Commun. ACM*, 31(9):1116–1127, 1988. doi:10.1145/48529.48535.
- 3 Lars Arge. The Buffer Tree: A Technique for Designing Batched External Data Structures. *Algorithmica*, 37(1):1–24, 2003. doi:10.1007/s00453-003-1021-x.
- 4 Lars Arge, Gerth Stølting Brodal, and Loukas Georgiadis. Improved Dynamic Planar Point Location. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 305–314, 2006. doi:10.1109/FOCS.2006.40.
- 5 Lars Arge, Gerth Stølting Brodal, and S. Srinivasa Rao. External Memory Planar Point Location with Logarithmic Updates. *Algorithmica*, 63(1-2):457–475, 2012. doi:10.1007/s00453-011-9541-2.
- 6 Lars Arge, Vasilis Samoladas, and Jeffrey Scott Vitter. On Two-Dimensional Indexability and Optimal Range Search Indexing. In *PODS*, pages 346–357. ACM Press, 1999.

- 7 Lars Arge and Jan Vahrenhold. I/O-efficient dynamic planar point location. *Comput. Geom.*, 29(2):147–162, 2004. doi:10.1016/j.comgeo.2003.04.001.
- 8 Lars Arge, Darren Erik Vengroff, and Jeffrey Scott Vitter. External-Memory Algorithms for Processing Line Segments in Geographic Information Systems. *Algorithmica*, 47(1):1–25, 2007. doi:10.1007/s00453-006-1208-z.
- 9 Lars Arge and Jeffrey Scott Vitter. Optimal External Memory Interval Management. *SIAM J. Comput.*, 32(6):1488–1508, 2003. doi:10.1137/S009753970240481X.
- 10 Hanna Baumgarten, Hermann Jung, and Kurt Mehlhorn. Dynamic Point Location in General Subdivisions. *J. Algorithms*, 17(3):342–380, 1994. doi:10.1006/jagm.1994.1040.
- 11 Rudolf Bayer and Edward M. McCreight. Organization and Maintenance of Large Ordered Indices. *Acta Inf.*, 1:173–189, 1972. doi:10.1007/BF00288683.
- 12 Jon Louis Bentley. Decomposable Searching Problems. *Inf. Process. Lett.*, 8(5):244–251, 1979. doi:10.1016/0020-0190(79)90117-0.
- 13 Gerth Stølting Brodal. External Memory Three-Sided Range Reporting and Top-k Queries with Sublogarithmic Updates. In *33rd Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 23:1–23:14, 2016. doi:10.4230/LIPIcs.STACS.2016.23.
- 14 Gerth Stølting Brodal and Rolf Fagerberg. Lower bounds for external memory dictionaries. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 546–554, 2003. URL: <http://dl.acm.org/citation.cfm?id=644108.644201>.
- 15 Timothy M. Chan and Yakov Nekrich. Towards an Optimal Method for Dynamic Planar Point Location. In *IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 390–409, 2015. doi:10.1109/FOCS.2015.31.
- 16 Bernard Chazelle. Computational Geometry for the Gourmet: Old Fare and New Dishes. In *Automata, Languages and Programming, 18th International Colloquium (ICALP)*, pages 686–696, 1991. doi:10.1007/3-540-54233-7_174.
- 17 Bernard Chazelle. Computational geometry: a retrospective. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing (STOC)*, pages 75–94, 1994. doi:10.1145/195058.195110.
- 18 Siu-Wing Cheng and Ravi Janardan. New Results on Dynamic Planar Point Location. *SIAM J. Comput.*, 21(5):972–999, 1992. doi:10.1137/0221057.
- 19 Yi-Jen Chiang and Roberto Tamassia. Dynamization of the trapezoid method for planar point location in monotone subdivisions. *Int. J. Comput. Geometry Appl.*, 2(3):311–333, 1992.
- 20 Herbert Edelsbrunner and Hermann A. Maurer. On the Intersection of Orthogonal Objects. *Inf. Process. Lett.*, 13(4/5):177–181, 1981. doi:10.1016/0020-0190(81)90053-3.
- 21 Michael T. Goodrich and Roberto Tamassia. Dynamic Trees and Dynamic Point Location. *SIAM J. Comput.*, 28(2):612–636, 1998.
- 22 J. Ian Munro and Yakov Nekrich. Dynamic Planar Point Location in External Memory. In *SoCG*, volume 129 of *LIPIcs*, pages 52:1–52:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019.
- 23 Eunjin Oh and Hee-Kap Ahn. Point Location in Dynamic Planar Subdivisions. In *34th International Symposium on Computational Geometry, SoCG 2018*, pages 63:1–63:14, 2018. doi:10.4230/LIPIcs.SoCG.2018.63.
- 24 Jack Snoeyink. Point Location. In *Handbook of Discrete and Computational Geometry, Second Edition*, pages 767–785. Chapman & Hall/CRC, 2004. doi:10.1201/9781420035315.pt4.
- 25 Jeffrey Scott Vitter. External memory algorithms and data structures. *ACM Comput. Surv.*, 33(2):209–271, 2001. doi:10.1145/384192.384193.
- 26 Jeffrey Scott Vitter. Algorithms and Data Structures for External Memory. *Foundations and Trends in Theoretical Computer Science*, 2(4):305–474, 2006. doi:10.1561/04000000014.

A

 Queries in the Left and Right Structures

In this Section we give further details on the left (right) structure which were omitted from Section 3.

Queries. We begin with the queries and we show the correctness of the query algorithm of the static left (right) structure.

Correctness: The correctness of the query algorithm follows from the next lemma. For a node $v \in \mathcal{T}$ let S_v be the set of segments stored in buffers \mathcal{S} in \mathcal{T}_v .

► **Lemma 4.** *Assume that at the end of the i th step of the query algorithm, either v_+ or v_- is defined. Then σ_+ is the first segment hit by ρ^+ among the segments of $\mathcal{L} - (S_{v_-} \cup S_{v_+})$.*

Proof. We prove the lemma by induction.

Induction Base: At the end of the first step, v_+ and v_- are children of the root r and σ_+ is the first segment hit by ρ^+ among all segments stored at the root (in \mathcal{S}_r and \mathcal{M}_r). By definition of $v_s = v_+$, for any child of the root v with higher y -range than v_+ , σ_+ is below all segments of S_v . Similarly, for any child of the root v' with smaller y -range than v_- (if v_- exists), there is no segment in $S_{v'}$ hit by ρ^+ (since there exists a segment in \mathcal{S}_{v_-} hit by ρ^-). Finally, for any child v'' of the root whose y -range is between the range of v_- and v_+ , by definition of v_+ , there is no segment in $S_{v''}$ hit by ρ^+ . We conclude that σ_+ is the first segment hit by ρ^+ among the segments in $\mathcal{L} - (S_{v_-} \cup S_{v_+})$.

Inductive Step: Assume the lemma holds at the end of step i , i.e. we have at least one of v_+ and v_- at level i and σ_+ is the first segment hit by ρ^+ among all segments in $\mathcal{L} - (S_{v_+} \cup S_{v_-})$.

During $(i+1)$ th step we ray-shoot on ρ^+ among segments stored in $\mathcal{S}_{v_+}, \mathcal{S}_{v_-}, \mathcal{M}_{v_+}$ and \mathcal{M}_{v_-} , and update σ_+ if necessary. Let v_s be the node containing the first segment hit by ρ^+ among \mathcal{M}_{v_+} and \mathcal{M}_{v_-} (if such a segment exists). Let also v_p be the node containing the first segment hit by ρ^- among \mathcal{M}_{v_+} and \mathcal{M}_{v_-} (if such a segment exists).

By definition of v_s , for any node v which is a child of v_- or v_+ with higher y -range than v_s , σ_+ is below all segments of S_v . Similarly, for a node v' which is a child of v_- or v_+ with smaller y -range than v_p (if v_p exists), there is no segment of $S_{v'}$ hit by ρ^+ (since there exists a segment in \mathcal{S}_{v_p} hit by ρ^-). Finally, for any child v'' of v_- or v_+ whose y -range is between the range of v_- and v_+ , by definition of v_+ , there is no segment in $S_{v''}$ hit by ρ^+ .

Recall that by the induction hypothesis σ_+ at the end of the previous step was the first segment hit by ρ^+ among segments of $\mathcal{L} - (S_{v_+} \cup S_{v_-})$. Now we updated σ_+ and showed that there is no segment hit by ρ^+ before σ_+ in any subtree other than \mathcal{T}_{v_s} or \mathcal{T}_{v_p} . We conclude that σ is the first segment hit by ρ^+ among the segments in $\mathcal{L} - (S_{v_s} \cup S_{v_p})$. Since at the end of the $(i+1)$ th step we set $v_- = v_p$ and $v_+ = v_s$, the lemma follows. ◀

We now explain how Lemma 4 implies the correctness of the query algorithm. To see that, let i be the last level where either v_+ or v_- is defined; at the beginning of the query algorithm at level i , σ_+ is the first segment hit by ρ^+ among segments of $\mathcal{L} - (S_{v_-} \cup S_{v_+})$. Moreover at the end of this step, both v_s and v_p are not defined, i.e., for each child v of v_- or v_+ there is no segment in S_v hit by ρ^+ before σ_+ . Since $S_{v_-} \cup S_{v_+} = S_{v_-} \cup S_{v_+} \cup (\cup_v S_v)$, we get that σ_+ is the first segment hit by ρ^+ among segments of $\mathcal{L} - (S_{v_-} \cup S_{v_+})$. By checking all segments of $\mathcal{S}_{v_-} \cup \mathcal{S}_{v_+}$ and updating σ_+ if necessary, we make sure that σ_+ is the first segment hit by ρ^+ among segments of \mathcal{L} .

Minimizing and Computing the Inverse Geodesic Length on Trees

Serge Gaspers 

UNSW Sydney, Australia
Data61, CSIRO, Sydney, Australia
sergeg@cse.unsw.edu.au

Joshua Lau¹ 

UNSW Sydney, Australia
joshua.lau@unsw.edu.au

Abstract

For any fixed measure H that maps graphs to real numbers, the MINH problem is defined as follows: given a graph G , an integer k , and a target τ , is there a set S of k vertices that can be deleted, so that $H(G - S)$ is at most τ ? In this paper, we consider the MINH problem on trees.

We call H *balanced on trees* if, whenever G is a tree, there is an optimal choice of S such that the components of $G - S$ have sizes bounded by a polynomial in n/k . We show that MINH on trees is Fixed-Parameter Tractable (FPT) for parameter n/k , and furthermore, can be solved in subexponential time, and polynomial space, whenever H is additive, balanced on trees, and computable in polynomial time.

A particular measure of interest is the Inverse Geodesic Length (IGL), which is used to gauge the efficiency and connectedness of a graph. It is defined as the sum of inverse distances between every two vertices: $IGL(G) = \sum_{\{u,v\} \subseteq V} \frac{1}{d_G(u,v)}$. While MINIGL is $W[1]$ -hard for parameter treewidth, and cannot be solved in $2^{o(k+n+m)}$ time, even on bipartite graphs with n vertices and m edges, the complexity status of the problem remains open in the case where G is a tree. We show that IGL is balanced on trees, to give a $2^{O((n \log n)^{5/6})}$ time, polynomial space algorithm.

The *distance distribution* of G is the sequence $\{a_i\}$ describing the number of vertex pairs distance i apart in G : $a_i = |\{\{u,v\} : d_G(u,v) = i\}|$. Given only the distance distribution, one can easily determine graph parameters such as diameter, Wiener index, and particularly, the IGL. We show that the distance distribution of a tree can be computed in $O(n \log^2 n)$ time by reduction to polynomial multiplication. We also extend the result to graphs with small treewidth by showing that the first p values of the distance distribution can be computed in $2^{O(\text{tw}(G))} n^{1+\varepsilon} \sqrt{p}$ time, and the entire distance distribution can be computed in $2^{O(\text{tw}(G))} n^{1+\varepsilon}$ time, when the diameter of G is $O(n^{\varepsilon'})$ for every $\varepsilon' > 0$.

2012 ACM Subject Classification Mathematics of computing → Trees; Mathematics of computing → Graph algorithms

Keywords and phrases Trees, Treewidth, Fixed-Parameter Tractability, Inverse Geodesic Length, Vertex deletion, Polynomial multiplication, Distance distribution

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.59

Related Version A full version of the paper is available: <https://arxiv.org/abs/1811.03836>.

Funding *Serge Gaspers*: Serge Gaspers is the recipient of an Australian Research Council (ARC) Future Fellowship (FT140100048).

Acknowledgements We thank David Harvey and Ray Li for fruitful discussions and feedback.

¹ Corresponding author



1 Introduction

The *Inverse Geodesic Length (IGL)* is a widely-used measure for quantifying the connectedness and efficiency of a given graph or network. In mathematical chemistry, it is also known as the *Harary Index* [39], and in network science as the *(global) efficiency* [14].

To test the resilience of a graph to vertex failures, the problem of minimizing a particular measure by deleting a fixed number of vertices has been studied extensively [29, 25, 20]. In these cases, heuristics have been used to choose which vertices to delete, and their effect has been assessed using the chosen measure. In particular, Szczepński et al. [34] chose IGL as the measure to be minimized when examining this problem. Nonetheless, only recently has the exact optimization problem itself (MINIGL) been studied.

Veremyev et al. [36] formulated MINIGL as a special case of the Distance-Based Critical Node Detection Problem (DCNP), and reduced the problem to Integer Linear Programming. Aziz et al. [4] observed that MINIGL is NP-complete, since it corresponds to VERTEX COVER when $\tau = 0$, but it is also both NP-complete, and $W[1]$ -hard for parameter k , on both split and bipartite graphs. Najeebullah [30] showed that, under the Exponential Time Hypothesis of Impagliazzo and Paturi [22], MINIGL cannot be solved in $2^{o(k+n+m)}$ time, even on bipartite graphs. On the positive side, it was shown that MINIGL is Fixed-Parameter Tractable (FPT) for parameter twin (or vertex) cover number, and also for $\omega + k$, where ω is the neighbourhood diversity of the graph. In another paper, Aziz et al. [3] showed that MINIGL is $W[1]$ -hard for parameter treewidth. The complexity status of MINIGL when the input graph is a tree was stated as an open question by Aziz et al. [4, 3], and in open problem sessions of IWOCA 2017 and the Sydney Algorithms Workshop 2017.

In Section 3, we examine MINIGL on trees, giving the following results.

► **Theorem 1.1.** *MINIGL is FPT for parameter n/k on trees.*

► **Theorem 1.2.** *There is a $2^{O((n \log n)^{5/6})}$ time, $O(n^3)$ space algorithm for MINIGL on trees, on a real RAM.*

To do so, we prove more general versions of these results, for the MINH problem in the case when H is additive, balanced on trees, and computable in polynomial time.

We give a Dynamic Programming (DP) algorithm that solves MINIGL by matching ordered trees to the structure of the given tree, to give a forest with $n - k$ vertices and minimum IGL. The running time of this algorithm is exponential in L , but polynomial in n , where L is the size of the largest tree in this forest. Since H is balanced, L is bounded by a polynomial in n/k , so MINH is FPT for parameter n/k . Proving that IGL is balanced on trees then gives Theorem 1.1. Choosing this DP algorithm when k is large compared to n , and a simple brute-force algorithm otherwise, gives Theorem 1.2.

IGL has been used to identify key protein residues [9], compare the robustness of botnet structures [16], and assess the impact of attacks on power grids [40]. Thus, the ability to compute the IGL of a graph efficiently serves practical purpose in identifying characteristics of real-world networks.

Since the IGL of a graph can easily be computed from its distance distribution, we examine the problem of computing the distance distribution of trees. By combining the relatively well-known techniques of centroid decomposition and fast polynomial multiplication, we obtain the following result on trees.

► **Theorem 1.3.** *The distance distribution of a tree with n vertices can be computed in $O(n \log^2 n)$ time on a log-RAM.*

We extend this result to graphs with small treewidth. This is of practical note, as real-world graphs for which IGL is an indicator of strength – such as electrical grids [2] and road transport networks [27] – have been found to have relatively small treewidth.

The distance distribution of a graph can be trivially computed from the All Pairs Shortest Paths (APSP). The output of APSP is of size n^2 , so any APSP algorithm requires $\Omega(n^2)$ time. On graphs with treewidth k , APSP can be computed in $O(kn^2)$ time [31], so we seek algorithms that find the distance distribution with a subquadratic dependence on n . Abboud et al. [1] proved that, under the Orthogonal Vectors Conjecture (OVC), there is no algorithm that distinguishes between graphs of diameter 2 and 3 in $2^{o(k)}n^{2-\varepsilon}$ time. Williams [38] showed that the OVC is implied by the Strong Exponential Time Hypothesis (SETH) of Impagliazzo, Paturi and Zane [22, 23]. Since the distance distribution of a graph immediately gives its diameter, this hardness result also applies to computing the distance distribution. We prove the following result.

► **Theorem 1.4.** *The prefix a_1, \dots, a_p of the distance distribution of a graph with n vertices and treewidth k can be computed in $2^{O(k)}n^{1+\varepsilon}\sqrt{p}$ time on a log-RAM, for any $\varepsilon > 0$.*

In particular, the number of relevant values of p is at most the graph's diameter, so when the diameter is $O(n^{\varepsilon'})$ for every $\varepsilon' > 0$, we obtain a $2^{O(k)}n^{1+\varepsilon}$ time algorithm to compute the distance distribution. This matches the known hardness bounds above, in the sense that under the OVC, (or the stronger SETH), the dependence on k must be $2^{\Omega(k)}$ when the dependence on n is subquadratic.

Cabello and Knauer [12] reduced the problem of computing the Wiener index [37] (the sum of distances between every two vertices) to orthogonal range queries in $k - 1$ dimensions. They did so by applying a divide-and-conquer strategy that divides the graph with small separators that are found efficiently. Abboud et al. [1] adapted this approach to find radius and diameter. We take a similar approach, but reduce computing the distance distribution to the following problem rather than to orthogonal range queries.

If v and w are vectors in \mathbb{R}^d , write $v < w$ if each coordinate of w is strictly greater than the corresponding coordinate in v . In this case, we say that w (strictly) *dominates* v . We define the REDBLUEPOLYNOMIAL problem as follows.

REDBLUEPOLYNOMIAL

Input: r red points R_1, \dots, R_r , and b blue points B_1, \dots, B_b in \mathbb{R}^d , along with corresponding non-negative integer values r_1, \dots, r_r , and b_1, \dots, b_b , respectively.

Question: Determine the non-zero coefficients of the polynomial $\sum_{(p,q): R_p < B_q} x^{r_p+b_q}$, as a list of (exponent, coefficient) pairs.

This problem can be solved naively in quadratic time, but we seek a more efficient solution in the case when the value of each point is bounded.

To our knowledge, this problem is new, and a variant of a well-known counting problem, which asks for the number of red points dominated by each blue point. Chan and Pătraşcu [13] showed that this variant can be solved in $O(n\sqrt{\log n})$ time on a Word RAM, using word operations to facilitate efficient counting. Bentley [6] gave a multidimensional divide-and-conquer approach for a similar problem, which Monier [28] showed had complexity $O(dn \cdot B(n, d))$ where $B(n, d) = \binom{d+\lceil \log n \rceil}{d}$.

Bringmann et al. [11] used this fact to show that the method employed by Cabello and Knauer [12], and Abboud et al. [1] can, in fact, be used to compute the Wiener index, radius, and diameter of graphs with treewidth k in $2^{O(k)}n^{1+\varepsilon}$ time for any $\varepsilon > 0$, by proving that $B(n, k) = 2^{O(k)}n^\varepsilon$. Furthermore, Husfeldt [21] gave an improved $2^{O(k)}n$ time algorithm for

computing diameter and radius in the case where the graph also has constant diameter. However, it was noted that this result only pertains to the existence of pairs of vertices at certain distances, and not to counting the number of such pairs. Thus, the result does not directly give further insight to computing distance distributions.

We follow Bentley's method, where it suffices to consider the one-dimensional case, $d = 1$. We resolve this case using square-root decomposition and fast polynomial multiplication. Applying the approach of Bringmann et al. to analyse the running time of this approach gives Theorem 1.4. A detailed discussion of this algorithm is given in Section 4.

Due to space constraints, we omit and abbreviate proofs to some of the more straightforward results, and refer the reader to the full version of this paper for more details.

2 Preliminaries

Let $G = (V, E)$ be a graph and suppose $u, v, w \in V$. We define the *distance* $d_G(u, v)$ between u and v to be the fewest number of edges in any path from u to v , or ∞ if no such path exists, with the convention that $\frac{1}{\infty} = 0$.

In Section 3, we consider the problem when the provided graph is a tree T . In this case, precisely one simple path exists between every pair $\{u, v\} \subseteq V$. Define $\mathcal{P}_T(u, v)$ to be the set of vertices along the simple path from u to v in T , including the endpoints u and v .

Observe that $d_T(u, w) + d_T(w, v) = d_T(u, v)$ if and only if $w \in \mathcal{P}_T(u, v)$. For a vertex w , we also define $\mathcal{P}_T^{-1}(w)$ to be the set of all (unordered) pairs of vertices whose path in T passes through w . Formally, $\mathcal{P}_T^{-1}(w) = \{\{u, v\} \subseteq V : w \in \mathcal{P}_T(u, v)\}$.

A vertex u is a *centroid* of T if the maximum size of a connected component in $T - u$ is minimized. We will use the following results, concerning centroids.

► **Lemma 2.1** (Jordan [24]). *Every tree has either one centroid or two adjacent centroids. If a centroid is deleted from a tree, each tree in the remaining forest contains no more than $\frac{n}{2}$ vertices, where n is the number of vertices in the original tree.*

► **Lemma 2.2.** *Let u be a centroid of a tree T with $n \geq 2$ vertices. Then, $|\mathcal{P}_T^{-1}(u)| \geq \frac{n^2}{4}$.*

Proof. See the full version of the paper. ◀

In Section 4 we also consider the problem of computing the IGL, using the tree decompositions of graphs with small treewidth. A *tree decomposition* of G is a tree \mathcal{T} whose vertices (called *nodes*) are $\{1, \dots, I\}$ and a sequence $\mathcal{V}_1, \dots, \mathcal{V}_I$ of subsets of V (called *bags*) such that

1. $V = \bigcup_{i=1}^I \mathcal{V}_i$;
2. If $uv \in E$, then $\{u, v\} \subseteq \mathcal{V}_i$ for some i ;
3. $\mathcal{V}_a \cap \mathcal{V}_c \subseteq \mathcal{V}_b$ whenever $b \in \mathcal{P}_{\mathcal{T}}(a, c)$.

The *width* of such a tree decomposition is $\max_{i=1}^I |\mathcal{V}_i| - 1$. The *treewidth* $\text{tw}(G)$ of G is the minimum width among all tree decompositions of G .

2.1 Model of computation

We establish our results on models of computation that closely reflect what is available to programmers of high-level languages on physical computing devices today.

In Section 3, we solve MINIGL by explicitly computing the minimum IGL that can be obtained by deleting k vertices from the given tree. We perform this on the *real RAM* formulated by Shamos [33], which allows addition, subtraction, multiplication, division and

comparisons of real numbers in constant time, but does not support rounding a value to the nearest integer, or modulo as native operations. This permits efficiently adding and comparing contributions of distances between vertices to the IGL.

In Section 4, we reduce the problem of computing the IGL of a graph to finding its distance distribution. We solve this on a *log-RAM* introduced by Fürer [19], which is a Word RAM that also supports constant time arithmetic operations (including multiplication, integer modulo, and division) on words of length $O(\log n)$. Fürer showed that on a log-RAM, multiplication of two n -bit integers can be done in $O(n)$ time, using either the approach of Schönhage and Strassen [32] (performing a complex polynomial-based Fast Fourier Transform (FFT) and maintaining sufficient precision), or that of Fürer [18] (performing an FFT over a ring of polynomials).

We extend this to integer polynomials with bounded coefficients, as follows.

► **Lemma 2.3.** *Suppose P and Q are integer polynomials of degree n whose coefficients are non-negative integers, such that their product PQ has coefficients not exceeding some integer m . Then, the coefficients of PQ can be computed from the coefficients of P and Q in $O(n \log m)$ time on a log-RAM.*

Proof. This can be done using Kronecker substitution [26]. See the full version of the paper for further details. ◀

3 MinIGL on Trees

In this section, we give a new subexponential time, polynomial space algorithm for MINH on trees, when H satisfies the following properties. We use this to prove Theorems 1.1 and 1.2, by showing that IGL also satisfies these properties.

► **Definition 3.1 (Additivity).** *We say that a measure H on graphs is additive if $H(G_1 \oplus G_2) = H(G_1) + H(G_2)$ for any vertex-disjoint graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, where $G_1 \oplus G_2$ is the graph $(V_1 \dot{\cup} V_2, E_1 \dot{\cup} E_2)$.*

Call a forest *L -trimmed* if none of its trees contain more than L vertices. In the same way, call a subset of vertices in a tree *L -trimming* if their deletion gives an L -trimmed forest.

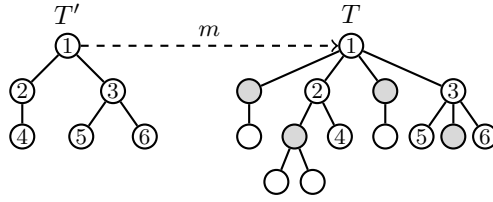
► **Definition 3.2 (Balanced on trees).** *We say that a measure H is balanced (on trees) if there exist positive constants c_H and t_H , such that, for any YES-instance (T, k, τ) of MINH on a tree T with n vertices, there exists a witness that is $c_H (n/k)^{t_H}$ -trimming.*

Hereafter, we will assume that the value of H on a forest is computable in $O(n^\alpha)$ time, and $O(n^\beta)$ space, on a real RAM, where $\alpha, \beta \geq 1$ are constants. We also assume that such a value can be stored in a constant number of words on a real RAM.

We prove Theorems 1.1 and 1.2 by giving compatible algorithms for MINH on trees, then complete the proof by showing that IGL satisfies the same properties that H does. Now it is easy to see that there is a naïve, brute-force algorithm for MINH.

► **Lemma 3.3.** *There is an $O(n^{k+\alpha})$ time, $O(n^\beta)$ space algorithm for MINH on a tree, on a real RAM.*

Proof. We simply try all $\binom{n}{k} = O(n^k)$ subsets of k vertices. The value of H on the forest that remains after each subset has been removed can be computed in $O(n^\alpha)$ time and $O(n^\beta)$ space. ◀



■ **Figure 1** Mapping the vertices of T' to T in Lemma 3.4. Note that T' is an ordered tree, and that children (and their subtrees) must be mapped in order. Shaded vertices will be deleted, and we recursively solve for the subtrees rooted at their children.

If k is small, this algorithm may be efficient. When k is large, the vertices forming an optimal solution will leave a forest of relatively small trees after they are deleted, since H is balanced. We use this property to develop an alternate, more efficient algorithm for MINH in this case. Let $L = c_H (n/k)^{t_H}$. Our algorithm minimizes H , considering only L -trimming subsets of k vertices. The running time of this algorithm is exponential in L , but polynomial in n , so it is fast when k is large, relative to n .

► **Lemma 3.4.** *Let $T = (V, E)$ be a tree with n vertices. There is an $O\left(\frac{4^L}{\sqrt{L}}(n^2 + L^{\alpha-1})\right)$ time, $O(nkL + L^\beta)$ space algorithm on a real RAM, which finds the minimum value of $H(T - S)$, among all L -trimming subsets S of k vertices.*

Proof. We root T arbitrarily and employ DP to compute this minimum value for every subtree and budget, in two cases: the case where the root of the subtree is deleted, and the case where it is not. Denote these minimum values by $f(u, b)$ and $g(u, b)$, respectively, for the subtree rooted at u and budget b . The leaves of the tree form the base cases for this algorithm, and the final answer is derived from the minimum of $f(\text{root}, k)$ and $g(\text{root}, k)$. It remains to give recurrences for f and g .

In the case where u is deleted, we simply need to distribute the remaining $b - 1$ deletions among the subtrees rooted at each child of u . Let the children of u be $v_1, \dots, v_{\text{ch}_T(u)}$ in a fixed order. Our recurrence takes the form of another DP algorithm: let $f'(u, i, b')$ be the minimum value of f distributing a budget of b' deletions among the subtrees rooted at the first i children of u . Our recurrence is as follows:

$$f'(u, i, b') = \min_{0 \leq b'' \leq b'} (\min(f(v_i, b''), g(v_i, b'')) + f'(u, i - 1, b' - b''))$$

and we have that $f(u, b) = f'(u, \text{ch}_T(u), b - 1)$.

If u is not deleted, it will be the root of some tree with no more than L vertices after our chosen subset has been deleted. We fix the structure (formally, an *ordered tree*) for this rooted tree, and attempt to match the vertices in this structure to vertices in the subtree rooted at u . Formally, let the structure be an ordered tree T' over $L' \leq L$ vertices. Let its vertex set be $V' = \{1, \dots, L'\}$ and, without loss of generality, suppose 1 is its root. We seek a total, injective mapping $m : V' \rightarrow V$ satisfying the following conditions.

1. $m(1) = u$;
2. Suppose p and p' are the parents of q and q' in T and T' , respectively. If $m(q') = q$ then $m(p') = p$;
3. Let p and p' be vertices in T and T' such that their children are, in order, $q_1, \dots, q_{\text{ch}_T(p)}$ and $q'_1, \dots, q'_{\text{ch}_{T'}(p')}$, respectively. If $m(q'_{j_1}) = q_{i_1}$, $m(q'_{j_2}) = q_{i_2}$ and $j_1 \leq j_2$, then $i_1 \leq i_2$. That is, children are matched in order.

Note that the structure of the chosen ordered tree uniquely characterises the value of H on the component containing u , since H is only defined on unlabelled graphs, and is additive, so this value is independent of the structure of other components.

Let v be some vertex in T . If v is mapped to by m , then v is a part of this component. Otherwise, if v is not mapped to by m but its parent is, then v must be a vertex chosen for deletion, and so we should recursively consider each of its children's subtrees.

This implies a DP approach to determine the optimal choice of m , similar to that of f' . We let $g'(u, i, b', u', j)$ be the minimum value (of H) induced by a mapping which maps u' to u and maps the first j children of u' among the first i children of u with a total budget of b' deletions in the subtree rooted at u . This value *does not* include the contributions of vertex pairs in T which both end up in the current component (are mapped to by m).

We have a choice to either delete the i th child v_i of u , or map it to the j th child v'_j of u' . In both cases, we allocate a budget of $b'' \leq b'$ deletions to the subtree rooted at v_i . This gives the following recurrence:

$$g'(u, i, b', u', j) = \min_{0 \leq b'' \leq b'} \min \begin{cases} g'(u, i-1, b'-b'', u', j) + f(v_i, b''), \\ g'(u, i-1, b'-b'', u', j-1) + g'(v_i, ch_T(v_i), b'', v'_j, ch_{T'}(v'_j)) \end{cases}$$

and $g(u, b) = \min_{T'} (H(T') + g'(u, ch_T(u), b, 1, ch_{T'}(1)))$. This concludes the description of the algorithm.

A detailed analysis of the time and space complexity of this algorithm is given in the full version of the paper, using the key result that there are $O\left(\frac{4^L}{L\sqrt{L}}\right)$ unlabelled, ordered trees on L or fewer vertices [17, 35]. ◀

Since $n^2 + L^{\alpha-1} = O(n^{\max(2, \alpha-1)})$, and $L \leq c_H \left(\frac{n}{k}\right)^{t_H}$, it follows that MINH is FPT for parameter $\frac{n}{k}$.

► **Corollary 3.5.** *Suppose H is a measure on graphs, that is additive, balanced on trees, and computable in polynomial time on trees, on a real RAM. Then MINH is FPT for parameter n/k on trees.*

With an appropriate threshold, we can combine the approaches of Lemma 3.3 and Lemma 3.4 to give a subexponential time, polynomial space algorithm for MINH.

► **Corollary 3.6.** *Suppose H is a measure on graphs, that is additive, balanced on trees, and computable in polynomial time on trees, on a real RAM. Then there is a $2^{O((n \log n)^{t_H/(t_H+1)})}$ time, polynomial space algorithm for MINH on trees, where t_H is the constant given in Definition 3.2.*

Proof. Lemma 3.3 gives us an $O(n^{k+\alpha}) = 2^{O(k \log n)}$ time algorithm for MINH on a tree. Lemma 3.4 gives us an alternate $O\left(4^{c_H(n/k)^{t_H}} n^{\max(2, \alpha-1)}\right) = 2^{O((n/k)^{t_H} + \log n)}$ time algorithm for the same problem. Note that the memory consumption of both algorithms is bound by $O(n^{\max(3, \beta)})$, so they are both polynomial in space.

Let $k^* = n^{t_H/(t_H+1)} \log^{-1/(t_H+1)} n$. We select the former algorithm when $k \leq k^*$, and the latter algorithm otherwise. In both cases, our running time is bound by $2^{O((n \log n)^{t_H/(t_H+1)})}$, as required. ◀

We now prove that IGL satisfies the requirements of Corollary 3.5 and Corollary 3.6. IGL is clearly additive, since pairs of vertices belonging to different components contribute $\frac{1}{\infty} = 0$ to the IGL. We can easily compute the IGL in $O(n^2)$ time, and $O(n)$ space, on the

real-RAM by traversing from each vertex. Hence, it remains to show that IGL is balanced on trees: it suffices to show that there is a constant t_{IGL} , such that any subset of vertices whose deletion minimizes the IGL is $O\left((n/k)^{t_{\text{IGL}}}\right)$ -trimming.

To do so, we choose to reason about the decrease in IGL caused by the removal of a subset of k vertices, rather than the IGL itself. Maximizing this decrease (which we call *utility*) is equivalent to minimizing the IGL of the graph after removal.

► **Definition 3.7** (Utility). *Let $G = (V, E)$ be a graph. Then the utility of some $S \subseteq V$ is:*

$$\mathcal{U}_G(S) = \text{IGL}(G) - \text{IGL}(G - S).$$

If $S = \{v\}$, we write $\mathcal{U}_G(v)$ instead of $\mathcal{U}_G(\{v\})$, which we call the utility of v in G .

Suppose $S = S' \cup \{v\}$ is a subset of k vertices in a tree T with maximum utility. Necessarily, v must have maximum utility in $T - S'$. This means that v has no less utility than any vertex in its component in $T - S'$, and that it also has no less utility than the optimal vertex in any other component. In this vein, we would like to consider the case when $k = 1$ so we can reason about the individual optimality of each vertex in an optimal solution.

We use the following upper and lower bounds on the utility of the optimal choice of vertex in this case. The proofs of these bounds are straightforward, and provided in the full version of the paper.

► **Lemma 3.8.** *Let $T = (V, E)$ be a tree with $n \geq 2$ vertices. Then, $\max_{v \in V} \mathcal{U}_T(v) \geq n/2$.*

► **Lemma 3.9.** *Let $G = (V, E)$ be a tree with n vertices. Then $\mathcal{U}_G(v) \leq \text{IGL}(G) \leq \frac{1}{2}n(n-1)$ for any vertex $v \in V$.*

Next, we show that the removal of a vertex with maximum utility leaves the remaining forest somewhat balanced. Specifically, it is never the case that one tree in this forest is so large that it contains all but $o(n^{1/4})$ vertices.

► **Theorem 3.10.** *Let $T = (V, E)$ be an unweighted tree with $n \geq 3$ vertices and suppose $v \in V$ minimizes $\text{IGL}(T - v)$. Further, suppose C is a connected component in $T - v$ containing l vertices and let $r = n - l - 1$ be the number of vertices in $T - v$ not in C . Then, there is a constant $0 < c < 1$ independent of n such that $r \geq cn^{1/4}$.*

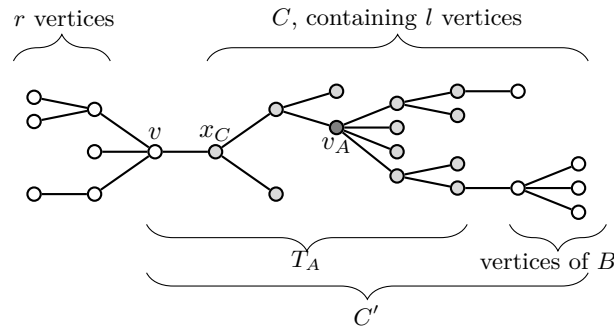
Proof. We may assume $l \geq 1$, since the case when $l = 0$ is trivial. We may also assume that $r \geq 1$, since if $r = 0$, v is a leaf, which contradicts its optimality since $n \geq 3$.

Since T is a tree, each neighbour of v belongs to a different component in $T - v$. Suppose x_C is the neighbour of v in C and let C' be the subtree $T[V(C) \cup \{v\}]$. Thus, v is a leaf of C' . We use this structure (pictured in Figure 2) to give two different, but related, upper bounds for the utility $\mathcal{U}_T(v)$ of v in T .

▷ **Claim 3.11.** $\mathcal{U}_T(v) \leq \frac{1}{2}r(r+1) + (r+1)\mathcal{U}_{C'}(v)$.

Proof. Let us upper bound $\mathcal{U}_T(v)$ by considering the utility of v in C' and also in $T - V(C)$. There are $n - l$ vertices in $T - V(C)$, so by Lemma 3.9, we have that $\mathcal{U}_{T-V(C)}(v) \leq \frac{1}{2}(n-l)(n-l-1) = \frac{1}{2}r(r+1)$. This accounts for the pairs of vertices disconnected by the deletion of v in $T - V(C)$.

We still need to consider such pairs where one vertex is in C , and the other is in $T - V(C)$ (this includes v). Since v is a leaf in C' , the only pairs of vertices connected in C' that are disconnected in $C = C' - v$ are those of the form $\{v, v_C\}$, where v_C ranges over $V(C)$.



■ **Figure 2** Layout of the vertices of T , in Theorem 3.10. Shaded vertices are in A , and are no more than $D = 5$ away from v . The value of D here has chosen for example's sake, and is not the true value constructed in the proof.

Now let u be a vertex in $V \setminus V(C)$. The path from u to v_C must pass through v , and thus $d_T(u, v_C) \geq d_T(v, v_C)$. Hence, the contribution of each disconnected $\{u, v_C\}$ pair is at most that of $\{v, v_C\}$ towards $\mathcal{U}_T(v)$. Putting these inequalities together gives us

$$\begin{aligned} \mathcal{U}_T(v) &= \sum_{\{p,q\} \in \mathcal{P}_T^{-1}(v)} \frac{1}{d_T(p,q)} \\ &= \mathcal{U}_{T-V(C)}(v) + \sum_{\substack{u \in V \setminus V(C) \\ v_C \in V(C)}} \frac{1}{d_T(u, v_C)} \\ &\leq \frac{1}{2}r(r+1) + |V \setminus V(C)| \sum_{v_C \in V(C)} \frac{1}{d_T(v, v_C)} \\ &\leq \frac{1}{2}r(r+1) + (r+1) \sum_{v_C \in V(C)} \frac{1}{d_T(v, v_C)} \\ &= \frac{1}{2}r(r+1) + (r+1)\mathcal{U}_{C'}(v), \end{aligned}$$

as required. ◁

▷ **Claim 3.12.** $\mathcal{U}_T(v) \leq rn$.

Proof. Since v is a leaf of C' , it is distance 1 away from its sole neighbour, and only this neighbour, in C' . Also, the only pairs disconnected by v 's removal in C' are those containing v itself. Now there are $l - 1$ other vertices in C' , each at least distance 2 away from v . Hence, $\mathcal{U}_{C'}(v) \leq 1 + \frac{l-1}{2} = \frac{l+1}{2} = \frac{n-r}{2}$.

Since $r \geq 1$, we know that $r + 1 \leq 2r$. Hence, by Claim 3.11

$$\begin{aligned} \mathcal{U}_T(v) &\leq r^2 + 2r\mathcal{U}_{C'}(v) \\ &\leq r^2 + r(n-r) \\ &= rn, \end{aligned}$$

as required. ◁

Since the utility of deleting v is maximal among all vertices, and $n \geq 2$, we know $\mathcal{U}_T(v) \geq n/2$ from Lemma 3.8. Combining this with Claim 3.11 and rearranging gives

$$\mathcal{U}_{C'}(v) \geq \frac{n - r(r+1)}{2(r+1)}. \tag{1}$$

59:10 Minimizing and Computing the Inverse Geodesic Length on Trees

Suppose, for a contradiction, that $r < \frac{1}{15}n^{1/4}$. Since r is purported to be relatively small, $\mathcal{U}_{C'}(v)$ must be rather large (note it is proportional to n). Intuitively, this implies that many vertices in C' are close to v , and hints towards a more central choice of vertex to delete. We will formally show that such a vertex exists, and is a more optimal choice.

Fix some distance D . We can divide the vertices of C into two groups, A and B : those at most distance D from v in C' (and thus, also in T) and those that are not, respectively. Suppose that $|A| = t$ and that $|B| = |V(C)| - t$. We have the following upper bound:

$$\mathcal{U}_{C'}(v) \leq t + \frac{|V(C)| - t}{D + 1} \leq t + \frac{n - t}{D + 1}, \quad (2)$$

because each vertex in B is at least distance $D + 1$ away from v , and $|V(C)| \leq n$. Note that we do not account for v itself, since the distance to itself does not contribute to its utility.

Recall that $r < \frac{1}{15}n^{1/4}$. It is easy to see that $r(r + 1) \leq n/2$. Combining this with (1) and (2) gives us the following inequality:

$$\frac{n}{4(r + 1)} \leq \mathcal{U}_{C'}(v) \leq t + \frac{n - t}{D + 1},$$

from which we can obtain

$$tD \geq \frac{n(D + 1)}{4(r + 1)} - n.$$

If we choose $D = 8(r + 1) - 1$, it holds that $t \geq \frac{n}{D} = \frac{n}{8r + 7} \geq \frac{n}{15r}$.

Consider the subgraph (a tree) T_A induced by the vertex set $A \cup \{v\}$. T_A contains at least two vertices as v and x_C both must be in A . Also, since T_A is a tree, by Lemma 2.1 it must have a centroid. Let one of the centroids of T_A be v_A . The diameter of T_A is at most $2D$, since every vertex in T_A is at most distance D from v . Combining this with Lemma 2.2, we have

$$\mathcal{U}_{T_A}(v_A) \geq \frac{t^2}{8D} \geq \frac{n^2}{8D^3} \geq \frac{n^2}{8(15)^3 r^3}.$$

Now every pair in T_A that is disconnected by the deletion of v_A is also disconnected in T by the deletion of v_A , so $\mathcal{U}_{T_A}(v_A) \leq \mathcal{U}_T(v_A)$. Also, by the optimality of v in T , we have that $\mathcal{U}_T(v_A) \leq \mathcal{U}_T(v)$. Hence, using the result of Claim 3.12, we can conclude that

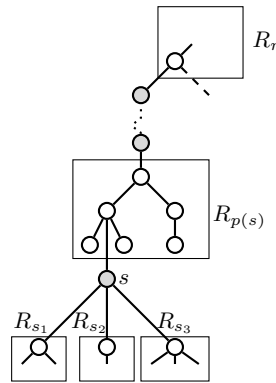
$$\frac{n^2}{8(15)^3 r^3} \leq \mathcal{U}_{T_A}(v_A) \leq \mathcal{U}_T(v) \leq rn.$$

Thus, we have that $r^4 \geq \frac{n}{8(15)^3} \geq \frac{n}{15^4}$, so $r \geq \frac{1}{15}n^{1/4}$, which is a contradiction. The result follows with a choice of $c = \frac{1}{15}$. \blacktriangleleft

We can use this result to finally upper bound the number of vertices in any component after an optimal set of vertices has been removed.

► Theorem 3.13. *Let $T = (V, E)$ be a tree with n vertices, and let $S \subseteq V$ be some subset of vertices such that $|S| = k \geq 1$. There exists a positive constant c' , independent of T and k , such that whenever S minimizes $IGL(T - S)$, S is $(c'(n/k)^5)$ -trimming.*

Proof. We will call the components of $T - S$ *remaining components* and denote each of them by their vertex set. Suppose the remaining components are $R = \{R_1, R_2, \dots, R_{|R|}\}$, where $R_i \subseteq V$ and $S \dot{\cup} R_1 \dot{\cup} \dots \dot{\cup} R_{|R|} = V$. We need to show that $|R_i| \leq c'(n/k)^5$ for each R_i .



■ **Figure 3** Bounding the size of the largest remaining component in Theorem 3.13. In this case, the parent of s is not the root and s has $ch(s) = 3$ children in T' . The shaded vertices are those in S .

We first construct a new graph $T' = (V', E')$ by collapsing each of the remaining components. Formally, $V' = R \dot{\cup} S$, and, for each $R_i \in R$ and $s \in S$, $\{R_i, s\} \in E'$ if and only if there exists some $r \in R_i$ such that $\{r, s\} \in E$. It can be seen that T' is necessarily a tree, and that every R_i is only incident to elements in S . For the remainder of the proof, we further assume that every element in S is only incident to remaining components in T' : if this is not the case, one can add a “dummy” remaining component with cardinality zero between every pair of adjacent elements of S in T' .

Let R_r be a remaining component containing at least as many vertices as any other remaining component. Note that $|R_r| > 0$: it is never a “dummy”. It suffices to show the upper bound holds for R_r . We root T' at R_r . Since $k > 0$, there are strictly fewer than n vertices among the remaining components R . Hence, by the Pigeonhole Principle, there must be some $s \in S$ such that the children $R_{s_1}, R_{s_2}, \dots, R_{s_{ch(s)}}$ of s in T' together contain fewer than n/k vertices. Let the parent of s in T' be $R_{p(s)}$. See Figure 3.

Since S is optimal, s must be an optimal choice of vertex to delete in an instance of MINIGL with graph $T - (S \setminus \{s\})$ and a budget of 1 deletion. In particular, it must also be the optimal choice of vertex to delete in the component containing s in $T - (S \setminus \{s\})$. Hence, we may apply Theorem 3.10 to $T - (S \setminus \{s\})$, in that component to give

$$\frac{n}{k} > \sum_{i=1}^{ch(s)} |R_{s_i}| \geq c(|R_{p(s)}| + |R_{s_1}| + \dots + |R_{s_{ch(s)}}|)^{1/4} \geq c|R_{p(s)}|^{1/4},$$

since $c > 0$, where c is the constant in Theorem 3.10. Thus, we have $|R_{p(s)}| \leq c^{-4} (n/k)^4$.

We now have two cases: if the parent $R_{p(s)}$ of s in T' is the root, R_r , or if it is not the root. If $R_{p(s)}$ is the root, then $p(s) = r$, so $|R_r| \leq c^{-4} (n/k)^4$. Otherwise, s is not a child of the root, and so s must have been a more optimal choice than the best choice in the component induced by R_r in $T - (S \setminus \{s\})$. Since this component contains $|R_r|$ vertices, the best choice had utility at least $\frac{|R_r|}{2}$, by Corollary 3.8. Now the paths that pass through s in $T - (S \setminus \{s\})$ must have one endpoint in some R_{s_j} and the other either in another $R_{s'_j}$ or in $R_{p(s)}$. This is the case since no path can have both endpoints in $R_{p(s)}$. Hence, there are at most $(n/k + 1)(n/k + |R_{p(s)}|)$ such pairs, accounting also for those paths starting at s . Since each of these paths have length at least 1, we have that

$$\frac{|R_r|}{2} \leq \mathcal{U}_{T - (S \setminus \{s\})}(s) \leq \left(\frac{n}{k} + 1\right) \left(\frac{n}{k} + |R_{p(s)}|\right) \leq 2\frac{n}{k} \left(\frac{n}{k} + |R_{p(s)}|\right),$$

59:12 Minimizing and Computing the Inverse Geodesic Length on Trees

because $k \leq n$. Thus

$$\frac{|R_r|}{2} \leq 2\frac{n}{k} \left(\frac{n}{k} + c^{-4} \left(\frac{n}{k} \right)^4 \right) \leq 4c^{-4} \left(\frac{n}{k} \right)^5,$$

because $0 < c < 1$. Hence, $|R_r| \leq 8c^{-4} (n/k)^5$ and the result follows with a choice of $c' = 8c^{-4}$. ◀

Thus, we can choose $c_{\text{IGL}} = c'$ and $t_{\text{IGL}} = 5$, showing that IGL is indeed balanced on trees. This gives Theorem 1.1 and Theorem 1.2.

4 Computing the IGL

Computing the IGL of a graph is trivial once its distance distribution has been determined. In this section, we describe algorithms for efficiently computing the distance distribution of trees, and extend these ideas to graphs with small treewidth.

4.1 Trees

To compute the distance distribution on trees, we present a divide-and-conquer method (commonly known as the *centroid decomposition*, as used in [7]) as follows. We pick a vertex and compute the contribution to the distance distribution of all paths passing through that vertex, using fast polynomial multiplication. Then, we delete the vertex from the tree, and recurse on the remaining connected subtrees. We first provide a method that efficiently computes this contribution.

► **Lemma 4.1.** *Let $T = (V, E)$ be an unweighted tree with n vertices and suppose $r \in V$. Then, the contribution to the distance distribution of all pairs in $\mathcal{P}_T^{-1}(r)$ can be found in $O(n \log n)$ time on a log-RAM.*

Proof. We begin by rooting the tree at r . Suppose the children of r are $s_1, \dots, s_{ch(r)}$ and let $S_1, \dots, S_{ch(r)}$ denote the set of vertices in the subtrees rooted at each child, respectively. With the addition of $S_0 = \{r\}$, the sets S_i form a partition of V .

We perform a depth-first search from r , to find $d_T(r, u) = d_T(u, r)$ for each vertex u and construct a sequence of distance polynomials $P_0, P_1, \dots, P_{ch(r)}$, where $P_i(x) = \sum_{w \in S_i} x^{d_T(r, w)}$. This takes $O(n)$ time, storing each distance polynomial in coefficient form: there are at most n terms overall. Now let

$$P(x) = \left(\sum_{0 \leq i \leq ch(r)} P_i(x) \right)^2 - \left(\sum_{0 \leq i \leq ch(r)} P_i^2(x) \right) = \sum_{0 \leq j \leq n} b_j x^j.$$

We observe that

$$b_j = 2|\{\{u, v\} \in \mathcal{P}_T^{-1}(r) : u \neq v \text{ and } d_T(u, v) = j\}|, \quad (3)$$

that is, b_j is twice the number of pairs of distinct vertices which have a path of length j passing through r . Thus, the required contribution to the distance distribution can be read off from the coefficient form of $P(x)$. The result follows by computing this efficiently from the coefficients of each P_i by applying Lemma 2.3, and observing that the degree of $\sum_{0 \leq i \leq ch(r)} P_i(x)$, and the sum of the degrees of the P_i 's are both at most n . ◀

If we always pick r in Lemma 4.1 to be a centroid of the tree, Lemma 2.1 ensures that each vertex can appear in at most $\log_2 n + 1$ trees throughout the execution of our divide-and-conquer algorithm. A centroid must always exist (also by Lemma 2.1), and we can find one in linear time by recursively computing, then examining, subtree sizes. This gives Theorem 1.3.

► **Theorem 1.3.** *The distance distribution of a tree with n vertices can be computed in $O(n \log^2 n)$ time on a log-RAM.*

If we only wish to determine the first p values of the distance distribution of T , we can modify Lemma 4.1 to run in $O(n + p \log n)$ time, by discarding all terms with degree greater than p when constructing the polynomials. Thus, the expensive multiplication step costs $O(p \log n)$ time by Lemma 2.3, and we obtain Theorem 4.2 as a corollary.

► **Theorem 4.2.** *The prefix a_1, \dots, a_p of the distance distribution of a tree with n vertices can be computed in $O(n \log n + p \log^2 n)$ time on a log-RAM.*

4.2 Graphs with small treewidth

Here, we extend the ideas used in the previous section to prove Theorem 1.4.

Let $G = (V, E)$ be an undirected graph with n vertices, whose edges each have a non-negative weight. We describe a modification of the method of Cabello and Knauer [12], to recursively reduce the task of computing the distance distribution of G to solving instances of REDBLUEPOLYNOMIAL over points in $O(\text{tw}(G))$ dimensions, with values at most p .

In time $2^{O(k)}n$, we can compute a tree decomposition of G of width at most $k = 5 \cdot \text{tw}(G) + 4$ containing at most $O(kn)$ nodes [10]. Using a common technique, we can transform this decomposition into a *nice* tree decomposition with $N = O(kn)$ nodes (see, for example [15]). The nodes of a nice tree decomposition form a rooted binary tree.

Let A be a subset of vertices. A *portal* of A is a vertex in A which has, as a neighbour, some vertex outside A . If these portals are contained in some set $S \subseteq A$, we can partition the vertices of the graph into three sets: $A \setminus S$, S and $V \setminus A$, such that every path from a vertex in A to a vertex in $V \setminus A$ passes through some vertex in S .

Since the nice tree decomposition is a binary tree, there is some edge ij in the decomposition whose removal splits the decomposition's tree into two components I and J (containing nodes i and j , respectively), each containing between $\frac{N}{3}$ and $\frac{2N}{3}$ nodes. Let A be the set of vertices that appear in component I , and let S be the intersection $B_i \cap B_j$ of the bags corresponding to nodes i and j . Necessarily, S must contain all the portals of A due to properties of the tree decomposition. Moreover, $|B_i \cap B_j| \leq \min(|B_i|, |B_j|) \leq k + 1$.

Given this fixed A , recursively find the distance distribution among all pairs of vertices in A as follows. First, perform Dijkstra's algorithm from all vertices in S . For every pair of vertices in S , add an edge whose weight equals the length of the shortest path between them. After these edges are added, the length of the shortest path in G between any pair of vertices in A can be found by only considering paths passing through the vertices of A . Hence, we remove all vertices in $V \setminus A$, and recurse on this smaller graph. Note that I is a valid tree decomposition for this new graph, since $i \in I$, and all the added edges have their endpoints in B_i . Thus, we do not need to find another tree decomposition for this new graph, and its treewidth does not exceed k .

In the same way, we recursively find the distance distribution induced by the pairs of vertices in $(V \setminus A) \dot{\cup} S$. Between these two sets of pairs, we have counted pairs of vertices in S twice, so we subtract the distance distribution induced by these pairs using the shortest paths already computed.

59:14 Minimizing and Computing the Inverse Geodesic Length on Trees

Finally, we must compute the distance distribution among shortest paths between the remaining pairs of vertices: these are the pairs in $(A \setminus S) \times (V \setminus A)$. Let the vertices in S be $s_1, \dots, s_{|S|}$. For every a in $A \setminus S$ and every b in $V \setminus A$, we associate (a, b) with precisely one vertex in s through which some shortest path between the vertices passes. More formally, we will associate (a, b) with the only s_i such that

$$\begin{aligned} d_G(a, s_i) + d_G(s_i, b) &< d_G(a, s_j) + d_G(s_j, b) && \text{for all } j < i, \text{ and} \\ d_G(a, s_i) + d_G(s_i, b) &\leq d_G(a, s_j) + d_G(s_j, b) && \text{for all } j > i. \end{aligned}$$

By rearranging, and observing that all distances are integers, we deduce that this is precisely when

$$\begin{aligned} d_G(a, s_i) - d_G(a, s_j) &< d_G(s_j, b) - d_G(s_i, b) && \text{for all } j < i, \text{ and} \\ d_G(a, s_i) - d_G(a, s_j) &< d_G(s_j, b) - d_G(s_i, b) + 1 && \text{for all } j > i. \end{aligned}$$

Note that all these distances are known from our application of Dijkstra's algorithm from each vertex in S . Since any path from a to b must pass through S , it follows from these inequalities that $d_G(a, s_i) + d_G(s_i, b) = d_G(a, b)$. For each vertex s_i in turn, we will compute the contribution of all pairs of vertices associated with s_i to the distance distribution. We do so by reducing this task to an instance of REDBLUEPOLYNOMIAL.

Our instance will have points in $|S| - 1$ dimensions: one dimension for each $j \neq i$. For each $a \in A \setminus S$, create a red point with coordinate $d_G(a, s_i) - d_G(a, s_j)$ in the dimension corresponding to j , and value $d_G(a, s_i)$, corresponding to each $s_j \neq s_i$. Similarly, for each $b \in V \setminus A$, create a blue point with coordinate $d_G(s_j, b) - d_G(s_i, b)$, for each $j < i$, and $d_G(s_j, b) - d_G(s_i, b) + 1$ for each $j > i$, with value $d_G(s_i, b)$. Importantly, we omit any points with value greater than p : these cannot contribute to the prefix we are trying to compute. Hence, we have created no more than n points in all, each with a non-negative integer value no greater than p . The coefficient of x^l produced by our instance of REDBLUEPOLYNOMIAL corresponds to the number of pairs associated with s_i that are distance l apart. This concludes the description of our reduction.

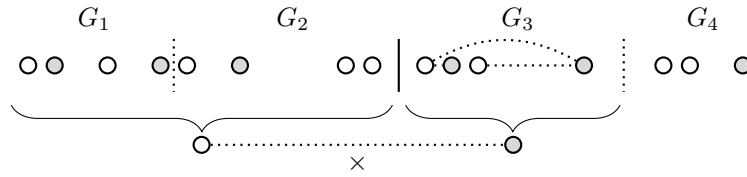
Naturally, we now turn our attention to solving REDBLUEPOLYNOMIAL. Naively, this can be done in quadratic time by considering every pair of points. However, when values are bounded – such as in our instance – we can solve the problem more efficiently.

For a given instance of REDBLUEPOLYNOMIAL, let $n = r + b$ be the total number of points and suppose the value of each point does not exceed some integer $v \geq 0$. Below, we give solutions with time complexity parameterized by both n and v . We consider the 1-dimensional case, then extend this result to higher dimensions recursively.

► **Lemma 4.3.** *When $d = 1$, there is an algorithm that solves REDBLUEPOLYNOMIAL in $O(n\sqrt{v}\log n + n \log n)$ time on a log-RAM.*

Proof. Sort the red and blue points together in non-decreasing order of the coordinate, placing blue points earlier in the order when there are ties. Let t be a positive integer no greater than n . Assign points to groups of size no more than t by placing the first t points, in order, into a group G_1 , followed by the next t points in order into a group G_2 , and so on, so we create $\lceil \frac{n}{t} \rceil$ groups in all. An example is given in Figure 4.

We will separately consider pairs of points that both belong to the same group, and those that belong to different groups. In each group, consider every pair of points, and check if they contribute a term to the polynomial. This takes $O(nt)$ time over all groups.



■ **Figure 4** Square root decomposition in Lemma 4.3. The empty dots represent red points, and the shaded dots represent blue points. When processing G_3 , we consider pairs of points within the group where the red point precedes the blue point. We then consider cross-group pairs whose blue point is in G_3 using fast polynomial multiplication.

It remains to consider pairs that belong to different groups: call these *cross-group* pairs. For each blue point in G_i , we must add an extra term for each red point among G_1, \dots, G_{i-1} . Thus, the total cross-group contribution of all pairs with a blue point in G_i can be written as the following product of two polynomials.

$$\sum_{B_q \in G_i} \sum_{R_p \in G_1 \cup \dots \cup G_{i-1}} x^{r_p + b_q} = \left(\sum_{R_p \in G_1 \cup \dots \cup G_{i-1}} x^{r_p} \right) \left(\sum_{B_q \in G_i} x^{b_q} \right)$$

To compute these contributions, iterate over each group in order, maintaining the coefficient form of the polynomial representing all red points in groups processed thus far. This corresponds to the first multiplicand on the right hand side. We can quickly construct the second multiplicand directly from the elements in this group. Note that the degree of both multiplicands does not exceed v , and that the coefficients of the product do not exceed n^2 . Hence, we can compute the product of these two polynomials in $O(v \log n)$ time by Lemma 2.3, so we can compute the cross-group contributions in $O(\frac{n}{t} v \log n)$ time.

Combining these parts with an appropriate choice of t gives the required result. ◀

► **Theorem 4.4.** *There is an algorithm that solves REDBLUEPOLYNOMIAL in $2^{O(d)} n^{1+\varepsilon} \sqrt{v}$ time on a log-RAM, for every $\varepsilon > 0$.*

Proof. When $d = 1$, we use Lemma 4.3. Otherwise, we will use the divide-and-conquer method of Bentley [6] to reduce the problem to smaller dimensions.

First, combine the red and blue points into one list and apply divide-and-conquer as follows. Let x_m be the median value among the first coordinate of all points. This can be found in $O(n)$ time [8]. We divide the list into two halves as follows. First assign those points with first coordinate less than x_m into the first half, and those with first coordinate greater than x_m into the second half. Among those with first coordinate precisely x_m , assign blue points to the first half until the first half has $\frac{n}{2}$ points. Assign the remaining points to the second half. This assignment can be done in $O(n)$ time and has the property that if $R_p < B_q$, then either both points belong to the same half, or they belong to the first and second half, respectively.

Next, recursively compute the contribution of both groups to the final polynomial. The remaining pairs that may contribute terms to the result must have a red point in the first half, and a blue point in the second half. Since the ordering guarantees that all points in the first half have a first coordinate no greater than those in the second half, we project the red points in the first half together with the blue points in the second half onto a $(d - 1)$ -dimensional space by simply ignoring the first coordinate of each point. We then solve REDBLUEPOLYNOMIAL for this set of points in $d - 1$ dimensions recursively.

59:16 Minimizing and Computing the Inverse Geodesic Length on Trees

The time and space complexity of this algorithm follows from the results of Monier [28] and Bringmann et al. [11], and applying an additional multiplicative factor of \sqrt{v} . A full analysis can be found in the full version of this paper. ◀

An analysis of the algorithm we have described in this section gives Theorem 1.4.

► **Theorem 1.4.** *The prefix a_1, \dots, a_p of the distance distribution of a graph with n vertices and treewidth k can be computed in $2^{O(k)}n^{1+\varepsilon}\sqrt{p}$ time on a log-RAM, for any $\varepsilon > 0$.*

Proof. To find the contribution of pairs in $(A \setminus S) \times (V \setminus A)$, we solve $|S| \leq k + 1$ instances of REDBLUEPOLYNOMIAL in $|S| - 1 \leq k$ dimensions, using the result of Theorem 4.4. As our algorithm performs divide-and-conquer over the nodes of the tree decomposition, each vertex induces the creation of a point in $O((k + 1) \log(kn)) = O(k \log n)$ instances of REDBLUEPOLYNOMIAL. Hence, since the time complexity of Theorem 4.4 is superadditive with respect to n , the total running time over all instances of REDBLUEPOLYNOMIAL is $2^{O(k)}n^{1+\varepsilon} \log n \sqrt{p} = 2^{O(k)}n^{1+\varepsilon'} \sqrt{p}$ for any $\varepsilon' > 0$.

Since we are working on a (nice) tree decomposition with $O(kn)$ nodes, the running time of finding an appropriate dividing edge in the tree, and performing k Dijkstra's per instance are negligible compared to that of solving our instances of REDBLUEPOLYNOMIAL. The result follows from the fact that $k = O(\text{tw}(G))$. ◀

This result can easily be extended to directed graphs, and graphs with bounded edge weights, with some modifications, and a suitable choice of p . On graphs with unit weight edges, setting $p = n - 1$ determines the entire distance distribution.

► **Corollary 4.5.** *The distance distribution of an undirected graph G with n vertices, edges of unit weight and treewidth $\text{tw}(G)$ can be computed in $2^{O(\text{tw}(G))}n^{3/2+\varepsilon}$ time on a log-RAM.*

5 Conclusion

We have provided a general method to solve MINH on trees in subexponential time and polynomial space, whenever H is additive, balanced on trees, and computable in polynomial time. We used this to give a $2^{O((n \log n)^{5/6})}$ time, polynomial space algorithm for MINIGL, by proving that IGL is balanced on trees. Our proof ideas can be used to show that other measures (such as the Wiener index), are also balanced on trees.

For graphs with treewidth k , we have shown that in $2^{O(k)}n^{3/2+\varepsilon}$ time, one can compute the entire distance distribution of the input graph. Compared to the $O(kn^2)$ time algorithm for computing APSP [31], our dependence on n is a factor of $O(\sqrt{n})$ less, though our dependence on k is exponential. Our algorithm is a $O(\sqrt{n})$ factor slower than the current best-known $2^{O(k)}n^{1+\varepsilon}$ time algorithm for diameter [1]. For graphs with diameter $O(n^{\varepsilon'})$ for all $\varepsilon' > 0$, including graphs with polylogarithmic diameter, the extra factor becomes $O(n^\varepsilon)$ for any $\varepsilon > 0$, when compared to the current best-known $2^{O(k)}n$ time algorithm for diameter [21] in this setting. This might be expected, as the distance distribution implies the diameter, and is implied by the APSP, but we find it somewhat surprising that the distance distribution can be computed faster than APSP on graphs with small treewidth.

Our results can be immediately applied to compute any measure of a graph that is a function of the distance distribution. However, they are difficult to adapt to measures that compute properties of individual vertices in the graph, as we exploit properties exclusive to counting pairs that are certain distances apart, without expressly considering which vertices belong to such pairs. In particular, this means that our results are unlikely to directly provide further insight into the efficient computation of related measures, such as the task of computing closeness centrality [5] of every vertex in a given graph.

References

- 1 Amir Abboud, Virginia Vassilevska Williams, and Joshua R. Wang. Approximation and Fixed Parameter Subquadratic Algorithms for Radius and Diameter in Sparse Graphs. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2016)*, pages 377–391. SIAM, 2016. doi:10.1137/1.9781611974331.ch28.
- 2 Karla Atkins, Jiangzhuo Chen, V. S. Anil Kumar, and Achla Marathe. The structure of electrical networks: a graph theory based analysis. *International Journal of Computational Intelligence Systems*, 5(3):265–284, 2009. doi:10.1504/IJCIS.2009.024874.
- 3 Haris Aziz, Serge Gaspers, Edward J. Lee, and Kamran Najeebullah. Defender Stackelberg Game with Inverse Geodesic Length as Utility Metric. In *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018)*, pages 694–702. ACM, 2018.
- 4 Haris Aziz, Serge Gaspers, and Kamran Najeebullah. Weakening Covert Networks by Minimizing Inverse Geodesic Length. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence, (IJCAI 2017)*, pages 779–785. IJCAI, 2017. doi:10.24963/ijcai.2017/108.
- 5 Alex Bavelas. Communication Patterns in Task-Oriented Groups. *The Journal of the Acoustical Society of America*, 22(6):725–730, 1950. doi:10.1121/1.1906679.
- 6 Jon Louis Bentley. Multidimensional divide-and-conquer. *Communications of the ACM*, 23(4):214–229, 1980. doi:10.1145/358841.358850.
- 7 Davide Bilò, Feliciano Colella, Luciano Gualà, Stefano Leucci, and Guido Proietti. A faster computation of all the best swap edges of a tree spanner. In *Proceedings of the 22nd International Colloquium on Structural Information and Communication Complexity (SIROCCO 2015)*, volume 9439 of *LNCS*, pages 239–253. Springer, 2015. doi:10.1007/978-3-319-25258-2_17.
- 8 Manuel Blum, Robert W. Floyd, Vaughan Pratt, Ronald L. Rivest, and Robert E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448–461, 1973. doi:10.1016/S0022-0000(73)80033-9.
- 9 Csaba Böde, István A. Kovács, Máté S. Szalay, Robin Palotai, Tamás Korcsmáros, and Péter Csermely. Network analysis of protein dynamics. *FEBS Letters*, 581(15):2776–2782, 2007. doi:10.1016/j.febslet.2007.05.021.
- 10 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 Journal on Computing*, 45(2):317–378, 2016. doi:10.1137/130947374.
- 11 Karl Bringmann, Thore Husfeldt, and Måns Magnusson. Multivariate Analysis of Orthogonal Range Searching and Graph Distances. In *13th International Symposium on Parameterized and Exact Computation (IPEC 2018)*, volume 115 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:13, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.IPEC.2018.4.
- 12 Sergio Cabello and Christian Knauer. Algorithms for graphs of bounded treewidth via orthogonal range searching. *Computational Geometry: Theory and Applications*, 42(9):815–824, 2009. doi:10.1016/j.comgeo.2009.02.001.
- 13 Timothy M. Chan and Mihai Pătraşcu. Counting inversions, offline orthogonal range counting, and related problems. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2010)*, pages 161–173. SIAM, 2010. doi:10.1137/1.9781611973075.15.
- 14 Paolo Crucitti, Vito Latora, Massimo Marchiori, and Andrea Rapisarda. Efficiency of scale-free networks: error and attack tolerance. *Physica A: Statistical Mechanics and its Applications*, 320:622–642, 2003. doi:10.1016/S0378-4371(02)01545-5.
- 15 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.
- 16 David Dagon, Guofei Gu, Christopher P. Lee, and Wenke Lee. A Taxonomy of Botnet Structures. In *Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC 2007)*, pages 325–339. IEEE Computer Society, 2007. doi:10.1109/ACSAC.2007.44.

- 17 Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 1st edition, 2009.
- 18 Martin Fürer. Faster Integer Multiplication. *SIAM Journal on Computing*, 39(3):979–1005, 2009. doi:10.1137/070711761.
- 19 Martin Fürer. How fast can we multiply large integers on an actual computer? In *Proceedings of the 11th Latin American Symposium on Theoretical Informatics (LATIN 2014)*, volume 8392 of *LNCS*, pages 660–670. Springer, 2014. doi:10.1007/978-3-642-54423-1_57.
- 20 Murad Hossain, Sameer Alam, Tim Rees, and Hussein Abbass. Australian Airport Network Robustness Analysis : A Complex Network Approach. In *Proceedings of the 36th Australasian Transport Research Forum (ATRF 2013)*, pages 1–21, 2013.
- 21 Thore Husfeldt. Computing Graph Distances Parameterized by Treewidth and Diameter. In *Proceedings of the 11th International Symposium on Parameterized and Exact Computation (IPEC 2016)*, volume 63 of *LIPICs*, pages 16:1–16:11. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPICs.IPEC.2016.16.
- 22 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.
- 23 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.
- 24 Camille Jordan. Sur Les Assemblages des Lignes. *Journal für die Reine und Angewandte Mathematik*, 70:185–190, 1869.
- 25 István A. Kovács and Albert-László Barabási. Destruction perfected. *Nature*, 524(7563):38–39, 2015. doi:10.1038/524038a.
- 26 Leopold Kronecker. *Grundzüge einer arithmetischen Theorie der algebraischen Grössen*. G. Reimer, 1882.
- 27 Silviu Maniu, Pierre Senellart, and Suraj Jog. An Experimental Study of the Treewidth of Real-World Graph Data. In *22nd International Conference on Database Theory (ICDT 2019)*, volume 127 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:18, 2019. doi:10.4230/LIPIcs.ICDT.2019.12.
- 28 Louis Monier. Combinatorial solutions of multidimensional divide-and-conquer recurrences. *Journal of Algorithms*, 1(1):60–74, 1980. doi:10.1016/0196-6774(80)90005-X.
- 29 Flaviano Morone and Hernán A. Makse. Influence maximization in complex networks through optimal percolation. *Nature*, 524(7563):65–68, 2015. doi:10.1038/nature14604.
- 30 Kamran Najeebullah. *Complexity of Optimally Attacking and Defending a Network*. PhD thesis, UNSW Sydney, 2018.
- 31 Léon Planken, Mathijs de Weerd, and Roman van der Krogt. Computing All-Pairs Shortest Paths by Leveraging Low Treewidth. *Journal of Artificial Intelligence Research*, 43:353–388, 2012. doi:10.1613/jair.3509.
- 32 A. Schönhage and V. Strassen. Schnelle Multiplikation großer Zahlen. *Computing*, 7(3-4):281–292, 1971. doi:10.1007/BF02242355.
- 33 Ian Michael Shamos. *Computational geometry*. PhD thesis, Yale University, 1978.
- 34 Piotr L. Szczyptański, Tomasz P. Michalak, and Talal Rahwan. Efficient algorithms for game-theoretic betweenness centrality. *Artificial Intelligence*, 231:39–63, 2016. doi:10.1016/j.artint.2015.11.001.
- 35 Kevin Topley. Computationally Efficient Bounds for the Sum of Catalan Numbers. Technical Report 1601.04223, ArXiv, 2016. arXiv:1601.04223.
- 36 Alexander Veremyev, Oleg A. Prokopyev, and Eduardo L. Pasiliao. Critical nodes for distance-based connectivity and related problems in graphs. *Networks*, 66(3):170–195, 2015. doi:10.1002/net.21622.
- 37 Harry Wiener. Structural Determination of Paraffin Boiling Points. *Journal of the American Chemical Society*, 69(1):17–20, 1947. doi:10.1021/ja01193a005.

- 38 Ryan Williams. A New Algorithm for Optimal Constraint Satisfaction and Its Implications. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP 2004)*, volume 3142 of *LNCS*, pages 1227–1237. Springer, 2004. doi:10.1007/978-3-540-27836-8_101.
- 39 Bo Zhou, Xiaochun Cai, and Nenad Trinajstić. On Harary index. *Journal of Mathematical Chemistry*, 44(2):611–618, 2008. doi:10.1007/s10910-007-9339-2.
- 40 Yihai Zhu, Jun Yan, Yan Sun, and Haibo He. Revealing cascading failure vulnerability in power grids using risk-graph. *IEEE Transactions on Parallel and Distributed Systems*, 25(12):3274–3284, 2014. doi:10.1109/TPDS.2013.2295814.

Result-Sensitive Binary Search with Noisy Information

Narthana S. Epa

School of Computing and Information Systems, The University of Melbourne, Victoria, Australia
nepa@student.unimelb.edu.au

Junhao Gan 

School of Computing and Information Systems, The University of Melbourne, Victoria, Australia
junhao.gan@unimelb.edu.au

Anthony Wirth 

School of Computing and Information Systems, The University of Melbourne, Victoria, Australia
awirth@unimelb.edu.au

Abstract

We describe new algorithms for the *predecessor* problem in the *Noisy Comparison Model*. In this problem, given a sorted list L of n (distinct) elements and a query q , we seek the *predecessor* of q in L : denoted by u , the largest element less than or equal to q . In the Noisy Comparison Model, the result of a comparison between two elements is non-deterministic. Moreover, multiple comparisons of the same pair of elements might have different results: each is generated *independently*, and is correct with probability $p > 1/2$. Given an overall error tolerance Q , the cost of an algorithm is measured by the total number of noisy comparisons; these must guarantee the predecessor is returned with probability at least $1 - Q$. Feige et al. showed that predecessor queries can be answered by a modified binary search with $\Theta(\log \frac{n}{Q})$ noisy comparisons.

We design result-sensitive algorithms for answering predecessor queries. The query cost is related to the index, k , of the predecessor u in L . Our first algorithm answers predecessor queries with $O(\log \frac{\log^{*(c)} n}{Q} + \log \frac{k}{Q})$ noisy comparisons, for an arbitrarily large constant c . The function $\log^{*(c)} n$ iterates c times the iterated-logarithm function, $\log^* n$. Our second algorithm is a genuinely result-sensitive algorithm whose *expected* query cost is bounded by $O(\log \frac{k}{Q})$, and is guaranteed to terminate after at most $O(\log \frac{\log n}{Q})$ noisy comparisons.

Our results strictly improve the state-of-the-art bounds when $k \in \omega(1) \cap o(n^\varepsilon)$, where $\varepsilon > 0$ is some constant. Moreover, we show that our result-sensitive algorithms immediately improve not only predecessor-query algorithms, but also *binary-search-like* algorithms for solving key applications.

2012 ACM Subject Classification Theory of computation → Sorting and searching; Theory of computation → Predecessor queries

Keywords and phrases Fault-tolerant search, random walks, noisy comparisons, predecessor queries

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.60

Funding *Anthony Wirth*: Funded by the Melbourne School of Engineering.

Acknowledgements We thank our anonymous reviewer for directing us to the work of Karp and Kleinberg [8].

1 Introduction

Let U be a totally ordered universe of elements. Consider a sorted (abstract) list L of n (distinct) elements from U , in ascending order, and indexed starting from 1. Denote by $L[j]$ the j^{th} element in L for $j = 1, 2, \dots, n$. Given a query element q , the *predecessor* problem on L is to return the index $k = \max\{j \mid L[j] \leq q\}$; if the set $\{j \mid L[j] \leq q\}$ is empty, return 0. As the 0 case can be identified easily, without loss of generality, we assume that the predecessor always exists in L . The predecessor query is one of the most fundamental and important



© Narthana S. Epa, Junhao Gan, and Anthony Wirth;
licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 60; pp. 60:1–60:15

Leibniz International Proceedings in Informatics



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

primitives in computer science; it is a crucial building block for a large number of data structures and algorithms. Any improvements (even a small constant) in the efficiency of answering predecessor queries would immediately benefit all these algorithms, improving practicality. In fact, the predecessor query has been widely studied in various computation models. For example, it is well known that a predecessor query can be answered with $\Theta(\log n)$ comparisons by *binary search* in the (usual, non-noisy) Comparison Model.

In this paper, we focus on the *Noisy Comparison Model*, which was proposed by Feige et al. [7]. It has two parameters: (i) the *probability of a correct comparison*, $p > 1/2$, and (ii) the *overall error tolerance* $Q \in (0, 1/2]$. In this model, each comparison between a pair of elements is “noisy”, in the sense that the comparison result is not deterministic. Instead, each comparison is answered *correctly* by a comparison oracle *independently* with probability p . In other words, for two different comparisons queries between the *same* pair of elements, the oracle may return different results, completely independently. The correct answer is returned with probability p , the incorrect answer with probability $1 - p$.

In this paper, we seek algorithms that minimize the number of calls to the *comparison oracle*. The *cost* of an algorithm is measured by the total *number* of noisy comparisons. We require that the algorithm solves the predecessor problem correctly with probability at least $1 - Q$.

1.1 Applications

The Noisy Comparison Model is particularly useful for modelling application scenarios where comparisons between elements are difficult and costly. For example, in a crowd-sourcing scenario, a human worker may make mistakes when comparing two given objects such as images, and each such comparison incurs some cost, e.g., a dollar. As a result, an efficient algorithm in the Noisy Comparison Model nicely trades off budget with accuracy. Another example is in employee recruitment, where the comparison result between two applicants may be noisy (“incorrect”) due to lack of familiarity with or even bias in relation to certain applicants. Potentially, a noisy comparison algorithm could help guide a process to improve fairness with limited resources.

1.2 Previous Innovation

To answer a predecessor query under the Noisy Comparison Model, we must return the correct answer with probability $1 - Q$. To achieve this, a naive approach is to replace each deterministic comparison in an algorithm (in the usual model) with a sequence of repeated comparisons between the same two elements in the Noisy Comparison Model. If we want the probability of each such comparison being correct to be $1 - \delta$, each such sequence comprises $O(\log_p(1/\delta))$ repeated noisy comparisons and returns the majority answer. Moreover, to account for the $O(\log n)$ comparisons in traditional binary search, by setting $\delta = O(\frac{Q}{\log n})$, we solve the predecessor problem with probability at least $1 - Q$, making $O(\log n \cdot \log(\frac{\log n}{Q}))$ noisy comparisons.

While the traditional binary search is optimal under the (deterministic) Comparison Model, this naive adaptation of binary search is actually far from optimal under the Noisy Comparison Model. As Feige et al. show [7], the worst-case lower bound on answering a predecessor query under the Noisy Comparison Model is only $\Omega(\log(n/Q))$. Indeed, Feige et al. introduce a binary-search based algorithm whose noisy comparison cost matches this lower bound. Some of our methods build on this algorithm, which we hence refer to as Feige.

1.3 The Open Question

In general, Feige is worst-case optimal. However, when the index k of the predecessor is $O(1)$, the brute-force algorithm which compares q naively with elements of L one-by-one, in ascending order, performs better. The total number of noisy comparisons is bounded by $O(\log(1/Q))$. Hence the state-of-the-art result is $O(\log \frac{1}{Q})$ when $k = O(1)$, and $O(\log \frac{n}{Q})$ when $k = \omega(1)$. The subtle, but crucial, question is: Can we bridge these two bounds smoothly over the entire spectrum $k \in \{1, \dots, n\}$?

With this motivation, in this paper, we design *result-sensitive* algorithms for answering predecessor queries, and solving related problems, in the Noisy Comparison Model. That is, the costs of the algorithms should depend on the result index, k .

1.4 Our Contributions

We develop result-sensitive algorithms for the predecessor problem under noisy comparisons, then apply these to Range Count, Stabbing Count, and Shortlisting problems. We start with a function definition. The function $\log^{*(c)} n$ iterates the iterated-logarithmic function c times. That is, $\log^{*(1)} n = \log^* n$ and $\log^{*(c)} n = \log^*(\log^{*(c-1)} n)$.

► **Theorem 1.** *Let $c \geq 1$ be an arbitrarily large integer constant. There exists an algorithm that, on every predecessor query, answers correctly with probability at least $1 - Q$, and makes $O(\log \frac{\log^{*(c)} n}{Q} + \log \frac{k}{Q})$ noisy comparisons.*

Of course, $\log^* n$ grows very slowly, e.g., $\log^* n = 6$ for $n = 2^{2^{32}}$. However, $\log^{*(c)} n$, the bound in Theorem 1, is unfortunately *not* genuinely result sensitive as it depends on n , not k . Nonetheless, the analysis of this algorithm, supporting Theorem 1, is relatively simple, building on the Feige algorithm [7]. With a more careful algorithm design and analysis, we show that:

► **Theorem 2.** *There exists an algorithm that, on every predecessor query, answers correctly with probability at least $1 - Q$, and makes $O(\log \frac{k}{Q})$ noisy comparisons in expectation.*

As a result, our algorithm in Theorem 2 has bridged the noisy-comparison bounds, of $O(\log \frac{1}{Q})$ for $k = O(1)$, and of $O(\log \frac{n}{Q})$ for $k = \omega(1)$, over the whole spectrum of k . Moreover, we highlight that for $k \in \omega(1) \cap o(n^\varepsilon)$, with ε being an arbitrarily small constant, our algorithm strictly improves both of the two state-of-the-art bounds; for the $O(1)$ and $\Omega(n^\varepsilon)$ ranges of k , our algorithm matches the better of the two bounds.

Our Contribution to Applications

Our result-sensitive algorithm also immediately improves two types of noisy-comparison algorithms for solving certain problems:

Type-I. algorithms that include predecessor search as a black box; and

Type-II. algorithms that generalise the comparison oracle in predecessor search

Type-I algorithms. We consider the following two example problems:

► **Problem 1 (Range Count Query).** *Given a sorted list L of n elements from U , and a query range $(a, b]$, a range count query returns the number of elements in L falling into $(a, b]$.*

► **Problem 2 (Stabbing Count Query).** *Consider a set S of n closed intervals, each of which appears in two sorted lists. One list comprises S sorted by left endpoints; the other list comprises S sorted by right endpoints. Given a query value q , a stabbing count query returns the number of intervals of S that contain q .*

In addition, by Theorem 2, we have:

► **Corollary 3.** *There exists an algorithm that, on every range count query, answers correctly with probability at least $1 - Q$, and makes $O(\log \frac{k_a+1}{Q} + \log \frac{\text{count}+1}{Q})$ noisy comparisons in expectation. The value k_a is the index of the predecessor of a in L , while count is the number of elements in L falling in $(a, b]$.*

► **Corollary 4.** *There exists an algorithm that, on every stabbing count query, answers correctly with probability at least $1 - Q$, and makes $O(\log \frac{k_r+1}{Q} + \log \frac{\text{count}+1}{Q})$ noisy comparisons in expectation. The value k_r is the number of right endpoints no larger than q , while count is the number of intervals in S stabbed by q .*

When $k_a, k_r, \text{count} \in \omega(1) \cap o(n^\epsilon)$, the algorithms in these two corollaries strictly improve the results implied by the state-of-the-art Feige algorithm.

Type-II algorithms. We consider this problem:

► **Problem 3 (Shortlisting).** *Given two sorted lists, A and B , with n_A and n_B elements, respectively, that are disjoint, and a value m , $1 \leq m \leq n_A + n_B$, return the m smallest elements (i.e., a shortlist of size m) from the conceptual merged sorted list of A and B .*

By generalising the comparisons in a predecessor search, we prove:

► **Theorem 5.** *There exists an algorithm for the Shortlisting problem that, on every input, with probability at least $1 - Q$, returns a set of the m smallest elements, not necessarily sorted, and makes $O(\log \frac{\min\{k, m-k\}+1}{Q})$ noisy comparisons in expectation, where k is the number of elements from A being in the shortlist.*

2 Related Work

An early model for computing with errors was the Rényi-Ulam game, where the player must identify an element of a finite set of integers by asking questions from an adversary who may lie a bounded number of times [15, 18]. Variants have been considered where the errors were bounded, either globally or as a running total. Binary-search style algorithms to play this game under such models were put forward by Rivest et al. [16], Saks and Wigderson [17], and Borgstrom and Kosaraju [4].

The Noisy Comparison Model used in this paper is different, but related to the Rényi-Ulam game [11, 7]. In this model, errors are random, independent and transient, meaning that repeated comparisons can be used to bound the correctness probability for the result. Feige et al. [7] consider the problems of binary search, sorting, merging, and ranked selection under this model. Their algorithm for binary search is a basic result that we rely on for much of this paper. Algorithms for the problems of sorting, merging and ranked selection with related error models have been more recently considered by Ravikumar for merge sorting with bounded errors [14], Luecci and Liu for minimum selection with noisy errors [10], and Chen et al. for ranked selection with noisy errors [5]. For a more complete history and summary of the various error models for fault tolerant searching, refer to the survey paper by Pelc [12] or the book of Cicalese [6].

Karp and Kleinberg [8] pursued an alternative comparison model. There, the oracle's probability of reporting "less than" is an increasing function of the index in the list L . In that the probability p is fixed, the model presented here is a special case of that model. Unlike Karp and Kleinberg [8], however, we succeed only if we find the exact predecessor; they permit an approximately close answer. Applying the algorithm of Karp and Kleinberg [8] to our scenario would find an approximate predecessor with probability at least $3/4$.

The predecessor problem may be solved with an unbounded search algorithm to achieve a result-sensitive comparison bound. It is straightforward to apply an unbounded search algorithm to solve the predecessor problem with an output-sensitive running time of $O(\log k)$, where k is the index of the predecessor. The unbounded search problem was established, for the deterministic case, by Bentley and Yao [3]. Further lower and upper bounds have been established due to the work of Raoult and Vuilleman [13], Knuth [9], and Beigel [2]. Pelc [11] also addresses the problem of unbounded search in the Noisy Comparison Model that is used in this paper. However, if we treat the tolerance, Q , as a constant, letting k be the index of the result, when the probability of an correct individual comparison, p , is in $[\frac{1}{3}, \frac{1}{2})$, these algorithms make $O(\log^2 k)$ comparisons [11]. However, in a survey article [12], Pelc notes that for the case of linearly bounded errors, Aslam and Dhagat [1] improved the comparison bound to $O(\log k)$ for all $p < \frac{1}{2}$. Furthermore, Pelc observed that this implies the existence of an unbounded search algorithm in the Noisy Comparison Model that performs $O(\log k)$ for every $p < \frac{1}{2}$. However this asymptotic result on the number of comparisons is due to varying the tolerance, Q . In our algorithms, we add a $\log(1/Q)$ term to the number of comparisons; the algorithms Pelc refers to have a $1/(1 - \sqrt{1-Q})$ term, which grows much faster than $\log(1/Q)$ as $Q \rightarrow 0$.

3 Almost Result-Sensitive Algorithms

The predecessor *index* of a query element q is k : we denote the actual predecessor, $L[k]$, by u . Moreover, we assume that $k \neq 0$. For simplicity, we also assume that n is a tower-of-two number, namely, $n = 2^{2^{c-2}}$. Otherwise, we could pad L with dummy elements of value ∞ , increasing n to be a tower-of-two. As a result, for every integer i with $1 \leq i \leq \log^* n$, $\log^{(i)} n$ is an integer, where $\log^{(1)} n = \log n$ and $\log^{(i)} n = \log(\log^{(i-1)} n)$.

In this section, we prove Theorem 1, showing an almost result-sensitive predecessor algorithm, with

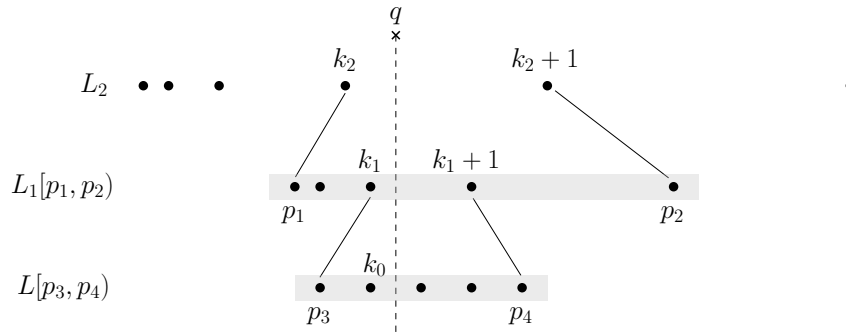
$$O\left(\log \frac{\log^{*(c)} n}{Q} + \log \frac{k}{Q}\right)$$

noisy comparisons. The basic idea is to incorporate Feige as a black box. As the first step, in Section 3.1 we describe an algorithm with a slightly worse asymptotic bound on the number of noisy comparisons.

3.1 An $O\left(\log \frac{\log^* n}{Q} + \log \frac{k}{Q}\right)$ -Algorithm

Denote by $L[a, b)$ the sub-list of L of a *contiguous* index range $\{a, \dots, b-1\}$ with $1 \leq a \leq b \leq n$. Suppose we identify a sub-list $L[a, b)$ that contains the predecessor u and whose length, $b - a$, is bounded by $O(k^{c'})$, for some constant c' . By applying Feige, we can answer the query on $L[a, b)$ with $O(\log \frac{k}{Q})$ noisy comparisons. The basic idea of the algorithm proposed in this subsection is to use binary search with noisy comparisons to identify such a sub-list $L[a, b)$ with length bounded by $O(k)$.

To ease the presentation, setting $L_0 \equiv L$, we let $L_i[j]$ denote the item in location j of list L_i , while $I_i[u]$ is the *dual*, the index of item u in list L_i . Meanwhile, $Pred_i[q]$ is the *index* of the predecessor of q in list L_i . For this purpose, we define two *conceptual* lists:



■ **Figure 1** Our three-level binary search with noisy comparisons. Searching for q in L_2 first, we then “zoom in” to a sub-list of L_1 , i.e., $L_1[p_1, p_2]$. Then, via the predecessor of q in $L_1[p_1, p_2]$, we further “zoom in” to a sub-list of L , i.e., $L[p_3, p_4]$, in which we can find the predecessor of q in L .

Power-of-two List L_1 : This is the sorted sub-list of L comprising the elements at indexes that are *powers of 2*. It has $1 + \log n$ elements:

$$L[1], L[2], L[4], L[8], \dots, L[n] = L[2^{\log n}].$$

Iterated-log List L_2 : This is the sorted sub-list of L comprising the elements at the *iterated-logarithmic* indexes. It has $1 + \log^* n$ elements:

$$L[1] = L[\log^{(\log^* n)} n], L[\log^{(\log^* n - 1)} n], L[\log^{(\log^* n - 2)} n], \dots, \\ L[\log^{(2)} n], L[\log n], L[n] = L[\log^{(0)} n].$$

Our algorithm adopts a three-level binary search with noisy comparisons, in which the search range is narrowed, by searching at finer and finer granularities. Figure 1 illustrates this principle: while L_2 has $O(\log^* n)$ elements, both the lengths of $L_1[p_1, p_2]$ and $L[p_3, p_4]$ are bounded by $O(k)$. As a result, the total number of noisy comparisons is bounded by $O(\log \frac{\log^* n}{Q} + \log \frac{k}{Q})$, as claimed. At each level, the noisy binary search has error tolerance $Q/3$; by the union bound, the overall error tolerance is Q .

The details of the algorithm are as follows. Observe that the indexes p_1, p_2, p_3, p_4 , and k_2 are in some sense absolute, whereas the indexes k_0 and k_1 are relative to the sublist in which they are defined.

Feige on L_2 : Let $k_2 = \text{Pred}_2[q]$. If $k_2 = 1 + \log^* n$, then return n as the answer, as the largest element of L_2 is in fact $L[n]$. Otherwise, let $u_2 = L_2[\text{Pred}_2[q]]$ and let $p'_1 = I_0[u_2]$. Then $p'_1 = \log^{(\alpha+1)} n$, where $\alpha = \log^* n - k_2$. Let $p'_2 = \log^{(\alpha)} n$. Furthermore, define $p_1 = \lceil \log p'_1 \rceil + 1$ and $p_2 = \lfloor \log p'_2 \rfloor + 1$. It can be verified that the elements in the sub-list $L_1[p_1, p_2]$ consists of all the elements in L_1 whose indices in L are in $[p'_1, p'_2]$.

Feige on $L_1[p_1, p_2]$: Denote by k_1 the (relative) index of the predecessor u_1 of q in sublist $L_1[p_1, p_2]$. Let $p_3 = I_0[u_1]$, so we have $p_3 = 2^{k_1 + p_1 - 2}$, and also let $p_4 = 2 \cdot p_3$.

Feige on $L[p_3, p_4]$: Let k_0 be the (relative) index of the predecessor u of q in sublist $L[p_3, p_4]$. Finally, return $k = p_3 + k_0 - 1$ as the index of u in L .

Number of Noisy Comparisons

The first search, in L_2 , makes $O(\log \frac{\log^* n}{Q})$ noisy comparisons, as the length of L_2 is bounded by $O(\log^* n)$. In the second search, $L_1[p_1, p_2]$ contains at most $O(\log \frac{p_2'}{p_1'})$ elements. Since $p_1' = \log p_2'$, the length of $L_1[p_1, p_2]$ is bounded by $O(\log p_2') = O(p_1') = O(k)$. Hence the second search makes $O(\log \frac{k}{Q})$ noisy comparisons. Finally, as $L[p_3, p_4]$ has length at most $p_4 - p_3 = p_3 \leq k$, the number of noisy comparisons in the third search is bounded by $O(\log \frac{k}{Q})$. Therefore, the overall number of comparisons is $O(\log \frac{\log^* n}{Q} + \log \frac{k}{Q})$.

3.2 An $O(\log \frac{\log^{*(c)} n}{Q} + \log \frac{k}{Q})$ -Algorithm

Observe that when $k \in \Omega(\log^* n)$, the comparison cost of the three-level algorithm becomes $O(\log \frac{k}{Q})$; but when $k \in o(\log^* n)$, the most expensive part, asymptotically, is Feige on L_2 . To improve the $\log^* n$ term in the bound, we bootstrap our algorithm, replacing Feige on L_2 with our three-level binary search algorithm itself, setting the tolerance $Q/6$ for each run of Feige. That is, we treat L_2 as an input to a new predecessor query of q , and solve it with $O(\log \frac{\log^* |L_2|}{Q} + \log \frac{k_2}{Q})$ noisy comparisons. Combining with the search costs on sub-lists L_1 and L , the overall bound is improved to $O(\log \frac{\log^* \log^* n}{Q} + \log \frac{k}{Q})$.

In fact, we can repeat this process, bootstrapping c times. By setting the tolerance of each run of Feige to $Q/(3c)$, by the union bound, the tolerance of the overall algorithm is still at most Q . The total number of noisy comparisons is bounded by $O(\log \frac{\log^{*(c)} n}{Q} + c \cdot \log \frac{c \cdot k}{Q})$. As long as c is a constant, the bound is $O(\log \frac{\log^{*(c)} n}{Q} + \log \frac{k}{Q})$, proving Theorem 1.

4 Genuinely Result-Sensitive Algorithms

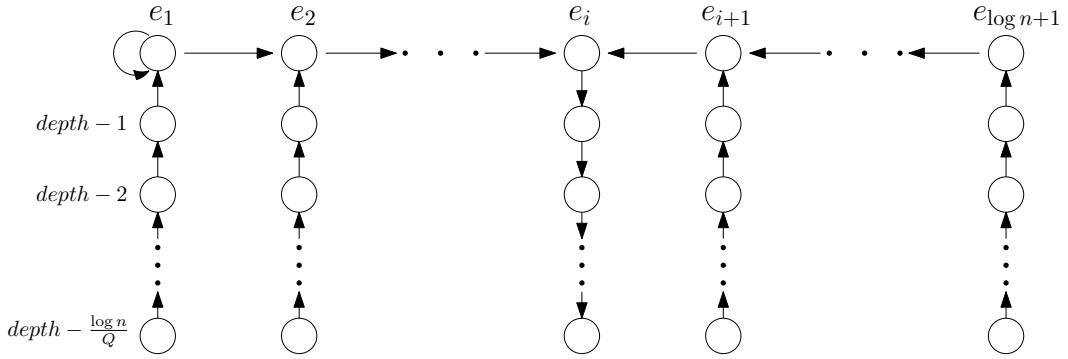
Incorporating Feige as a black box, as we did in Section 3, admits clean analysis. Unfortunately, the bound on the number of noisy comparisons is not quite result sensitive. In this section, we introduce truly result-sensitive asymptotic behavior, by describing an algorithm comprising two phases:

Phase 1: Find $u_1 = L_1[\text{Pred}_1[q]]$, with tolerance $Q' = Q/2$.

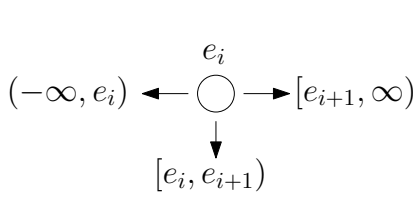
Phase 2: Let $p_1 = I_0[u_1]$. Find the predecessor of q in the sub-list $L[p_1, 2 \cdot p_1]$ with tolerance $Q/2$. Let k_0 be the (relative) index, and return $k = p_1 + k_0 - 1$.

By the union bound, the overall tolerance of this two-phase algorithm is at most Q . Moreover, since $k \geq p_1$, the length of $L[p_1, 2 \cdot p_1]$ is at most k . Hence the cost of Feige in Phase 2 is $O(\log \frac{k}{Q})$ noisy comparisons. Therefore, in the rest of this section, we focus on designing an algorithm which completes Phase 1 with $O(\log \frac{k}{Q})$ noisy comparisons *in expectation*.

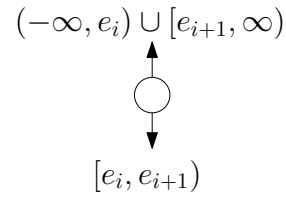
As the first step, we propose a Phase-1 method that can, with probability at least $1 - Q$, find the correct predecessor of q in L_1 , with $O(\log \frac{k}{Q})$ noisy comparisons (for convenience, we write Q , rather than Q' , locally). Unfortunately, there is no guarantee that this method terminates, and hence there is no worst-case bound on the number of noisy comparisons. Nonetheless, in Section 4.2 we refine the method so that it *always* halts, has error tolerance Q , and *in expectation* makes at most $O(\log \frac{k}{Q})$ noisy comparisons. For convenience, we assume $(\log n)/Q$ is an integer.



■ **Figure 2** Comparison tree T , including directed edges displayed for oriented comparison tree T_q^o , with item $e_i = L_1[\text{Pred}_1[q]]$.



■ **Figure 3** The horizontal node of e_i : its three edges partition the search space.



■ **Figure 4** Non-leaf vertical node of e_i : its two edges partition the search space.

4.1 A Random-Walk Phase-1 Algorithm

The algorithm proposed in this subsection is based on random walks on a *conceptual comparison tree*, denoted by T , which is defined as follows and shown in Figure 2 (ignoring the edge directions).

- There is a *horizontal path* with $1 + \log n$ nodes in T , where the i^{th} node, $i \in [1, \log n + 1]$, corresponds to $e_i = L_1[i]$, i.e., $L_0[2^{i-1}]$. Each node in this path is called a *horizontal node*.
- For each e_i , linked to its horizontal node, there is a *vertical extended path* with $(\log n)/Q$ nodes. The j^{th} node of this vertical extended path ($j \in [1, \frac{\log n}{Q}]$) is at *depth*- j . Each node in this extended path is called a *vertical node* of e_i .
- For each e_i , its horizontal node has three edges which correspond to a partition of the search space, as shown in Figure 3. Specifically, (i) the left edge (in fact, a self loop for e_1) corresponds to $(-\infty, e_i)$; (ii) the right edge corresponds to $[e_{i+1}, \infty)$; and (iii) the downward edge corresponds to $[e_i, e_{i+1})$ (in fact, for $e_{1+\log n}$, the right and downward edges are the same). When a walk (with respect to query q) is at the horizontal node e_i , if the noisy comparisons indicate that q falls in one of the three partitions, then the walk should move along the corresponding edge.
- A vertical node at *depth*- $(\log n)/Q$ has only an upward edge. Every other vertical node has two edges, corresponding to a partition of the solution space, as shown in Figure 4. The upward edge corresponds to $(-\infty, e_i) \cup [e_{i+1}, \infty)$, while the downward edge corresponds to $[e_i, e_{i+1})$. As with the horizontal nodes, when a walk for a query q is at the current vertical node, it should move along the edge corresponding to the range where q falls, as indicated by the comparisons.

Consider the query q in the *Deterministic Comparison Model*. Each edge in T can be *conceptually* oriented according to the result of a deterministic comparison on query q . We call this *the oriented comparison tree* with respect to q , denoted by T_q^o , as shown in Figure 2. We emphasize that both T and T_q^o are conceptual: neither is materialized.

Analysis

Referring to Figure 2, starting at an arbitrary node s in T_q^o , the walk that follows the edge directions, reaches the deepest vertical node t , the leaf at depth $(\log n)/Q$, of the predecessor of q in L_1 . Moreover, such a walk is the shortest path (in terms of the number of moves) from s to t in T_q^o .

Our first Phase-1 method, Algorithm 4.1, performs a random walk on T_q^o , starting at the horizontal node of e_1 . It comprises a sequence of moves, each of which the result of the appropriate number of repeated noisy comparisons so that with probability, $2/3$, the move follows the direction of the edge in T_q^o . Such a move is called *forward*, whereas a move in the opposite direction is *backward*. (Given $p > 1/2$, we can boost the probability of a forward move to at least $2/3$ with a constant-factor blowup in noisy comparisons [7].)

■ **Algorithm 4.1** A Random-Walk Phase-1 Algorithm.

Let c be some constant to be fixed.

$r \leftarrow 1$

▷ r is the number of rounds

Perform $c \cdot \log_3 \frac{1}{Q}$ moves, starting from the horizontal node of e_1

while true do

5: **for** $j \leftarrow 1; j \leq c; j \leftarrow j + 1$ **do**

 move

if The walk is at a vertical node of e_i at depth $(r + \log_3 \frac{1}{Q})$ **then**

 Return $I_1[e_i]$

$r \leftarrow r + 1$

 ▷ Increase the round number

► **Lemma 6.** *For every r , the probability of Algorithm 4.1 returning a wrong answer in round r is at most $Q/3^r$. Overall, the algorithm returns a wrong answer with probability at most $Q/2$.*

Proof. Observe that Algorithm 4.1 stops only when the random walk reaches a vertical node at depth $(r + \log_3 \frac{1}{Q})$. If the answer is wrong in round r , the walk must have made at least $r + \log_3 \frac{1}{Q}$ backward moves in some vertical path. The probability of this, in round r , is at most $(1/3)^{r + \log_3(1/Q)} \leq Q/3^r$. Summing over all rounds and applying a union bound, the overall failure probability at most $\sum_{r=1}^{\infty} Q/3^r \leq Q/2$. ◀

► **Theorem 7.** *With probability at least $1 - Q$, Algorithm 4.1 halts, having made $O(\log \frac{k}{Q})$ noisy comparisons, and returns $Pred_1[q]$.*

Proof. Let e_{i^*} be $L_1[Pred_1[q]]$. By Lemma 6, the probability of Algorithm 4.1 returning a wrong answer is at most $Q/2$. Next, we show the algorithm returns a correct answer with probability at least $1 - Q/2$ and makes $O(\log \frac{k}{Q})$ noisy comparisons. Then by the union bound, the theorem holds.

Consider the round $r = i^*$, where the random walk has made $c \cdot (i^* + \log_3 \frac{1}{Q})$ moves. Among these, let m_f and m_b be the numbers of forward moves and backward moves, respectively. In order to return a correct answer in this round, the random walk must satisfy

$$m_f - m_b \geq i^* + i^* + \log_3 \frac{1}{Q}. \quad (1)$$

60:10 Result-Sensitive Binary Search with Noisy Information

Here, the first i^* arises from horizontal forward moves to the horizontal node of e_{i^*} , while the $i^* + \log_3 \frac{1}{Q}$ component is the condition of returning an answer in round $r = i^*$. By the Chernoff-Bound argument of Feige et al. [7], $c' \cdot (2 \cdot i^* + \log_3 \frac{1}{Q})$ moves suffice to satisfy inequality (1) with probability at least $1 - Q/2$. Therefore, by setting $c = 2 \cdot c'$, we have $c \cdot (i^* + \log_3 \frac{1}{Q}) \geq c' \cdot (2 \cdot i^* + \log_3 \frac{1}{Q})$ and thus, the algorithm returns the correct answer in round i^* with probability at least $1 - Q/2$. Moreover, the total number of noisy comparisons is $O(i^* + \log \frac{1}{Q}) = O(\log \frac{2^{i^*}}{Q}) = O(\log \frac{k}{Q})$, from the definition of e_{i^*} . ◀

Although Theorem 7 shows that Algorithm 4.1 is result sensitive with error tolerance at most $Q/2$, it is a Las Vegas algorithm. That is, for every integer K , there is a non-zero (albeit negatively exponential in K), probability that the query cost of Algorithm 4.1 exceeds K . Nonetheless, we show in the next subsection that Algorithm 4.1 can be refined such that it *always* terminates with at most $O(\log \frac{\log n}{Q})$ noisy comparisons, while the expected number of noisy comparisons is $O(\log \frac{k}{Q})$.

4.2 The Ultimate Phase-1 Algorithm

The refined algorithm, which we call our Ultimate Phase-1 Algorithm, is as simple as this:

■ **Algorithm 4.2** The Ultimate Algorithm.

Run Algorithm 4.1 for at most $R = \log \log n$ rounds
if Algorithm 4.1 has not yet returned an answer **then**
 Run Feige on L_1 , with tolerance Q

Since Feige algorithm always terminates, Algorithm 4.2 always terminates. Observe that after R rounds of Algorithm 4.1, $O(\log \log n + \log \frac{1}{Q}) = O(\log \frac{\log n}{Q})$ comparisons are made. Running Feige on L_1 , the cost is bounded by $O(\log \frac{\log n}{Q})$. As a result, in the worst case, Algorithm 4.2 makes $O(\log \frac{\log n}{Q})$ noisy comparisons. For those queries whose correct result k satisfies $k = \Omega(\log n)$, this bound is within $O(\log \frac{k}{Q})$. We next show that for $k = o(\log n)$, the *expected* number of noisy comparisons is bounded by $O(\log \frac{k}{Q})$.

We state explicitly the Chernoff-Hoeffding bound that is the backbone of our proof.

► **Fact 1 (Chernoff-Hoeffding Bound).** *Let X_1, X_2, \dots, X_N be random variables such that $\alpha \leq X_i \leq \beta$ for all $i \in [1, N]$. Let $X = \sum_{i=1}^N X_i$ and set $\mu = \mathbb{E}(X)$. Then for all $\delta > 0$, we have:*

$$\Pr[X \leq (1 - \delta) \cdot \mu] \leq \exp\left(-\frac{\delta^2 \mu^2}{N(\beta - \alpha)^2}\right).$$

In our application of the bound, X_i is a random variable $\in \{+1, -1\}$ indicating the i^{th} move is forward ($X_i = 1$) or backward ($X_i = -1$), so that $X = m_f - m_b$. As a result, we have $\alpha = -1$, $\beta = 1$, $\mu = N/3$, and by setting $\delta = 1 - \tau/\mu$, with τ being the threshold for $m_f - m_b$ signifying termination, we have:

$$\Pr[X \leq \tau] \leq \exp\left(-\frac{(\mu - \tau)^2}{4N}\right).$$

Consider a query q with $e_{i^*} = L_1[\text{Pred}_1[q]]$. Let $M = i^* + \log \frac{1}{Q}$ and $\tau = 2 \cdot i^* + \log \frac{1}{Q}$. As shown in the proof of Theorem 7 and by this Chernoff-Hoeffding Bound, for some sufficiently large constant c , when $N = c \cdot M$,

$$\Pr[X \leq \tau] \leq \exp\left(-\frac{(\mu - \tau)^2}{4N}\right) \leq Q,$$

and so Algorithm 4.1 stops in the $(r^* = i^*)^{\text{th}}$ round with probability at least $1 - Q$.

Next, we show the *expectation* of the cost of running all $R = \log \log n$ rounds in Algorithm 4.1, plus the cost for Feige is bounded by $O(N) = O(M) = O(\log \frac{k}{Q})$.

► **Lemma 8.** *Let $\tau^{(0)} = \tau$, $\mu^{(0)} = \mu$ and $N^{(0)} = N$ be the parameters in the r^* th round. In the $(r^* + i)^{\text{th}}$ round for $i \in [1, R - r^*]$, the probability of Algorithm 4.1 not terminating is at most $Q^{1+i\varepsilon}$, where $\varepsilon = 1/M$.*

Proof. In round $r^* + i$, we denote the correct-stop threshold by $\tau^{(i)}$, the number of moves by $N^{(i)}$, and the expected value of X by $\mu^{(i)}$. According to Algorithm 4.1, in each round, the “correct-stop” threshold, τ , gets increased by 1 and N gets increased by c . Therefore, we have:

$$\begin{aligned} \tau^{(i)} &= \tau^{(0)} + i = \tau^{(0)} + \frac{i}{M} \cdot M \leq \left(1 + \frac{i}{M}\right) \cdot \tau^{(0)} = (1 + i \cdot \varepsilon) \cdot \tau^{(0)}, \\ N^{(i)} &= N^{(0)} + i \cdot c = N^{(0)} + \frac{i}{M} \cdot c \cdot M = \left(1 + \frac{i}{M}\right) \cdot N^{(0)} = (1 + i \cdot \varepsilon) \cdot N^{(0)}, \\ \mu^{(i)} &= (1/3)N^{(i)} = (1 + i \cdot \varepsilon) \cdot (1/3)N^{(0)} = (1 + i \cdot \varepsilon) \cdot \mu^{(0)}. \end{aligned}$$

Substituting into the Chernoff-Hoeffding Bound, we have the probability of *not* stopping and returning the correct value being

$$\begin{aligned} \Pr[X \leq \tau^{(i)}] &\leq \exp\left(-\frac{(\mu^{(i)} - \tau^{(i)})^2}{4N^{(i)}}\right) \\ &\leq \exp\left(-(1 + i \cdot \varepsilon) \cdot \frac{(\mu^{(0)} - \tau^{(0)})^2}{4N^{(0)}}\right) \\ &\leq Q^{1+i\varepsilon}. \end{aligned}$$

Denote by $\mathbb{E}[\text{Cost}]$ the expectation of the cost of Algorithm 4.2: that is, running all R rounds of Algorithm 4.1, plus the cost of Feige on L_1 .

► **Lemma 9.** *If the predecessor of a query q in L is at index $k = o(\log n)$, the expected query cost of Algorithm 4.2, i.e., $\mathbb{E}[\text{Cost}]$, is $O(\log \frac{k}{Q})$.*

Proof. We partition $\mathbb{E}[\text{Cost}]$ into three parts, and bound each:

- Cost_A : the expected cost of the first r^* rounds of Algorithm 4.1,
- Cost_B : the expected cost of the rounds from $r^* + 1$ to R of Algorithm 4.1,
- Cost_C : the expected cost of Feige on L_1 .

As the comparisons in the first r^* rounds are mandatory, $\text{Cost}_A = N$, where N is the number of comparisons in the first r^* rounds. In order to bound Cost_B , let us consider the following *conceptual* process, which starts after r^* rounds:

■ **Algorithm 4.3** A Conceptual Batch Process.

```

r ← r*
while r < R = log log n do
    run the next M rounds as a batch, without stopping
    if the depth-based stopping condition has been met in one of these M rounds then
5:     return the answer corresponding to when the stopping condition was first met
    r ← r + M

```

Clearly, the cost of this conceptual batch process, Algorithm 4.3, is no less than Algorithm 4.1. The latter executes each round separately and terminates immediately when the stopping condition is met. By Lemma 8, for the ℓ^{th} batch of M rounds, the probability of the algorithm *not stopping* during that batch is at most $Q^{1+\ell}$, for $\ell = 1, 2, \dots, z = \lceil \frac{R-r^*+1}{M} \rceil$. That is, the probability of the ℓ^{th} batch running is Q^ℓ .

Since the cost of each batch of M rounds is $c \cdot M = N$, the expected cost of the whole conceptual batch process, which is an upper bound on Cost_B , is (because $Q \leq 1/2$ and $\mathbb{E}[Y] = \sum_{i=1}^{\infty} \Pr[Y \geq i]$):

$$\text{Cost}_B \leq \sum_{\ell=1}^z Q^\ell \cdot N \leq N \cdot \sum_{l=1}^z \frac{1}{2^l} < N.$$

Finally, the probability that Algorithm 4.2 has not stopped in the first $R = \log \log n$ rounds is at most Q^{1+z} . So we have: $\text{Cost}_C \leq Q^{1+z} \cdot \gamma \cdot \log \frac{\log n}{Q}$, for some constant γ .

In total, the expected cost of the first $R = \log \log n$ rounds is at most $2N$. Moreover, the probability of Feige on L_1 occurring is at most Q^{1+z} , which is at most the probability of each noisy comparison made in the first $R = \log \log n$ rounds. Therefore the expected value of Cost_C is at most the expected value of $\text{Cost}_A + \text{Cost}_B$, up to a constant factor. Putting everything together, we have $\mathbb{E}[\text{Cost}] = \text{Cost}_A + \text{Cost}_B + \text{Cost}_C \leq O(N) = O(M) = O(\log \frac{k}{Q})$. ◀

Proof of Theorem 2. Since the correctness of Algorithm 4.2 is easy to verify, combining with Lemma 9, Theorem 2 holds. ◀

5 Conclusion

In this paper, we have designed result-sensitive algorithms under the Noisy Comparison Model for answering predecessor queries. Specifically, our algorithm in Section 3 correctly answers with probability at least $1 - Q$, and makes $O(\log \frac{\log^{*(c)} n}{Q} + \log \frac{k}{Q})$ noisy comparisons in the worst case, where k is the index of the predecessor in the sorted list. Incorporating Feige as a black box leads to a clean analysis of this algorithm.

In Section 4, we present a genuinely result-sensitive algorithm such that for the queries with $k = \Omega(\log n)$, the cost is $O(\log \frac{k}{Q})$ in the worst case, and for those queries with $k = o(\log n)$, its expected query cost is bounded by $O(\log \frac{k}{Q})$. Our algorithm nicely bridges the state-of-the-art known bounds – $O(\log \frac{1}{Q})$ for $k = O(1)$ and $O(\log \frac{n}{Q})$ for $k = \omega(1)$ – over the whole spectrum of $k = 1, \dots, n$. In particular, for $k \in \omega(1) \cap o(n^\epsilon)$, our algorithm strictly improves the better of the two bounds.

Finally, by solving some key range-query and shortlisting problems, we illustrate how the benefit of our result-sensitive algorithm.

References

- 1 Javed A Aslam and Aditi Dhagat. Searching in the presence of linearly bounded errors. In *STOC*, pages 486–93, 1991.
- 2 R. Beigel. Unbounded Searching Algorithms. *SIAM Journal on Computing*, 19(3):522–537, 1990.
- 3 Jon Louis Bentley and Andrew Chi-Chih Yao. An almost optimal algorithm for unbounded searching. *Information Processing Letters*, 5(3):82–7, 1976.
- 4 Ryan S. Borgstrom and S. Rao Kosaraju. Comparison-based Search in the Presence of Errors. In *STOC*, pages 130–6, 1993.
- 5 Xi Chen, Sivakanth Gopi, Jieming Mao, and Jon Schneider. Competitive analysis of the top- K ranking problem. In *SODA*, pages 1245–64, 2017.

- 6 Ferdinando Cicalese. *Fault-Tolerant Search Algorithms*. Springer, 2016.
- 7 U. Feige, P. Raghavan, D. Peleg, and E. Upfal. Computing with Noisy Information. *SIAM Journal on Computing*, 23(5):1001–1018, 1994.
- 8 Richard M. Karp and Robert Kleinberg. Noisy Binary Search and Its Applications. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '07*, pages 881–890, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.
- 9 Donald E. Knuth. Supernatural Numbers. In David A. Klarner, editor, *The Mathematical Gardner*, pages 310–25. Springer US, 1981.
- 10 Stefano Leucci and Chih-Hung Liu. A Nearly Optimal Algorithm for Approximate Minimum Selection with Unreliable Comparisons. *arXiv*, 2018. [arXiv:1805.02033](https://arxiv.org/abs/1805.02033).
- 11 Andrzej Pelc. Searching with known error probability. *Theoretical Computer Science*, 63(2):185–202, 1989.
- 12 Andrzej Pelc. Searching games with errors—fifty years of coping with liars. *Theoretical Computer Science*, 270(1):71–109, 2002.
- 13 J. C. Raoult and J. Vuillemin. Optimal unbounded search strategies. In Jaco de Bakker and Jan van Leeuwen, editors, *ICALP*, pages 512–530, 1980.
- 14 B. Ravikumar. A Fault-Tolerant Merge Sorting Algorithm. In Oscar H. Ibarra and Louxin Zhang, editors, *Computing and Combinatorics*, pages 440–447, 2002.
- 15 Alfréd Rényi. On a problem of information theory. *MTA Mat. Kut. Int. Kozl. B*, 6:505–516, 1961.
- 16 R.L. Rivest, A.R. Meyer, D.J. Kleitman, K. Winklmann, and J. Spencer. Coping with errors in binary search procedures. *Journal of Computer and System Sciences*, 20(3):396–404, 1980.
- 17 M. Saks and A. Wigderson. Probabilistic Boolean decision trees and the complexity of evaluating game trees. In *FOCS*, pages 29–38, 1986.
- 18 Stanisław M Ulam. *Adventures of a Mathematician*. Univ of California Press, 1991.

A Appendix: Applications

As mentioned in the Introduction, our result-sensitive algorithm for predecessor queries immediately improves two types of algorithms for related problems. We provide details here, proving Corollaries 3 and 4 and Theorem 5.

A.1 Direct Applications of Predecessor Search

Predecessor search appears as a black box in this type of application. We discuss two example problems: (i) Range Count Query, and (ii) Stabbing Count Query. Specifically, we prove Corollaries 3 and 4.

Proof of Corollary 3 for Range Count Query. For the given query range $(a, b]$, we can first apply our Algorithm 4.2 to find the index k_a of the predecessor of a in L , with error tolerance $Q/2$. Then apply Algorithm 4.2 again on the sub-list $L(k_a, n]$ to find the (relative) index of k_b of the predecessor of b , with error tolerance $Q/2$. Since all the elements in $L[1, k_a]$ must not be in the range $(a, b]$, it can be verified that k_b is exactly the number *count* of elements of L falling in $(a, b]$. The correctness of the algorithm is obvious and, allowing for *count* = 0, the cost is bounded by $O(\log \frac{k_a+1}{Q} + \log \frac{\text{count}+1}{Q})$ in expectation. ◀

Proof of Corollary 4 for Stabbing Count Query. Denote by L_ℓ (respectively, L_r) the list of the intervals of S sorted by their left (respectively, right) endpoints. Let k_ℓ and k_r be the indexes of the predecessors of the query value q in L_ℓ and L_r , respectively. Observe that the number of intervals stabbed by q can be computed as *count* = $k_\ell - k_r$. Thus, we can first apply Algorithm 4.2 on L_r to find the index k_r of $\text{Pred}_r[q]$, and then apply the algorithm

on the sub-list $L_\ell[k_r + 1, n]$ to find the index k'_ℓ of $\text{Pred}_\ell[q]$. Thus, $k'_\ell = k_\ell - k_r = \text{count}$. By setting the error tolerances of each search to $Q/2$, the correctness of the algorithm is straightforward, and the cost is bounded by $O(\log \frac{k_r+1}{Q} + \log \frac{\text{count}+1}{Q})$ in expectation. ◀

A.2 Predecessor Search with Generalised Comparison Oracle

In this subsection, we illustrate how our result-sensitive algorithm can be applied to improve Type-II algorithms by generalising the comparison oracle. As an example, we solve the Shortlisting Problem.

Recall that in the Shortlisting Problem, there are two disjoint sorted lists L_A and L_B , the goal is to return the *set* of m smallest elements (the “shortlist”) in the conceptual merged sorted list of L_A and L_B . Obviously, actually merging A and B Feige—making $O(N \log \frac{N}{Q})$ noisy comparisons (where $N = n_A + n_B$)—immediately solves the problem. However, our goal is to solve the problem with significantly fewer comparisons (in expectation) than this merge sledgehammer. Instead, we aim for $O(\log \frac{k}{Q})$ noisy comparisons, where k is the number of elements from L_A in the shortlist.

The crucial observation is that if we can determine the number k , then we can output the whole shortlist without further comparisons. As we illustrate below, we can compute k with a binary-search-like algorithm. Without loss of generality, we assume $n_A = n_B = m$ since all the elements with indices greater than m in either of the lists can be safely pruned.

The Comparison Oracle for Shortlisting

► **Observation 1.** For $1 \leq i \leq m$, $L_A[i]$ is in the shortlist if and only if $L_A[i] < L_B[m - i + 1]$.

Proof. First, suppose that $L_A[i]$ is in the shortlist; then there are at most $m - i$ elements from L_B in the shortlist. Thus, $L_B[m - i + 1]$ cannot be in the shortlist and therefore, $L_A[i] < L_B[m - i + 1]$.

Conversely, suppose that $L_A[i] < L_B[m - i + 1]$. As a result, if $L_B[m - i + 1]$ is in the shortlist, then $L_A[i]$ must be in the shortlist. However, this forces the shortlist to have size at least $m + 1$. Therefore, there are no more than $m - i$ elements from L_B in the shortlist, implying that there are at least i elements from L_A in the shortlist. Hence, $L_A[i]$ is in the shortlist. ◀

By Observation 1, we can decide whether $L_A[\lfloor m/2 \rfloor]$ is in the shortlist by comparing it with $L_B[m - \lfloor m/2 \rfloor + 1]$. We thus determine which half of the list L_A should be further considered, which is a binary-search-like algorithm. Since checking the condition in Observation 1 only takes one (deterministic) comparison, by the same analysis of Feige [7], we know that we can identify the largest element $L_A[k]$ from L_A in the shortlist with $O(\log \frac{m}{Q})$ noisy comparisons, with an error tolerance Q .

The Comparison Oracle for Our Result-Sensitive Algorithm

We adapt the two-phase algorithm proposed in Section 4 to obtain a result-sensitive algorithm for the Shortlisting Problem. Again, the second phase can be solved by binary search, which we just designed efficiently. We focus on Phase 1, where our goal is to identify the largest element in the power-of-two list of L_A , denoted by L_{A1} , which should be in the shortlist. We simply design the search-space partitions for both the horizontal nodes and the vertical nodes, as shown in Figures 3 and 4. The subsequent analysis follows immediately.

For a horizontal node $e_i = L_{A1}[i] = L_A[2^{i-1}]$, the space partition for its edges is as follows:

- the left edge corresponds to the case that e_i is not in the shortlist: $e_i > L_B[m - 2^{i-1} + 1]$;
- the right edge corresponds to the case that e_{i+1} is in the shortlist: $e_{i+1} < L_B[m - 2^i + 1]$;
- the downward edge corresponds to the case that e_i is in the shortlist, but e_{i+1} is not.

For a vertical node of e_i , the space partition for its two edges is as follows:

- the downward edge corresponds to the case that e_i is in the shortlist, but e_{i+1} is not;
- the upward edge corresponds to the other case.

By plugging the above comparison oracle to Algorithm 4.2, we have a result-sensitive Phase-1 algorithm for the Shortlisting Problem.

Proof of Theorem 5. By the above analysis, and allowing for $k = 0$, we obtain an algorithm to find k in L_A with $O(\log \frac{k+1}{Q})$ noisy comparisons in expectation. By symmetry, we can find the dual index, $m - k$, for L_B with an expected cost $O(\log \frac{m-k+1}{Q})$. Therefore, by running both of the algorithms for L_A and L_B *simultaneously* and stopping as soon as either terminates, the Shortlisting Problem can be solved with $O(\log \frac{\min\{k, m-k\}+1}{Q})$ noisy comparisons in expectation. The answer is correct with probability at least $1 - Q$. ◀

Improved Algorithms for Clustering with Outliers

Qilong Feng

School of Computer Science and Engineering, Central South University, P.R. China
csufeng@csu.edu.cn

Zhen Zhang

School of Computer Science and Engineering, Central South University, P.R. China
csuzz@foxmail.com

Ziyun Huang

Department of Computer Science and Software Engineering, Penn State Erie,
The Behrend College, Erie, PA, USA
zxh201@psu.edu

Jinhui Xu

Department of Computer Science and Engineering, State University of New York at Buffalo, USA
jinhui@cse.buffalo.edu

Jianxin Wang

School of Computer Science and Engineering, Central South University, P.R. China
jxwang@csu.edu.cn

Abstract

Clustering is a fundamental problem in unsupervised learning. In many real-world applications, the to-be-clustered data often contains various types of noises and thus needs to be removed from the learning process. To address this issue, we consider in this paper two variants of such clustering problems, called k -median with m outliers and k -means with m outliers. Existing techniques for both problems either incur relatively large approximation ratios or can only efficiently deal with a small number of outliers. In this paper, we present improved solution to each of them for the case where k is a fixed number and m could be quite large. Particularly, we gave the first PTAS for the k -median problem with outliers in Euclidean space \mathbb{R}^d for possibly high m and d . Our algorithm runs in $O(nd(\frac{1}{\epsilon}(k+m))^{\frac{k}{\epsilon}})^{O(1)}$ time, which considerably improves the previous result (with running time $O(nd(m+k)^{O(m+k)} + (\frac{1}{\epsilon}k \log n)^{O(1)})$) given by [Feldman and Schulman, SODA 2012]. For the k -means with outliers problem, we introduce a $(6 + \epsilon)$ -approximation algorithm for general metric space with running time $O(n(\beta\frac{1}{\epsilon}(k+m))^k)$ for some constant $\beta > 1$. Our algorithm first uses the k -means++ technique to sample $O(\frac{1}{\epsilon}(k+m))$ points from input and then select the k centers from them. Compared to the more involving existing techniques, our algorithms are much simpler, i.e., using only random sampling, and achieving better performance ratios.

2012 ACM Subject Classification Theory of computation \rightarrow Facility location and clustering

Keywords and phrases Clustering with Outliers, Approximation, Random Sampling

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.61

Funding This work is supported by the National Natural Science Foundation of China under Grants (61672536, 61420106009, 61872450, 61828205), National Science Foundation (CCF-1716400, IIS-1910492), Hunan Provincial Science and Technology Program (2018WK4001).

1 Introduction

Clustering is a fundamental problem in computer science and finds applications in a wide range of domains. Depending on the objective function, it has many different variants. Among them, k -median and k -means are perhaps the two most commonly considered versions. For a given set P of n points in some metric space, the k -median problem aims to identify a set of centers $C = \{c_1 \cdots c_k\}$ that minimizes the objective function $\sum_{x \in P} \min_{c_i \in C} \mathbf{d}(x, c_i)$,



© Qilong Feng, Zhen Zhang, Ziyun Huang, Jinhui Xu, and Jianxin Wang;
licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 61; pp. 61:1–61:12

Leibniz International Proceedings in Informatics



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

where $d(x, c_i)$ denotes the distance from x to c_i . The k -means problem is very similar to the k -median problem, except that the clustering cost is measured by the squared distance from each point to its corresponding center.

Both the k -median and the k -means problems have shown to be NP-hard [6, 21]. Thus, most of the previous efforts have concentrated on obtaining approximation solutions. In the metric space settings, Charikar, Guha, and Shmoys [9] gave the first constant factor approximation solution to the k -median problem. Arya et al. [8] later showed that a simple local search heuristic yields a $(3 + \epsilon)$ -approximation. Li and Svensson [26] gave a $(1 + \sqrt{3} + \epsilon)$ -approximation based on a pseudo-approximation. Byrka et al. [23] further improved the approximation ratio to $(2.671 + \epsilon)$. This is the current best known result for the k -median problem. For the k -means problem, Gupta and Tangwongsan [18] demonstrated that local search can achieve a $(25 + \epsilon)$ -approximation in metric spaces. The approximation ratio has been recently improved to $(9 + \epsilon)$ by Ahmadian et al. [3] using a primal-dual algorithm.

All the above results allow the number k of clusters to be any integer between 1 and n . A common way to relax the problem is to assume that k is a fixed number and the space is Euclidean (instead of general metric). For this type of clustering problem, better results have already been achieved. Kumar, Sabharwal, and Sen [25] gave a linear time (i.e., $O(2^{(k/\epsilon)^{O(1)}} nd)$) $(1 + \epsilon)$ -approximation algorithms for either problem in any dimensions. Chen [11] later improved the running time to $O(ndk + 2^{(k/\epsilon)^{O(1)}} d^2 n^\sigma)$ by using coresets, where σ is an arbitrary positive number. Feldman, Monemizadeh, and Sohler [15] further demonstrated that one can construct a coreset for the k -means problem with size independent of n and d . With this, they showed that a $(1 + \epsilon)$ -approximation can be obtained in $O(ndk + d \cdot \text{poly}(k, \epsilon) + (\frac{k}{\epsilon})^{O(k/\epsilon)})$ time. Moreover, both the k -median and the k -means problems admit $(1 + \epsilon)$ -approximations for the case where the dimensionality of the space is a constant [17, 13].

The clustering problem has an implicit assumption that all input points can be clustered into k distinct groups, which may not always hold in real-world applications. Data from such applications are often contaminated with various types of noises, which need to be excluded from the solution. To deal with such noisy data, Charikar et al. [10] introduced the clustering with outliers problem. The problem is the same as the ordinary clustering problem, except that a small portion of the input data is allowed to be removed. The removed outlier points are ignored in the objective function. By discarding the set of outliers, one could significantly reduce the clustering cost and thus improve the quality of solution.

For the k -median with outlier problem, Charikar et al. [10] gave a $(4(1 + \frac{1}{\epsilon}))$ -approximation for metric space, which removes a slightly more than m (i.e., $O((1 + \epsilon)m)$) outliers. Chen [12] later obtained an algorithm which does not violate either k or m , but has a much large constant approximation ratio. Recently, Krishnaswamy, Li, and Sandeep [24] improved the approximation ratio to $7.08 + \epsilon$ [24] using an elegant iterative rounding algorithm. For Euclidean space, better results have also been achieved. Friggstad et al. [16] presented a $(1 + \epsilon)$ -approximation algorithm that uses $(1 + \epsilon)k$ centers and runs in $O((nk)^{1/\epsilon^{O(d)}})$ time. Their algorithm is efficient only in fixed dimensional space. Feldman and Schulman [14] gave a $(1 + \epsilon)$ -approximation algorithm without violating the number of the centers. Their algorithm runs in $O(nd(m + k)^{O(m+k)} + (\frac{1}{\epsilon}k \log n)^{O(1)})$ time, but needs to assume that both k and m are small constants to ensure a polynomial time solution. There has also been work on obtaining coresets for the problem [20].

For the k -means with outliers problem, Friggstad et al. [16] designed a bi-criteria algorithm that uses $(1 + \epsilon)k$ centers and has an approximation ratio of $(25 + \epsilon)$. Krishnaswamy, Li, and Sandeep [24] subsequently presented a $(53 + \epsilon)$ -approximation algorithm. This is the best existing approximation ratio for the problem.

1.1 Our Contributions

In this paper, we consider two variants of the clustering problem with outliers, k -median with outliers in Euclidean \mathbb{R}^d space (where d could be very high) and k -means with outliers in metric space. For both problems, we assume that k is a fixed number and m is a variable less than n .

For the Euclidean k -median with m outliers problem, we give the first PTAS for non-constant m , based on a simple random sampling technique. Our algorithm runs in $O(nd(\frac{1}{\epsilon}(k+m))^{\frac{k}{\epsilon}O(1)})$ time, which significantly improves upon the previously known $(1+\epsilon)$ -approximation algorithm for the problem [14, 16].

► **Theorem 1.** *Given an instance of the Euclidean k -median with m outliers problem and a parameter $0 < \epsilon \leq 1$, there is a $(1+\epsilon)$ -approximation algorithm that runs in $O(nd(\frac{1}{\epsilon}(k+m))^{\frac{k}{\epsilon}O(1)})$ time.*

For the k -means with m outliers problem, we give a $(6+\epsilon)$ -approximation. Our algorithm first uses k -means++ [7] to sample $O(\frac{1}{\epsilon}(k+m))$ points from the input and then select k points from them as centers. k -means++ is an algorithm proposed for resolving the sensitivity issue of Lloyd's k -means algorithm [27] to the locations of its initial centers. Since the k -means with outliers problem needs to discard m outliers, which may cause major changes in the topological structure and clustering cost of the solution, it could greatly deteriorate the performance of many classical clustering algorithms [19, 24]. However, several studies on k -means++ for noisy data seem to suggest that it is an exception and can actually yield quite good solutions [5, 19]. As far as we know, there is no known theoretical analysis that tries to explain the performance of k -means++ on noisy data. The following theorem takes the first step in this direction.

► **Theorem 2.** *Given a point set P in a metric space and a parameter $0 < \epsilon \leq 1$, let C be a set of $O(\frac{1}{\epsilon}(k+m))$ points sampled from P using k -means++. Then, C contains a subset of k centers that induces a $(6+\epsilon)$ -approximation for k -means with m outliers with constant probability.*

As a corollary to Theorem 2, it is easy to see that $O(\frac{1}{\epsilon}(k+m))^k$ sets of candidate centers for the problem can be generated in $O(n(k+m)\frac{1}{\epsilon})$ time. A $(6+\epsilon)$ -approximation can then be obtained by an exhaustive search over the candidate sets.

► **Corollary 3.** *Given an instance of the k -means clustering problem with m outliers and a parameter $0 < \epsilon \leq 1$, there is a $(6+\epsilon)$ -approximation algorithm that runs in time $O(n(\beta\frac{1}{\epsilon}(k+m))^k)$ for some constant $\beta > 1$.*

1.2 Other Related Work

Most of the aforementioned results are mainly for theoretical purpose. There are also more practical solutions available for clustering. The most popular one for k -means is probably the heuristic technique introduced by Lloyd [27], which iteratively assigns the points to their nearest centers and updates the centers as the means of their corresponding newly generated clusters. It is known that Lloyd's algorithm is sensitive to the locations of the initial centers. An effective remedy for this undesirable issue is the use of an initialization algorithm called k -means++, which generates an initial set of cluster centers close to the optimal solution. Arthur and Vassilvitskii [7] showed that the k centers generated by k -means++ induce an $O(\log k)$ -approximation in an expected sense. Ostrovsky et al. [29], Jaiswal and Garg [22],

and Agarwal, Jaiswal, and Pal [1] further revealed that these k centers can lead to $O(1)$ -approximations under some data separability conditions. Ailon, Jaiswal, and Monteleoni [4] demonstrated that a bi-criteria constant factor approximation can be obtained by sampling $O(k \log k)$ points using k -means++. Aggarwal, Deshpande, and Kannan [2] and Wei [30] later discovered that $O(k)$ points are actually sufficient to ensure a constant factor approximation.

2 Preliminaries

The clustering with outliers problem can be formally defined as follows.

► **Definition 4** (*k*-median/*k*-means clustering with outliers). *Let P be a set of n points in a metric space (X, \mathbf{d}) , and $k, m > 0$ be two integers. The *k*-median or *k*-means clustering problem with outliers is to identify a subset $Z \subseteq P$ of size m and a set $C \subseteq X$ of k centers, such that the clustering cost $\Phi(P \setminus Z, C)$ is minimized among all possible choices of Z and C , where $\Phi(P \setminus Z, C) = \sum_{x \in P \setminus Z} \min_{c \in C} \mathbf{d}(x, c)$ for *k*-median and $\Phi(P \setminus Z, C) = \sum_{x \in P \setminus Z} \min_{c \in C} \mathbf{d}^2(x, c)$ for *k*-means.*

In Euclidean space, the clustering with outliers problem is identical, except that the points lie in \mathbb{R}^d , and the centers can be k arbitrary points in \mathbb{R}^d .

We will use the following result to find the approximate centers, which is known as Chernoff bound.

► **Theorem 5** ([28]). *Let $A_1 \dots A_m$ be 0 – 1 independent random variables with $\Pr(A_i = 1) = p_i$. Let $A = \sum_{i=1}^m A_i$ and $u = \sum_{i=1}^m E(A_i)$. Let $0 < \alpha < 1$ be an arbitrary real number. Then, $\Pr[A \leq (1 - \alpha)u] \leq e^{-\frac{\alpha^2 u}{2}}$.*

Given a cluster $A \subseteq \mathbb{R}^d$, let $\Gamma(A)$ denote the optimal 1-median center of A . The following result says that a good approximation to $\Gamma(A)$ can be obtained using a small set of points randomly sampled from A .

► **Lemma 6** ([25]). *Given a set R of size $\frac{1}{\lambda^4}$ randomly sampled from a set $A \subseteq \mathbb{R}^d$, where $\lambda > 0$, there exists a procedure **Construct**(\mathbf{R}) that yields a set of $2^{(1/\epsilon)^{O(1)}}$ points **core**(\mathbf{R}), and there exists at least one point $r \in \mathbf{core}(\mathbf{R})$, such that the inequality*

$$\mathbf{d}(r, \Gamma(A)) \leq \lambda \frac{\Delta(A)}{|A|}$$

*holds with probability at least $\frac{1}{2}$. The procedure **Construct**(\mathbf{R}) runs in $O(2^{(1/\epsilon)^{O(1)}} \cdot d)$ time.*

3 *k*-Median Clustering with Outliers in Euclidean Space

In this section, we present a new algorithm for the *k*-median clustering problem with outliers in the geometric settings. Let $\Phi(x, C) = \min_{c \in C} \mathbf{d}(x, c)$ denote the cost of clustering a point $x \in \mathbb{R}^d$ using a set $C \subseteq \mathbb{R}^d$ of centers. The clustering cost of a point set $P \subseteq \mathbb{R}^d$ induced by C is denoted by $\Phi(P, C) = \sum_{p \in P} \Phi(p, C)$. For a singleton $C = \{c\}$, we also write $\Phi(P, C)$ as $\Phi(P, c)$. The minimum 1-median cost of a set $S \subseteq \mathbb{R}^d$ is denoted by $\Delta(S) = \sum_{x \in S} \mathbf{d}(x, \Gamma(S))$, where $\Gamma(S)$ denotes the optimal center of S .

3.1 The Algorithm

The general idea of our algorithm solving the k -median clustering problem with outliers is as follows. Assume that $\{P_1, \dots, P_k\}$ is the optimal partition of the k -median clustering problem with outliers, where $|P_1| \geq |P_2| \geq \dots \geq |P_k|$. The objective of our algorithm is to find the approximate centers of $P_i (i = 1, \dots, k)$. Assume that $P_i (1 \geq i \geq k)$ is the largest cluster whose approximate center has not yet been found. In our algorithm, two strategies are applied to find the approximate center of P_i . It is possible that the points in P_i are far away from the approximate centers already found. For this case, by randomly sampling points in the remaining data set, with large probability, a large portion of P_i is in the sampled set. By enumerating all possible certain size of subsets of the sampled set, there must exist one subset that an approximate center of P_i can be obtained from this set by Lemma 6. On the other hand, if the points in P_i are close to the set of the approximate centers already found (denoted by C), then one center in C can be used to approximate the center of P_i , and the points close to the approximate centers in C can be deleted from the point set. The specific algorithm for the k -median clustering problem with outliers is described in Algorithm 1. The algorithm Random-sampling has eight parameters $Q, g, k, C, \epsilon, U, N$, and M , where Q is the input dataset, g is the number of centers not yet found, k is the total number of clusters, C is the multi-set of obtained approximate centers, ϵ is a real number ($0 < \epsilon \leq 1$), U is the collection of candidate solutions, $N = (20k^{10} + 4mk^8)/\epsilon^5$, and $M = k^8/\epsilon^4$.

■ **Algorithm 1** The algorithm for k -median with outliers in Euclidean space.

Algorithm Find- k -centers

Input: a point set P , integers $k, m > 0$, and an approximation factor $0 < \epsilon \leq 1$.

Output: a point set $C = \{c_1, \dots, c_k\}$.

1. let $N = (20k^{10} + 4mk^8)/\epsilon^5$, $M = k^8/\epsilon^4$, $U = \emptyset$;
2. **loop** 2^k times **do**
3. **Random-sampling**($P, k, k, \emptyset, \epsilon, U$);
4. **return** the set of centers $C \in U$ with the smallest cost for k -median with m outliers.

Algorithm Random-sampling(Q, g, k, C, ϵ, U)

1. $S = \emptyset$;
 2. **if** $g = 0$ **then**
 3. $U = U \cup \{C\}$;
 4. **return**.
 5. sample a set S of size N from Q independently and uniformly;
 6. **for** each subset $S' \subseteq S$ of size M **do**
 7. **for** each point $c \in \text{core}(S')$ **do**
 8. **Random-sampling**($Q, g - 1, k, C \cup \{c\}, \epsilon, U$);
 9. find the median value β of all values in $\{\Phi(x, C) \mid x \in Q\}$;
 10. $Q' = \{x \in Q \mid \Phi(x, C) \leq \beta\}$;
 11. **Random-sampling**(Q', g, k, C, ϵ, U).
-

3.2 Analysis

In this section we show the correctness of Theorem 2. Given an instance of the k -median clustering problem with m outliers (P, k, m) , let $Z = \{z_1 \dots z_m\}$ be the set of outliers in the optimal solution, and $\mathbb{P} = \{P_1 \dots P_k\}$ be the k -partition of the remaining (inliers) points in P that minimizes the k -median objective function. Without loss of generality, assume that $|P_1| \geq |P_2| \geq \dots \geq |P_k|$. Denote by $\Delta_k = \sum_{i=1}^k \Delta(P_i)$ the clustering cost induced by the optimal solution.

We now give an outline of the proof. In order to prove the correctness of Algorithm Find- k -centers, we need to get that there exists a set of centers in U that achieves the desired approximation for the centers of clusters P_1, \dots, P_k . Assume that a set $C = \{c_1, \dots, c_{i-1}\}$ of centers has been found. The key point is to prove that the c_i obtained by Algorithm Random-sampling based on C can get a good approximation for P_i . The general idea of proving that c_i is a good approximate center of P_i is as follows. A set B of points that are close to C by a fixed value r can be obtained, where the possible value of r can be enumerated efficiently. The following two cases are discussed: (1) $P_i \cap B \neq \emptyset$, and (2) $P_i \cap B = \emptyset$. For the first case, we show that $\Gamma(P_i)$ is close to a previously sampled point, and there exists a center in C that achieves the desired approximation for $\Gamma(P_i)$. For the second case, we prove that $P \setminus B$ contains a substantial part of P_i . We show that by randomly sampling from $P \setminus B$, a subset of points from P_i can be found, and a good approximate center for P_i is obtained by Lemma 6.

► **Lemma 7.** *With a constant probability, there exists a set of approximate centers C^* in the list U generated by the algorithm Find- k -centers, such that for any constant $1 \leq j \leq k$, we have*

$$\mathbf{d}(c_j, \Gamma(P_j)) \leq \frac{\epsilon \Delta(P_j) + 3(j-1)\epsilon \Delta_k}{k^2 |P_j|},$$

where c_j denotes the nearest point to $\Gamma(P_j)$ in C^* .

Before proving Lemma 7, we first show its implication. Let C^* denote the center set given in Lemma 7. Given a cluster $P_j \in \mathbb{P}$, we have

$$\begin{aligned} \Phi(P_j, C^*) &\leq \Phi(P_j, c_j) = \sum_{x \in P_j} \mathbf{d}(x, c_j) \leq \sum_{x \in P_j} (\mathbf{d}(x, \Gamma(P_j)) + \mathbf{d}(\Gamma(P_j), c_j)) \\ &= \Delta(P_j) + |P_j| \mathbf{d}(c_j, \Gamma(P_j)) \leq \Delta(P_j) + \frac{\epsilon \Delta(P_j) + 3(j-1)\epsilon \Delta_k}{k^2} \\ &\leq \Delta(P_j) + \frac{\epsilon \Delta(P_j)}{k^2} + \frac{3(k-1)\epsilon \Delta_k}{k^2}, \end{aligned} \quad (1)$$

where the third step is due to triangle inequality, and the fifth step follows from the assumption that Lemma 7 is true. Summing both sides of inequality (1) over all $P_j \in \mathbb{P}$, we have

$$\sum_{j=1}^k \Phi(P_j, C^*) \leq \Delta_k + \frac{\epsilon \Delta_k}{k^2} + \frac{3(k-1)\epsilon \Delta_k}{k} \leq (1 + 3\epsilon) \Delta_k. \quad (2)$$

This implies that Lemma 7 is sufficient to ensure the approximation guarantee for our algorithm. We now prove the correctness of Lemma 7.

Proof. (of Lemma 7) We prove the lemma by induction on j . We first consider the case of $j = 1$. Our algorithm initially samples a set of N points from P . Let $S = \{s_1, \dots, s_N\}$ denote the set of N points sampled from P . Define N random variables A_1, \dots, A_N , such that if $s_i \in P_1$, $A_i = 1$. Otherwise, $A_i = 0$. Since $|P_1| \geq |P_2| \geq \dots \geq |P_k|$, it is easy to know that for any constant $0 < \epsilon \leq 1$, we have

$$\Pr[A_i = 1] = \frac{|P_1|}{|P|} \geq \frac{|P_1|}{|Z| + k|P_1|} \geq \frac{1}{m+k}.$$

Let $A = \sum_{i=1}^N A_i$ and $u = \sum_{i=1}^N E(A_i)$. We have $u \geq \frac{N}{m+k} \geq \frac{2k^8}{\epsilon^4}$. Using Lemma 5, we get

$$\Pr(A \geq \frac{k^8}{\epsilon^4}) \geq \Pr(A \geq \frac{1}{2}u) = 1 - \Pr(A \leq \frac{1}{2}u) \geq 1 - e^{-\frac{u}{8}} \geq 1 - e^{-k^8/4\epsilon^4} > \frac{1}{2}.$$

This implies that with probability at least $\frac{1}{2}$, the set of N points sampled from P contains more than $\frac{k^8}{\epsilon^4}$ points from P_i . By feeding $\lambda = \frac{k^2}{\epsilon}$ into Lemma 6, we know that the inequality $\mathbf{d}(c_1, \Gamma(P_1)) \leq \frac{\epsilon \Delta(P_1)}{k^2 |P_1|}$ holds with probability at least $\frac{1}{2}$, which implies that Lemma 7 holds for the case $j = 1$.

We now assume that Lemma 7 holds for $j \leq i - 1$ and consider the case of $j = i$. Define a multi-set $C_{i-1}^* = \{c_1, \dots, c_{i-1}\}$, where c_t is the nearest point to $\Gamma(P_t)$ from C_{i-1}^* for any $1 \leq t \leq i - 1$. Define $B_i = \{x \in P \mid \Phi(x, C_{i-1}^*) \leq r_i\}$, where $r_i = \frac{\epsilon \Delta_k}{k^2 |P_i|}$. It is easy to see that B_i is a subset of P that consists of the points close to C_{i-1}^* . We distinguish the analysis into the following two cases.

Case (1): $P_i \cap B_i \neq \emptyset$. In this case, P_i contains some points close to C_{i-1}^* . We prove that one center from C_{i-1}^* can be used to approximate $\Gamma(P_i)$.

Case (2): $P_i \cap B_i = \emptyset$. In this case, all the points from P_i are far from the centers in C_{i-1}^* . We prove that P_i contains a substantial part of $P \setminus B$. Thus, a subset of P_i can be randomly sampled from $P \setminus B$ with high probability. By enumerating this subset, a center can be obtained to approximate $\Gamma(P_i)$.

Case (1): $P_i \cap B_i \neq \emptyset$. Let p be an arbitrary point from $P_i \cap B_i$ and c_f be the nearest point to p in C_{i-1}^* . Let P_f denote the cluster in $\{P_1, \dots, P_{i-1}\}$ such that $\mathbf{d}(c_f, \Gamma(P_f))$ is the smallest value in $\{\mathbf{d}(c_f, \Gamma(P_j)) \mid 1 \leq j \leq i - 1\}$. We now prove that c_f can be used to approximate $\Gamma(P_i)$ by triangle inequality and induction assumption. Observe that

$$\begin{aligned} \mathbf{d}(\Gamma(P_i), c_f) &\leq \mathbf{d}(\Gamma(P_i), p) + \mathbf{d}(p, c_f) \leq \mathbf{d}(\Gamma(P_i), p) + r_i \leq \mathbf{d}(\Gamma(P_f), p) + r_i \\ &\leq \mathbf{d}(\Gamma(P_f), c_f) + \mathbf{d}(c_f, p) + r_i \leq \mathbf{d}(\Gamma(P_f), c_f) + 2r_i \\ &\leq \frac{\epsilon \Delta(P_f) + 3(f-1)\epsilon \Delta_k}{k^2 |P_f|} + 2r_i \\ &= \frac{\epsilon \Delta(P_f) + 3(f-1)\epsilon \Delta_k}{k^2 |P_f|} + \frac{2\epsilon \Delta_k}{k^2 |P_i|}, \end{aligned} \quad (3)$$

where the first step and the fourth step are due to triangle inequality, the second step follows from the fact that $p \in B_i$, the third step is derived from the fact that $p \in P_i$, the sixth step follows from the assumption that Lemma 7 holds for $j \leq i - 1$, and the last step follows from the definition of r_i . Since $f < i$, we have $|P_f| > |P_i|$. This implies that

$$\begin{aligned} \frac{\epsilon \Delta(P_f) + 3(f-1)\epsilon \Delta_k}{k^2 |P_f|} &= \frac{\epsilon \Delta(P_f)}{k^2 |P_f|} + \frac{3(f-1)\epsilon \Delta_k}{k^2 |P_f|} \leq \frac{\epsilon \Delta(P_f)}{k^2 |P_i|} + \frac{3(f-1)\epsilon \Delta_k}{k^2 |P_i|} \\ &\leq \frac{\epsilon \Delta_k}{k^2 |P_i|} + \frac{3(i-1)\epsilon \Delta_k}{k^2 |P_i|} = \frac{(3i-2)\epsilon \Delta_k}{k^2 |P_i|}. \end{aligned} \quad (4)$$

Combining inequalities (3) and (4) together, we get

$$\mathbf{d}(\Gamma(P_i), c_f) \leq \frac{(3i-2)\epsilon \Delta_k}{k^2 |P_i|} + \frac{2\epsilon \Delta_k}{k^2 |P_i|} = \frac{3i\epsilon \Delta_k}{k^2 |P_i|} \leq \frac{\epsilon \Delta(P_i) + 3i\epsilon \Delta_k}{k^2 |P_i|}.$$

This completes the proof of Lemma 7 in case (1).

Case (2): $P_i \cap B_i = \emptyset$. For this case, we will show that P_i contains a large fraction of $P \setminus B_i$. Furthermore, algorithm Random-sampling can find a set Q such that $P \setminus B_i \subseteq Q$ and $|Q| \leq 2|P \setminus B_i|$. Thus, a set S randomly sampled from Q contains a certain number of points from P_i . By enumerating the subsets of S , we can obtain a subset $S' \subseteq P_i$ of size M , which can be used to find the approximate center for P_i by Lemma 6.

We now show that the proportion of the points of P_i in $P \setminus B_i$ is large. We achieve this by dividing $P \setminus B_i$ into three portions: $Z \setminus B_i$, $\sum_{j=1}^{i-1} P_j \setminus B_i$, and $\sum_{j=i}^k P_j \setminus B_i$, and comparing their sizes with $|P_i|$ respectively.

61:8 Improved Algorithms for Clustering with Outliers

▷ **Claim 8.** $\frac{|P_i|}{|P \setminus B_i|} \geq \frac{\epsilon}{5k^2 + m\epsilon}$.

Proof. It is easy to show that $|Z \setminus B_i| \leq m$. By the definitions of B_i and r_i , we know that $\Phi(P_j, C_{i-1}^*) \geq r_i |P_j \setminus B_i|$ for any $1 \leq j \leq i-1$, which implies that

$$\sum_{j=1}^{i-1} |P_j \setminus B_i| \leq \frac{1}{r_i} \sum_{j=1}^{i-1} \Phi(P_j, C_{i-1}^*) \leq \frac{(1+3\epsilon)\Delta_k}{r_i} = \frac{k^2 |P_i| (1+3\epsilon)}{\epsilon},$$

where the second step is due to our induction assumption and a similar argument in obtaining (2), and the last step is due to the definition of r_i .

By the fact that $|P_1| \geq \dots \geq |P_k|$, we have $\sum_{j=i}^k |P_j \setminus B_i| \leq (k-i)|P_i|$. Thus,

$$\begin{aligned} \frac{|P_i|}{|P \setminus B_i|} &= \frac{|P_i|}{|Z \setminus B_i| + \sum_{j=1}^{i-1} |P_j \setminus B_i| + \sum_{j=i}^k |P_j \setminus B_i|} \\ &\geq \frac{|P_i|}{m + \frac{k^2 |P_i| (1+3\epsilon)}{\epsilon} + (k-i)|P_i|} \\ &\geq \frac{1}{m + \frac{k^2(1+3\epsilon)}{\epsilon} + (k-i)} \geq \frac{\epsilon}{5k^2 + m\epsilon}, \end{aligned} \quad (5)$$

where the last inequality is due to the fact that $0 < \epsilon \leq 1$. ◀

Claim 8 implies that P_i contains a large fraction of $P \setminus B_i$. The algorithm finds the set $P \setminus B_i$ by guessing the number of points from $P \setminus B_i$. Given an integer $1 \leq j \leq \log n$, let β_j denote the $\frac{n}{2^{j-1}}$ -th largest value in $\{\Phi(x, C_{i-1}^*) \mid x \in P\}$, and let Q_j denote the set of points $x \in P$ with $\Phi(x, C_{i-1}^*) \leq \beta_j$. We know that there exists a constant l , such that $P \setminus B_i \subseteq Q_l$ and $P \setminus B_i \not\subseteq Q_{l-1}$. It is easy to know that $|P \setminus B_i| \geq \frac{1}{2}|Q_l|$. By Claim 8, we have $\frac{|P_i|}{|Q_l|} \geq \frac{\epsilon}{10k^2 + 2m\epsilon}$. Using Lemma 5, we know that with probability at least $\frac{1}{2}$, the set of N points randomly sampled from Q_l contains more than $\frac{k^8}{\epsilon^4}$ points from P_j . Using Lemma 6, we can find an approximate center c_i such that $\mathbf{d}(c_i, \Gamma(P_i)) \leq \frac{\epsilon \Delta(P_i)}{k^2 |P_i|}$ with probability at least $\frac{1}{2}$. This implies that with probability more than $\frac{1}{2k}$, Algorithm Random-sampling identifies a set C^* of k centers, such that for any constant $1 \leq j \leq k$, we have

$$\mathbf{d}(c_j, \Gamma(P_j)) \leq \frac{\epsilon \Delta(P_j) + 3(j-1)\epsilon \Delta_k}{k^2 |P_j|}.$$

The probability can be boosted to a constant by repeatedly running Random-sampling for 2^k times. This completes the proof of Lemma 7. ◀

We are now ready to prove the correctness of Theorem 1.

► **Theorem 1.** *Given an instance of the Euclidean k -median with m outliers problem and a parameter $0 < \epsilon \leq 1$, there is a $(1+\epsilon)$ -approximation algorithm that runs in $O(nd(\frac{1}{\epsilon}(k+m))^{\binom{k}{\epsilon}})^{O(1)}$ time.*

Proof. Lemma 7 implies that our algorithm gives a $(1+\epsilon)$ -approximation for the problem. We focus on the running time of the algorithm. Let $T(n, g)$ be the running time of algorithm Random-sampling on input $(P, g, k, C, \epsilon, U)$. It is easy to show that $T(n, 0) = O(1)$ and $T(0, g) = 0$. In the algorithm, step 5 takes $(\frac{k+m}{\epsilon})^{O(1)}$ time, step 8 takes $(\frac{k+m}{\epsilon})^{\binom{k}{\epsilon}} \cdot d$ time and yield $(\frac{k+m}{\epsilon})^{\binom{k}{\epsilon}} \cdot d$ candidate centers, and step 9 takes $O(ndk)$ time. Thus we get the following recurrence.

$$T(n, g) = \left(\frac{k+m}{\epsilon}\right)^{O(\frac{k}{\epsilon})} \cdot T(n, g-1) + T\left(\frac{n}{2}, g\right) + \left(\frac{k+m}{\epsilon}\right)^{O(1)} + \left(\frac{k+m}{\epsilon}\right)^{\binom{k}{\epsilon}} \cdot d + O(ndk).$$

Choose $\lambda = (\frac{k+m}{\epsilon})^{(\frac{k}{\epsilon})^{O(1)}}$ to be large enough such that

$$T(n, g) \leq \lambda T(n, g-1) + T(\frac{n}{2}, g) + \lambda(nd).$$

We claim that $T(n, g) \leq nd\lambda^g \cdot 2^{2g^2}$. This claim holds in the base case. We suppose that the claim holds for $T(n', g') \forall n' < n, \forall g' < g$. It is easy to verify that

$$nd\lambda^g \cdot 2^{2g^2} \leq nd\lambda \cdot \lambda^{g-1} \cdot 2^{2(g-1)^2} + \frac{n}{2}d\lambda^g \cdot 2^{2g^2} + \lambda nd,$$

which implies that the claim $T(n, g) \leq nd\lambda^g \cdot 2^{2g^2}$ holds. Thus our algorithm runs in $nd(\frac{1}{\epsilon}(k+m))^{(\frac{k}{\epsilon})^{O(1)}}$ time. \blacktriangleleft

4 k -Means Clustering with Outliers in Metric Space

Our approach for the k -means clustering with m outliers problem first samples a set of $O(\frac{1}{\epsilon}(k+m))$ points with k -means++. Then, it enumerates all the subset of size k of the sampled set and finds the one with the minimal clustering cost. We prove that the subset with minimal clustering cost can achieve $(6 + \epsilon)$ -approximation to the k -means clustering with m outliers problem. The k -means++ algorithm samples a point with probability proportional to its squared distance to the nearest previously sampled point, as detailed in Algorithm 2. For t sampled points, the algorithm runs in $O(nt)$ time.

The notations for k -means follows from that of k -median except for a few modifications. We use the squared distances from the points to their corresponding centers to measure the clustering cost. Let (X, \mathbf{d}) be a metric space, where \mathbf{d} is the distance function defined over all points in X . Given a point $x \in X$ and a set $C \subseteq X$ of cluster centers, let $\Phi(x, C) = \min_{c \in C} \mathbf{d}(x, c)^2$. Given an instance (P, k, m) of the k -means clustering problem with outliers, let $Z = \{z_1 \dots z_m\}$ be the set of outliers in the optimal solution, and $\mathbb{P} = \{P_1 \dots P_k\}$ be the k -partition of the remaining points in P that minimizes the k -means objective function. Given a cluster $P_i \in \mathbb{P}$ and a point c , let $\Gamma(P_i)$ denote its optimal center. The definitions of $\Phi(P_i, C)$, $\Phi(P_i, c)$, and $\Delta(P_i)$ stay unchanged. Let $\mathbf{b}(P_i, \alpha) = \{x \in P_i \mid \mathbf{d}(x, \Gamma(P_i))^2 \leq \alpha r_i\}$ be the closed ball centered at $\Gamma(P_i)$ of radius αr_i , where $r_i = \frac{\Delta(P_i)}{|P_i|}$.

We first give an outline of the proof of Theorem 3. Given a cluster $P_j \in \mathbb{P}$, it is easy to see that if the value of α is small enough, then any point from $\mathbf{b}(P_j, \alpha)$ can be used to approximate the centroid of P_j . This implies that we can achieve the desired approximation ratio through finding a point from $\mathbf{b}(P_j, \alpha)$ for each cluster $P_j \in \mathbb{P}$. For the points in P_j , outliers, and the set of previously sampled points, there are only two possible relations: either the points in P_j and outliers are far away from the set of previously sampled points, or the points in P_j and outliers are close to the previously sampled points. For the case when the points in P_j and outliers are far away from the set of previously sampled points, by applying k -means++, the points in P_j and outliers can be sampled with high probability, and we prove that $\mathbf{b}(P_j, \alpha)$ contains a substantial portion of the sampled points from P_j . For the case when the points in P_j and outliers are close to the previously sampled points, we prove that the probability of sampling a point from $\mathbf{b}(P_j, \alpha)$ and outliers is small, and a previously sampled point can be used to approximate the centroid of P_j .

Let C_i denote the set of points sampled with k -means++ in the first i iterations. Define $\mathbb{O}_i = \{P_j \in \mathbb{P} \mid \text{cost}(P_j, C_i) \leq (6 + \frac{\epsilon}{2})\Delta(P_j)\}$, where $\text{cost}(P_j, C_i) = \min_{c \in C_i} \Phi(P_j, c)$. Let T be union of the set of points outside \mathbb{O}_i and Z . The following lemma shows that if the proportion of the cost from the points in T to C_i in $\Phi(P, C_i)$ is small enough, then the points in C_i give the desired approximation for the problem.

61:10 Improved Algorithms for Clustering with Outliers

■ **Algorithm 2** The k -means++ algorithm.

Input: a point set P and an integer $t > 0$.
Output: a point set $C = \{c_1, \dots, c_t\}$.

1. Sample a point $x \in P$ uniformly at random, initialize C_1 to $\{x\}$;
2. **for** $i = 2$ **to** t **do**:
3. Sample a point $x \in P$ with probability $\frac{\Phi(x, C_i)}{\Phi(P, C_i)}$;
4. $C_i \leftarrow C_{i-1} \cup \{x\}$;
5. $i \leftarrow i + 1$;
6. **return** $C \leftarrow C_i$.

► **Lemma 9.** *If $\sum_{P_j \in \mathbb{P} \setminus \mathbb{O}_i} \Phi(P_j, C_i) + \Phi(Z, C_i) \leq \frac{\epsilon}{53} \Phi(P, C_i)$, then $\sum_{j=1}^k \text{cost}(P_j, C_i) \leq (6 + \epsilon) \Delta_k$.*

We now give two useful properties of the closed ball $\mathbf{b}(P_j, \alpha)$. The first property says that any point in such ball is close to $\Gamma(P_j)$, which can be derived from triangle inequality easily. The second property says that the points in the closed ball $\mathbf{b}(P_j, \alpha)$ are quite far from the centers in C_i .

► **Lemma 10.** *For any cluster $P_j \in \mathbb{P} \setminus \mathbb{O}_i$, we have*

- (i) *For any point $c \in \mathbf{b}(P_j, \alpha)$, $\Phi(P_j, c) \leq (2 + 2\alpha) \Delta(P_j)$.*
- (ii) *Let d_j denote the squared distance between $\Gamma(P_j)$ and its nearest point in C_i . Let $\beta = \frac{d_j}{r_j}$ and $1 < \alpha < \beta$. Then $\beta > 2 + \frac{\epsilon}{2}$ and $\frac{\Phi(\mathbf{b}(P_j, \alpha), C_i)}{\Phi(P_j, C_i)} \geq \frac{1}{2(\beta+1)} (4 \frac{\sqrt{\beta_j}}{\sqrt{\alpha}} + \beta_j + \ln \alpha - 4\sqrt{\beta_j} - \frac{\beta_j}{\alpha})$.*

By feeding $\alpha = 2 + \frac{\epsilon}{4}$ into Lemma 10, we get that any point from $\mathbf{b}(P_j, 2 + \frac{\epsilon}{4})$ can give a $(6 + \frac{\epsilon}{2})$ -approximation for the optimal centroid of P_j . Now we show that $\frac{\Phi(\mathbf{b}(P_j, 2 + \frac{\epsilon}{4}), C_i)}{\Phi(P_j, C_i)}$ is bounded by a constant.

► **Lemma 11.** *For any cluster $P_j \in \mathbb{P} \setminus \mathbb{O}_i$, $\frac{\Phi(\mathbf{b}(P_j, 2 + \frac{\epsilon}{4}), C_i)}{\Phi(P_j, C_i)} \geq \frac{3}{500}$.*

Proof. Define $Q(\alpha, \beta) = \frac{1}{2(\beta+1)} (4 \frac{\sqrt{\beta}}{\sqrt{\alpha}} + \beta + \ln \alpha - 4\sqrt{\beta} - \frac{\beta}{\alpha})$. It is easy to verify that $Q(2, \beta)$ increases monotonously with increasing value of β for $\beta \geq 2$. Therefore,

$$\frac{\Delta(C_i, \mathbf{b}(P_j, 2 + \frac{\epsilon}{4}))}{\Delta(C_i, P_j)} \geq \frac{\Delta(C_i, \mathbf{b}(P_j, 2))}{\Delta(C_i, P_j)} \geq Q(2, \beta_j) > Q(2, 2) > \frac{3}{500},$$

where the first step is derived from the fact that $\mathbf{b}(P_j, 2 + \frac{\epsilon}{4}) \subseteq \mathbf{b}(P_j, 2)$, the second step is due to Lemma 10, and the third step follows from the fact that $\beta_j > 2$, which is derived from Lemma 10. ◀

We now prove the correctness of Theorem 2.

► **Theorem 2.** *Given a point set P in a metric space and a parameter $0 < \epsilon \leq 1$, let C be a set of $O(\frac{1}{\epsilon}(k + m))$ points sampled from P using k -means++. Then, C contains a subset of k centers that induces a $(6 + \epsilon)$ -approximation for k -means with m outliers with constant probability.*

Proof. By Lemma 9, we know that if the current set of the points (sampled with k -means++) does not give the desired approximation ratio, the set of outliers Z or a cluster outside \mathbb{O}_i will be sampled with high probability. In the worst case scenario, we have to pick out k approximate centers for the clusters in \mathbb{P} and all the m outliers.

At each iteration of k -means++, we define a variable A_i . If the algorithm samples a point from Z or $\bigcup_{P_j \in \mathbb{P} \setminus \mathbb{O}_i} \mathbf{b}(P_j, 2 + \frac{\epsilon}{4})$, then $A_i = 1$; otherwise, $A_i = 0$. By the argument above, $A_i = 1$ implies that the algorithm succeeds in finding an outlier or a $(6 + \frac{\epsilon}{2})$ -approximation for the optimal center of a cluster in $\mathbb{P} \setminus \mathbb{O}_i$. By Lemma 9 and Lemma 11, we have $E[A_i] \geq \frac{3}{500} \cdot \frac{\epsilon}{53} = \frac{3\epsilon}{26500}$. Let $N = \frac{53000(k+m)}{3\epsilon}$, $A = \sum_{i=1}^N A_i$, and $u = \sum_{i=1}^N E[A_i]$. Using Lemma 5, we have $Pr(A \geq k+m) \geq 1 - Pr(A \leq \frac{1}{2}u) \geq 1 - e^{-k/4} \geq 1 - e^{-1/4}$. This implies that the set of $O(\frac{1}{\epsilon}(k+m))$ points sampled with D^2 -sampling contains a subset of k points that induces a $(6 + \epsilon)$ -approximation with a high constant probability, which completes the proof of Theorem 2. ◀

References

- 1 Manu Agarwal, Ragesh Jaiswal, and Arindam Pal. k -means++ under Approximation Stability. *Theoretical Computer Science*, 588:37–51, 2015.
- 2 Ankit Aggarwal, Amit Deshpande, and Raivi Kannan. Adaptive sampling for k -means clustering. In *Proc. 12nd Int. Workshop and 13rd Int. Workshop on Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 15–28, 2009.
- 3 Sara Ahmadian, Ashkan Norouzi-Fard, Ola Svensson, and Justin Ward. Better Guarantees for k -Means and Euclidean k -Median by Primal-Dual Algorithms. In *Proc. 58th IEEE Symposium on Foundations of Computer Science*, pages 61–72, 2017.
- 4 Nir Ailon, Ragesh Jaiswal, and Claire Monteleoni. Streaming k -means approximation. In *Proc. 23rd Annual Conference on Neural Information Processing Systems*, pages 10–18, 2009.
- 5 Mohammad Al Hasan, Vineet Chaoji, Saeed Salem, and Mohammed J Zaki. Robust partitional clustering by outlier and density insensitive seeding. *Pattern Recognition Letters*, 30(11):994–1002, 2009.
- 6 Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Papat. NP-hardness of Euclidean sum-of-squares clustering. *Machine Learning*, 75(2):245–248, 2009.
- 7 David Arthur and Sergei Vassilvitskii. k -means++: the advantage of careful seeding. In *Proc. 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1027–1035, 2007.
- 8 Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristic for k -median and facility location problems. In *Proc. 33rd Annual ACM Symposium on Theory of Computing*, pages 21–29, 2001.
- 9 Moses Charikar, Sudipto Guha, and David B. Shmoys. A constant-factor approximation algorithm for the k -median problem. In *Proc. 31st Annual ACM Symposium on Theory of Computing*, pages 1–10, 1999.
- 10 Moses Charikar, Samir Khuller, David M Mount, and Giri Narasimhan. Algorithms for facility location problems with outliers. In *Proc. 20th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 642–651, 2001.
- 11 Ke Chen. On k -Median clustering in high dimensions. In *Proc. 17th ACM-SIAM Symposium on Discrete Algorithms*, pages 1177–1185, 2006.
- 12 Ke Chen. A constant factor approximation algorithm for k -median clustering with outliers. In *Proc. 27th Annual ACM-SIAM Symposium on Discrete Algorithms*, volume 8, pages 826–835, 2008.
- 13 Vincent Cohen-Addad, Philip N. Klein, and Claire Mathieu. Local Search Yields Approximation Schemes for k -Means and k -Median in Euclidean and Minor-Free Metrics. In *Proc. 57th IEEE Annual Symposium on Foundations of Computer Science*, pages 353–364, 2016.
- 14 Dan Feldman and Leonard J. Schulman. Data reduction for weighted and outlier-resistant clustering. In *Proc. 31st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1343–1354, 2012.
- 15 Dan Feldman, Morteza Monemizadeh, and Christian Sohler. A PTAS for k -means clustering based on weak coresets. In *Proc. 23rd Annual Symposium on Computational Geometry*, pages 11–18, 2007.

- 16 Zachary Friggstad, Kamyar Khodamoradi, Mohsen Rezapour, and Mohammad R Salavatipour. Approximation schemes for clustering with outliers. In *Proc. 37th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 398–414, 2018.
- 17 Zachary Friggstad, Mohsen Rezapour, and Mohammad R. Salavatipour. Local Search Yields a PTAS for k -Means in Doubling Metrics. In *Proc. 57th IEEE Annual Symposium on Foundations of Computer Science*, pages 365–374, 2016.
- 18 Anupam Gupta and Kanat Tangwongsan. Simpler analyses of local search algorithms for facility location. *arXiv*, 2008. [arXiv:0809.2554](https://arxiv.org/abs/0809.2554).
- 19 Shalmoli Gupta, Ravi Kumar, Kefu Lu, Benjamin Moseley, and Sergei Vassilvitskii. Local search methods for k -means with outliers. *Proceedings of the VLDB Endowment*, 10(7):757–768, 2017.
- 20 Huang ingxiao, Jiang Shaofeng, Li Jian, and Wu Xuan. ϵ -Coresets for Clustering (with Outliers) in Doubling Metrics. In *Proc. 59th IEEE Annual Symposium on Foundations of Computer Science*, pages 814–825, 2018.
- 21 Kamal Jain, Mohammad Mahdian, and Amin Saberi. A new greedy approach for facility location problems. In *Proc. 34th Annual ACM Symposium on Theory of Computing*, pages 731–740, 2002.
- 22 Ragesh Jaiswal and Nitin Garg. Analysis of k -means++ for separable data. In *Proc. 15th Int. Workshop and 16th Int. Workshop on Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 591–602, 2012.
- 23 Byrka Jaroslaw, Pensyl Thomas, Rybicki Bartosz, Srinivasan Aravind, and Trinh Khoa. An Improved Approximation for k -Median and Positive Correlation in Budgeted Optimization. *ACM Transactions on Algorithms*, 13(2):23, 2017.
- 24 Ravishankar Krishnaswamy, Shi Li, and Sai Sandeep. Constant Approximation for k -Median and k -Means with Outliers via Iterative Rounding. In *Proc. 50th Annual ACM Symposium on Theory of Computing*, pages 646–659, 2018.
- 25 Amit Kumar, Yogish Sabharwal, and Sandeep Sen. Linear-time approximation schemes for clustering problems in any dimensions. *J. ACM*, 57(2):5:1–5:32, 2010.
- 26 Shi Li and Ola Svensson. Approximating k -median via pseudo-approximation. *SIAM Journal on Computing*, 45(2):530–547, 2012.
- 27 Stuart Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.
- 28 Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge University Press, 1995.
- 29 Rafail Ostrovsky, Yuval Rabani, Leonard J. Schulman, and Chaitanya Swamy. The effectiveness of Lloyd-type methods for the k -means problem. *J. ACM*, 59(6):28:1–28:22, 2013.
- 30 Dennis Wei. A constant-factor bi-criteria approximation guarantee for k -means++. In *Proc. 30th Annual Conference on Neural Information Processing Systems*, pages 604–612, 2016.

Unbounded Regions of High-Order Voronoi Diagrams of Lines and Segments in Higher Dimensions

Gill Barequet

Dept. of Computer Science, The Technion – Israel Inst. of Technology, Haifa 3200003, Israel
barequet@cs.technion.ac.il

Evanthia Papadopoulou 

Faculty of Informatics, Università della Svizzera italiana, Lugano, Switzerland
evanthia.papadopoulou@usi.ch

Martin Suderland 

Faculty of Informatics, Università della Svizzera italiana, Lugano, Switzerland
martin.suderland@usi.ch

Abstract

We study the behavior at infinity of the farthest and the higher-order Voronoi diagram of n line segments or lines in a d -dimensional Euclidean space. The unbounded parts of these diagrams can be encoded by a *Gaussian map* on the sphere of directions \mathbb{S}^{d-1} . We show that the combinatorial complexity of the Gaussian map for the order- k Voronoi diagram of n line segments or lines is $O(\min\{k, n-k\}n^{d-1})$, which is tight for $n-k = O(1)$. All the d -dimensional cells of the farthest Voronoi diagram are unbounded, its $(d-1)$ -skeleton is connected, and it does not have tunnels. A d -cell of the Voronoi diagram is called a tunnel if the set of its unbounded directions, represented as points on its Gaussian map, is not connected. In a three-dimensional space, the farthest Voronoi diagram of lines has exactly $n^2 - n$ three-dimensional cells, when $n \geq 2$. The Gaussian map of the farthest Voronoi diagram of line segments or lines can be constructed in $O(n^{d-1}\alpha(n))$ time, while if $d = 3$, the time drops to worst-case optimal $O(n^2)$.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Voronoi diagram, lines, line segments, higher-order, order- k , unbounded, hypersphere arrangement, great hyperspheres

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.62

Funding Gill Barequet: BSF Grant 2017684

Evanthia Papadopoulou: Swiss National Science Foundation, project SNF-200021E-154387

Martin Suderland: Swiss National Science Foundation, project SNF-200021E-154387

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. The *nearest* variant partitions the underlying space into maximal regions such that all points within one region have the same nearest site. The Euclidean Voronoi diagram of points in \mathbb{R}^d has been studied thoroughly, see, e.g., [7, 9, 13, 17]. This not the case, however, for non-point sites, which have been much less considered.

In the plane, many algorithmic paradigms, such as plane sweep, incremental construction, and divide-and-conquer have been applied to construct the Voronoi diagram of line segments in the plane [7]. However, in higher-dimensional spaces, results are quite sparse. Already in a three-dimensional space, the algebraic description of the features, such as the edges, of the Voronoi diagram of lines become very complicated [14]. As a result, the combinatorial



© Gill Barequet, Evanthia Papadopoulou, and Martin Suderland;
licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 62; pp. 62:1–62:15

Leibniz International Proceedings in Informatics



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

complexity of this diagram has been a major open problem in computational geometry [21]. There is a gap of an order of magnitude between the $\Omega(n^2)$ lower bound [3] and the only known upper bound of $O(n^{3+\epsilon})$ [25], where n is the number of sites. The gap carries over (and expands) to the Voronoi diagram of lines in d -space, $d \geq 3$, where the known bounds are $\Omega(n^{\lfloor \frac{d}{2} \rfloor})$ [4] and $O(n^{d+\epsilon})$ [25]. The lower bound is derived from n parallel lines whose Voronoi diagram has the same complexity as the Voronoi diagram of n points in $d-1$ dimensional space. For points in \mathbb{R}^d , the bound is $\Theta(n^{\lceil \frac{d}{2} \rceil})$ [7], and for $(d-2)$ -dimensional hyperplanes, the lower bound is $\Omega(n^{d-1})$ [3]. To the best of our knowledge, no other lower bound, other than $\Omega(n^{\lceil \frac{d}{2} \rceil})$, is available for line segments in \mathbb{R}^d , $d > 3$. Better combinatorial bounds are known only for some restricted cases [6, 10, 18, 19]. A numerically robust algorithm for computing the Voronoi diagram of lines in 3D has been given by Hemmer et al. [16].

The order- k (resp., farthest) Voronoi diagram of a set of sites is a partition of the underlying space into regions, such that the points of one region have the same k nearest sites (resp., same farthest site). Seidel [24] derived exact bounds on the maximal complexity of the Euclidean farthest Voronoi diagram of points in \mathbb{R}^d . Asymptotically, the worst-case complexity of the latter diagram remains $\Theta(n^{\lceil \frac{d}{2} \rceil})$. Edelsbrunner and Seidel [13] pointed out that the order- k Voronoi diagram of points in \mathbb{R}^d can be derived from the $\leq k$ -level of an arrangement of hyperplanes in \mathbb{R}^{d+1} . Agarwal and Mulmuley provided an algorithm which computes the $\leq k$ -level of n hyperplanes in \mathbb{R}^d in expected $O(n^{\lfloor \frac{d}{2} \rfloor} k^{\lfloor \frac{d}{2} \rfloor})$ time [1, 22]. For non-point sites, the problems have been mostly considered in the plane.

The farthest Voronoi diagram of n segments in the plane was first studied by Aurenhammer et al. [5], who gave results on its structure and an algorithm to compute it in $O(n \log n)$ time. The order- k counterpart of this diagram was then considered by Papadopoulou and Zavershynskiy [23], who showed that its complexity is $O(k(n-k))$, if segments are disjoint or touch only at endpoints, and that it can be constructed iteratively in $O(k^2 n \log n)$ time. If segments intersect, then the number of intersections affects the complexity only if $k < \frac{n}{2}$ [23]. These diagrams illustrate fundamental structural differences from their counterparts of points, such as disconnected Voronoi regions and no relation to convex hulls. Naturally, these differences carry over to higher dimensions, which is the subject of study in this paper.

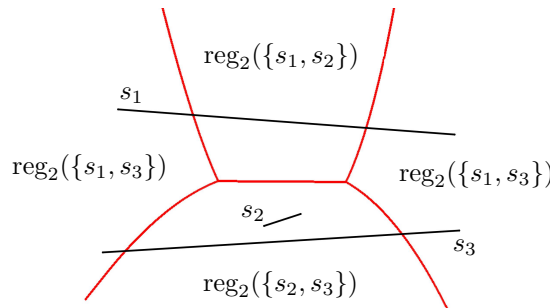
In three dimensions, the Euclidean farthest-site Voronoi diagram of lines or line segments has the property that all its three-dimensional cells are unbounded [8]. Barequet and Papadopoulou [8] used a structure on the sphere of directions, called the Gaussian map, which reflects the directions under which the cells of this diagram are unbounded.

In this paper, we study the Gaussian map of order- k and farthest Voronoi diagrams of n line segments and lines as sites in \mathbb{R}^d , and characterize the unbounded directions of the cells in these diagrams. The dimension d is assumed a constant. We derive the bound $O(\min\{k, n-k\}n^{d-1})$ on the complexity of the Gaussian map of order- k Voronoi diagrams for these sites. This implies the same upper bound on the complexity of the unbounded features of the corresponding order- k Voronoi diagrams. For the farthest-site diagram ($k = n-1$), this is $O(n^{d-1})$. For segments as sites, we prove that the complexity of the Gaussian map is $\Omega(k^{d-1})$, which is tight when $n-k = O(1)$. In fact, the complexity bound is derived by the number of vertices on the Gaussian map. This leads to a lower bound of $\Omega(k^{d-1})$ on the complexity of the entire order- k Voronoi diagram for line segments. For the farthest-site Voronoi diagram, this bound becomes $\Omega(n^{d-1})$, which also holds for lines as sites. As a byproduct, we derive a bound on the complexity of the arrangement of n great hyperspheres on \mathbb{S}^{d-1} .

■ **Table 1** Worst-case complexities of structures induced by a set S of n lines or segments in \mathbb{R}^d .

Structure	Lower bound	Upper bound
GM(VD $_k(S)$)	$\Omega(k^{d-1})^*$	$O(\min\{k, n-k\}n^{d-1})$
GM(FVD(S))	$\Omega(n^{d-1})$	$O(n^{d-1})$
VD $_k(S)$	$\Omega(k^{d-1})^*$	$O(\min\{k, n-k\}n^{d+\epsilon})$
FVD(S)	$\Omega(n^{d-1})$	$O(n^{d+\epsilon})$

*Only for segments.



■ **Figure 1** The order-2 Voronoi diagram (in red) of three segments s_1, s_2, s_3 in the plane.

Further, we describe a transformation that maps a set of lines to a set of segments, such that the two respective Gaussian maps of order- k Voronoi diagrams are identical. This transformation can be used to carry lower bounds from lines to segments and upper bounds from segments to lines. Table 1 summarizes most of the complexity results derived in this paper.

All the d -dimensional cells of the farthest Voronoi diagram of both lines and segments are unbounded, its $(d-1)$ -skeleton is connected, and it does not have tunnels. In three dimensions, the farthest Voronoi diagram of lines has exactly $n^2 - n$ many 3-dimensional cells, when $n \geq 2$.

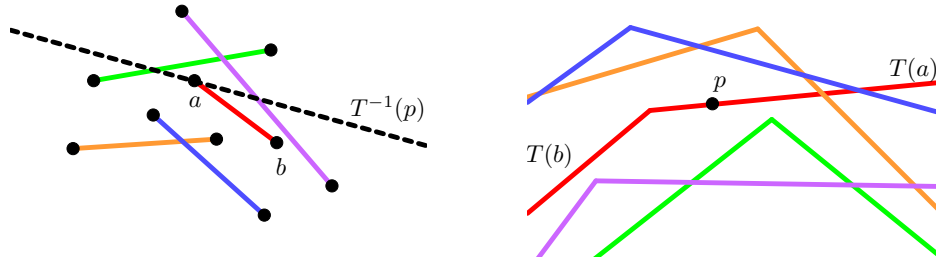
We show that we can compute the Gaussian map of this diagram in $O(n^{d-1}\alpha(n))$ time by using the algorithm of Edelsbunner et al. [12], which extends to higher dimensions [2, 15], for computing the envelope of piecewise-linear functions in \mathbb{R}^d . In fact, we conjecture that this bound can be improved to $O(n^{d-1})$. In three dimensions, we can compute the Gaussian map of the farthest Voronoi diagram of lines or segments in $O(n^2)$ time, which is optimal in the worst-case.

2 Preliminaries

2.1 Order- k Voronoi Diagrams

Let S be a set of sites in \mathbb{R}^d . In this paper, we consider as sites n (possibly intersecting) line segments or n lines in \mathbb{R}^d . The dimension d is considered constant. We denote by $d(x, y)$ the Euclidean distance between two points $x, y \in \mathbb{R}^d$. The distance $d(x, s)$ from a point $x \in \mathbb{R}^d$ to a site $s \in S$ is defined as $d(x, s) = \min\{d(x, s) \mid y \in s\}$.

► **Definition 1.** For a subset of sites $H \subset S$ of cardinality $|H| = k$, the order- k region of H is the set of points in \mathbb{R}^d whose distance to any site in H is smaller than to any site not in H . It is denoted as $\text{reg}_k(H) = \{p \in \mathbb{R}^d \mid \forall h \in H \forall s \in S \setminus H : d(p, h) \leq d(p, s)\}$.



■ **Figure 2** Point-hyperplane duality applied to segments: (left) Segments in primal space; and (right) their corresponding wedges in dual space.

The order- k regions of S induce a subdivision in \mathbb{R}^d . The induced cell complex is called the *order- k Voronoi diagram* of S , denoted by $\text{VD}_k(S)$. A maximally connected i -dimensional set of points, which is on the boundary of the same set of order- k regions, is called an i -dimensional cell of the cell complex. We call the i -dimensional cells of the order- k Voronoi diagram “ i -cells.”

When $k = 1$, this diagram is the well-known nearest-neighbor Voronoi diagram, denoted by $\text{VD}(S)$. For $k = n - 1$, it is the *farthest site Voronoi diagram*, denoted by $\text{FVD}(S)$. Its *farthest regions* can also be defined directly as $\text{freg}(h) = \{p \in \mathbb{R}^d \mid s \in S \setminus \{h\} : d(p, h) \geq d(p, s)\}$.

2.2 Point-Hyperplane Duality

Under the well-known point-hyperplane duality T in \mathbb{R}^d , a point $p \in \mathbb{R}^d$ is transformed to a non-vertical hyperplane $T(p)$, and vice versa. The transformation maps a point with coordinates (p_1, p_2, \dots, p_d) to the hyperplane $T(p)$ which satisfies the equation $x_d = -p_d + \sum_{i=1}^{d-1} p_i x_i$. The transformation is an involution, i.e., $T = T^{-1}$.

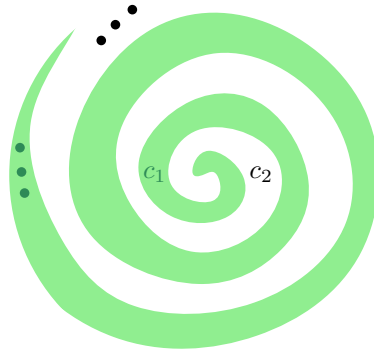
For a segment $s = uv$, the hyperplanes $T(u)$ and $T(v)$ partition the dual space into four *wedges*, among which the *lower wedge* (resp., the *upper wedge*) is the one that lies below (resp., above) both $T(u)$ and $T(v)$. The apex of the wedge is the intersection of $T(u)$ and $T(v)$.

Let S be a set of n segments, which in dual space corresponds to an arrangement of lower wedges. Let L_k be the k -th level of that arrangement. Let p be a point on L_k , which touches the dual wedge of segment $s = ab$, and let H be the set of segments whose wedges are below p , see Figure 2. Then, the point p corresponds to a hyperplane $T^{-1}(p)$ which touches the segment s . The closed halfspace above $T^{-1}(p)$ has a non-empty intersection with the segments in H . The open halfspace above $T^{-1}(p)$ does not intersect any segment in $S \setminus H$. We will use this property when we study the Gaussian map, which is defined in the next section. Symmetrically for the arrangement of upper wedges.

2.3 Levels in an arrangement of hyperplanes

This section reviews the definition of levels of an arrangement of surfaces, where those surfaces satisfy some *mildness* conditions (A1)-(A3) as given in [2]. We will use Theorem 2 by Clarkson and Shor several times in this paper.

The level of a point $p \in \mathbb{R}^d$ in an arrangement $\mathcal{A}(\Gamma)$ of a set Γ of surface patches is the number of surfaces of Γ lying vertically below p . For $0 \leq k < n$, the k -level (resp. $\leq k$ -level), denoted by $\mathcal{A}_k(\Gamma)$ (resp. $\mathcal{A}_{\leq k}(\Gamma)$), is the closure of all points on the surface of Γ whose level is k (resp. at most k). A face of $\mathcal{A}_k(\Gamma)$ or $\mathcal{A}_{\leq k}(\Gamma)$ is a maximal connected portion of a face of $\mathcal{A}(\Gamma)$ consisting of points having a fixed subset of surfaces lying below them. Let $\psi_k(\Gamma)$ (resp. $\psi_{\leq k}(\Gamma)$) be the total number of faces in $\mathcal{A}_k(\Gamma)$ (resp. $\mathcal{A}_{\leq k}(\Gamma)$). [2]



■ **Figure 3** A cell complex in which none of the cells is unbounded in a specific direction.

► **Theorem 2** (Clarkson and Shor [11]). *Let \mathcal{G} be an infinite family of surfaces satisfying some mildness assumptions (A1)-(A3) described in [2]. Then for any $0 \leq k < n - d$,*

$$\psi_{\leq k}(n, d, \mathcal{G}) = O\left((k + 1)^d \kappa\left(\frac{n}{k + 1}, d, \mathcal{G}\right)\right),$$

where $\kappa(n, d, \mathcal{G})$ is the maximum complexity of the lower envelope of n surfaces in \mathcal{G} .

It obviously holds that $\psi_k(\Gamma) \leq \psi_{\leq k}(\Gamma)$.

2.4 Defining the Gaussian Map

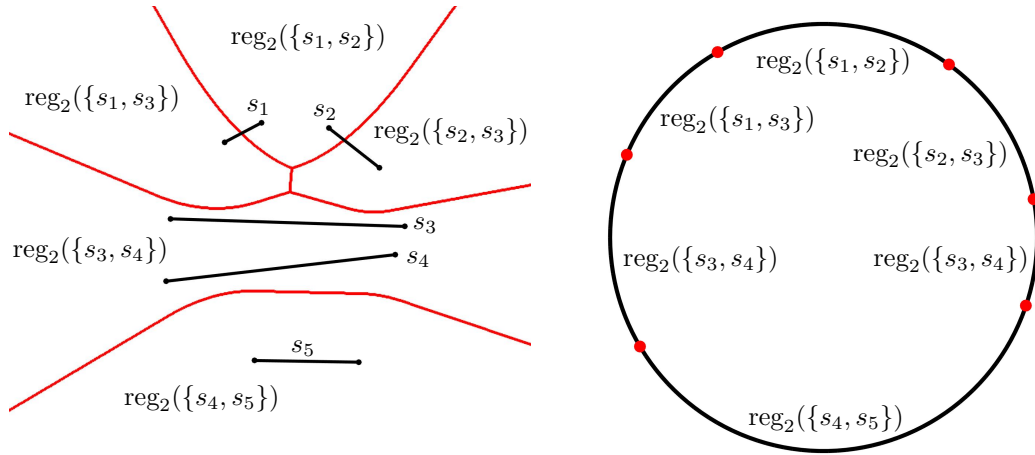
Let M be a cell complex in \mathbb{R}^d . The *complexity* of M is the total number of its cells of all dimensions. The *Gaussian map* of M encodes information about the unbounded cells of M . This structure is of particular interest when all cells of M are unbounded. For example, all the d -dimensional cells of the farthest Voronoi diagram of segments or lines are unbounded.

► **Definition 3.** *A cell c of M is unbounded in direction \vec{v} if in the limit $\lambda \rightarrow 0$, the intersection of the scaled cell $\lambda \cdot c$ and the unit sphere \mathbb{S}^{d-1} is non-empty in direction \vec{v} .*

The scaling of cell c can be done with an arbitrary center. The limit $\lim_{\lambda \rightarrow 0} (\lambda c \cap \mathbb{S}^{d-1})$ should be understood with the concept of the Kuratowski convergence [20], which we briefly review. For any point $x \in \mathbb{R}^d$ and subset $\mathcal{S} \subset \mathbb{R}^d$ let $d(x, \mathcal{S}) = \inf\{d(x, s) | s \in \mathcal{S}\}$ be the distance between x and \mathcal{S} . Let $\mathcal{S}_\lambda \subset \mathbb{R}^d$ be a sequence of compact sets. We say that \mathcal{S}_λ converges to \mathcal{S} for $\lambda \rightarrow 0$ iff $\mathcal{S} = \{x \in \mathbb{R}^d | \limsup_{\lambda \rightarrow 0} d(x, \mathcal{S}_\lambda) = 0\} = \{x \in \mathbb{R}^d | \liminf_{\lambda \rightarrow 0} d(x, \mathcal{S}_\lambda) = 0\}$.

Note that the Kuratowski limit does not always need to exist [20]. Consider a cell complex consisting of 2 cells circling around each other, see Figure 3. The unbounded directions of the cells of this cell complex would not be defined in this case, because for any cell $c \in \{c_1, c_2\}$ the sets $\{x \in \mathbb{R}^d | \limsup_{\lambda \rightarrow 0} d(x, \lambda c \cap \mathbb{S}^1)\} = \emptyset$ and $\{x \in \mathbb{R}^d | \liminf_{\lambda \rightarrow 0} d(x, \lambda c \cap \mathbb{S}^1) = 0\} = \mathbb{S}^1$ are not the same. In this paper we only consider cell complexes, where the unbounded directions of cells are well defined.

It might be tempting to use an alternative simpler definition: A cell c of M is unbounded in direction \vec{v} if it contains a ray with direction \vec{v} . However, this could cause problems for cells of dimension $< d$. For example, the trisector of three lines is in general a non-linear curve [14], containing no ray, therefore, it would not be unbounded in any direction. Thus Definition 3 is stronger in that sense and also defines unbounded directions for smaller dimensional cells.



■ **Figure 4** An order-2 Voronoi diagram $\text{VD}_2(\{s_1, s_2, \dots, s_5\})$ (left) and its Gaussian map (right).

► **Definition 4.** The Gaussian map of M , denoted by $\text{GM}(M)$, maps each cell in M to its unbounded directions, which are encoded on the unit sphere \mathbb{S}^{d-1} , see Figure 4. Let c be a cell of M ; the set of directions, in which c is unbounded, is called the region of c on $\text{GM}(M)$. The part of $\text{GM}(M)$ where the d -th coordinate is ≥ 0 (resp., ≤ 0) is called the upper (resp., lower) Gaussian map.

The Kuratowski limit is a closed set, if it exists, and therefore, cells of the Gaussian map are closed. In this paper, we focus on cell complexes, such as the farthest Voronoi diagram and the order- k Voronoi diagram of lines and segments, where cells have unbounded directions and the Gaussian map is the respective partition of \mathbb{S}^{d-1} . This partition induces a cell complex on \mathbb{S}^{d-1} . The collection of cells on the Gaussian map of a Voronoi diagram $\text{VD}_k(S)$, which correspond to the same set of sites $H \subset S$, is called the *region of H* on $\text{GM}(\text{VD}_k(S))$.

A Gaussian map region of a set of sites may consist of several $(d-1)$ -cells for two reasons: A region of a set of sites of VD_k may split into many d -cells, which all have unbounded directions on the Gaussian map. Moreover the Gaussian map region of just one d -cell of VD_k can consist of several cells, e.g., $\text{reg}_2(\{s_3, s_4\})$ in Fig. 4.

► **Definition 5.** A d -cell of the order- k Voronoi diagram is called a *tunnel* if its set of unbounded directions, represented as points on its Gaussian map, is not connected.

In Figure 4, one cell forms a *tunnel* in $\text{VD}_2(S)$.

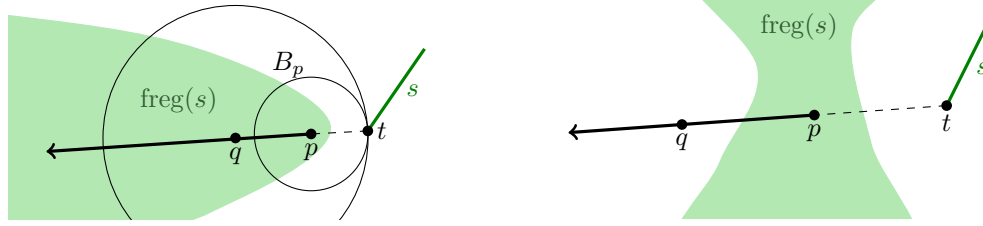
The Gaussian map essentially replaces the role of the convex hull in characterizing the unbounded regions the higher-order Voronoi diagram of $\text{VD}_k(S)$, for $k > 1$.

3 Properties of the Farthest and Order- k Voronoi Diagram

3.1 Combinatorial Properties

It has already been stated [8] that the complexity of the farthest Voronoi diagram is $O(n^{3+\varepsilon})$ by following the general bound of Sharir [25]. This bound generalizes for the order- k Voronoi diagram in \mathbb{R}^d .

► **Theorem 6.** The order- k Voronoi diagram of segments and lines in \mathbb{R}^d has complexity $O(\min\{k, n - k\}n^{d+\varepsilon})$.



■ **Figure 5** The farthest regions contain rays (left) and no farthest region can split the $d-1$ skeleton of the farthest Voronoi diagram into 2 parts (right).

Proof. Each site induces a distance function, which maps every point in \mathbb{R}^d to its distance to that site. The general framework of Sharir [25] shows that the complexity of the 0-level (resp., $(n-1)$ -level) of those distance functions is $O(n^{d+\epsilon})$. Applying Theorem 2 by Clarkson and Shor [11], the complexity of the $\leq k$ -level is $O(kn^{d+\epsilon})$ and $O((n-k)n^{d+\epsilon})$. ◀

In Section 4 we will prove the following lower bounds. These bounds are meaningful when k is comparable to n .

► **Theorem 7.** *The complexity of the order- k Voronoi diagram of segments in \mathbb{R}^d is $\Omega(k^{d-1})$ in the worst case. For the farthest Voronoi diagram ($k = n-1$), this is $\Omega(n^{d-1})$.*

3.2 Structural Properties

► **Lemma 8.** *Let S be a set of lines or segments, and let $p \in \text{freg}(s)$ be a point in the farthest region of site s . Let t be the point on s , which realizes the distance between s and p . Then, the entire ray \vec{r} , which emanates from p with direction \vec{tp} , is contained in $\text{freg}(s)$.*

Proof. The ball B_p , centered at p and of radius $|pt|$, touches s . Its interior intersects all other sites in S . In addition, any hyperball centered at any point $q \neq p$ along \vec{r} and of radius $|qt|$ must be properly enclosing B_p while touching s at t , see Figure 5. Thus, it must also intersect all sites in S except s . Therefore, $\text{freg}(s)$ must contain the entire ray \vec{r} . ◀

► **Corollary 9.** *Let S be a set of lines and segments. All d -cells of $\text{FVD}(S)$ are unbounded.*

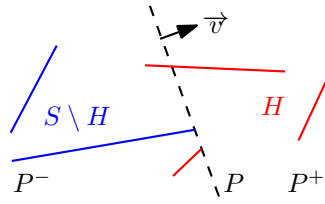
► **Remark 10.** The VD_k of segments can have bounded regions if $d \leq k \leq n-2$.

► **Definition 11.** *The i -skeleton of a cell complex M is the union of all j -cells in M with dimension $j \leq i$.*

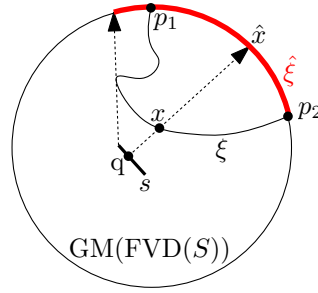
► **Theorem 12.** *Let S be a set of lines or segments in \mathbb{R}^d . The $(d-1)$ -skeleton of $\text{FVD}(S)$ is connected.*

Proof. Assume, for the sake of contradiction, that the diagram is not connected. Then, there exists a d -cell c that splits the $(d-1)$ -skeleton into at least two parts. Let s be the farthest site corresponding to c . The site s does not touch $\text{freg}(s)$. Let q be a point, which is separated from s by c . Let t be a point on s , which realizes the distance between q and s . Let p be a point on the segment \overline{qt} in $\text{freg}(s)$, see Fig. 5. Then, by Lemma 8, the entire ray \vec{r} , emanating from p in direction \vec{pq} , is contained in $\text{freg}(s)$. In particular, $q \in \text{freg}(s)$, which is a contradiction. ◀

► **Remark 13.** The $(d-1)$ -skeleton of $\text{VD}_k(S)$ need not be connected for $k \leq n-2$ and $S \subset \mathbb{R}^2$.



(a) A supporting hyperplane P (in dashed black) of sites H (in red) in direction \vec{v} .



(b) Construction of the path $\hat{\xi}$.

■ Figure 6

4 Line Segments as Sites

Let S be a set of line segments in \mathbb{R}^d . We assume that the segments are in general position, i.e. no $(d+1)$ segment endpoints lie on the same hyperplane. First, we characterize the segments that induce unbounded regions in the order- k Voronoi diagram in a given direction \vec{v} .

► **Definition 14.** Let S be a set of segments, and let H be a subset of S . A hyperplane P is called a supporting hyperplane of H in direction \vec{v} if

1. P is orthogonal to \vec{v} ;
2. The closed halfspace P^+ , bounded by P and unbounded in direction \vec{v} , intersects each of the sites in H ; and
3. The sites in $S \setminus H$ do not intersect the interior of P^+ , and at least one site in $S \setminus H$ touches P .

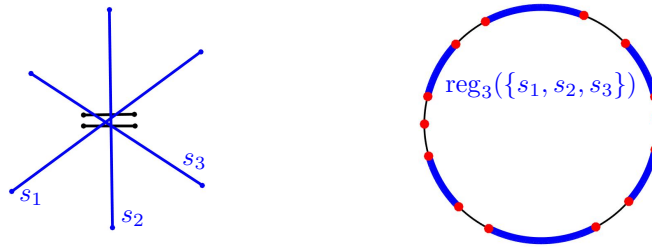
Figure 6a illustrates a hyperplane supporting three segments.

The following theorem is a generalization of results for the plane [5, 23].

► **Theorem 15.** A set of segments H , with $|H| = k$, induces an unbounded region in direction \vec{v} in the order- k Voronoi diagram of segments S , if and only if there exists a supporting hyperplane of H in direction \vec{v} .

Proof. Let H be a set of k segments, which has an unbounded d -cell c in direction \vec{v} in the order- k Voronoi diagram of a set of segments S . Each point in the cell corresponds to the center of a closed ball which has non-empty intersection with the segments in H , and does not intersect any of the other segments. By definition, there exists a curve unbounded in direction \vec{v} , which is contained in c . Any point p on that curve is the center of a closed ball, which has a non-empty intersection with the segments in H and does not intersect any of the other segments in its interior. When p moves along the curve to infinity, the ball around p becomes a halfspace which is orthogonal to \vec{v} . By moving the bounding hyperplane in direction $-\vec{v}$ until it hits a segment in $S \setminus H$, we can make it a supporting hyperplane.

Let P be a supporting hyperplane of segments H in direction \vec{v} . Let $H' \subseteq H$ be the subset of segments in H , which touch P . Let x be a point on P , which is closer to all endpoints of segments in H' than those which belong to other segments. Consider the ray r which emanates from x and is unbounded in direction \vec{v} . On that ray, we find a point y , which is the center of a closed ball, which touches x and intersects only the segments in H . Every point z on r beyond the point y has the same properties because the ball keeps growing on the side P^+ and shrinks on the other side. This means that all those points on r beyond y belong to the order- k region of the set H . ◀



■ **Figure 7** An instance of 5 segments (left), which has one region $\text{reg}_3(\{s_1, s_2, s_3\})$, shown in blue, on the Gaussian map of the order-3 Voronoi diagram (right) with high complexity.

► **Corollary 16.** *A supporting hyperplane of H in direction \vec{v} , which touches i segments (at least one of which is in H), corresponds to a $(d-i+1)$ -cell in $\text{VD}_k(S)$, which is unbounded in direction \vec{v} , and to a $(d-i)$ -cell in $\text{GM}(\text{VD}_k(S))$.*

► **Theorem 17.** *Let S be a set of segments. Then, $\text{FVD}(S)$ does not have tunnels.*

Proof. Let $p_1, p_2 \in \text{GM}(\text{FVD}(S))$ be two points representing unbounded directions of a farthest cell of segment s . These two points represent directions \vec{r}_1, \vec{r}_2 along which there exist points $x_1, x_2 \in \text{freg}(s)$, for which $\vec{r}_i = \overrightarrow{q_i x_i}$ with $(i = 1, 2)$, where q_i is the point on s realizing the distance between x_i and s . Since x_1 and x_2 are contained in the same cell $\text{freg}(s)$, there exists a continuous path ξ connecting the points and being fully contained in $\text{freg}(s)$. We can map every point $x \in \xi$ to the direction $\vec{r} = \overrightarrow{q x}$, with q realizing the distance between x and s . We represent direction \vec{r} as a point $p \in \text{GM}(\text{FVD}(S))$. Note that p is contained in a farthest cell of the Gaussian map corresponding to segment s . By continuity, mapping the whole path ξ to $\text{GM}(\text{FVD}(S))$ draws a continuous path $\hat{\xi}$ between p_1 and p_2 consisting solely of points that belong to s . Therefore, the points p_1 and p_2 belong to the same cell of the Gaussian map. ◀

► **Remark 18.** The order- k Voronoi diagram of segments S can have tunnels, for $k \leq n - d$.

The next theorem provides a lower bound on the complexity of the Gaussian map of order- k Voronoi diagrams. This bound is meaningful if k is comparable to n .

► **Theorem 19.** *Let S be a set of n line segments in \mathbb{R}^d . A single region of the Gaussian map of the order- k Voronoi diagram of S can have $\Omega(k^{d-1})$ many vertices. In particular, the $\text{GM}(\text{VD}_k(S))$ has $\Omega(k^{d-1})$ complexity in the worst-case.*

Proof. The bound is shown by a generalization of examples provided for \mathbb{R}^2 [5, 23]. Place k long segments connecting almost antipodal points on a $(d-1)$ -dimensional hypersphere and $n-k$ additional short segments near the center of the hypersphere, see Figure 7. Any $(d-1)$ -tuple of long segments, together with one specific short segment, define a supporting hyperplane corresponding to an unbounded edge of the order- k Voronoi diagram of S . The supporting hyperplane is spanned by an endpoint of each of the d segments. An unbounded edge of the diagram manifests itself as a vertex in $\text{GM}(\text{VD}_k(S))$. All these vertices are on the boundary of the Gaussian map region of the long segments. ◀

We can now prove Theorem 7.

62:10 Gaussian Map of order- k Voronoi Diagrams

Proof of Thm. 7. Let S be a set of n line segments in \mathbb{R}^d . In Theorem 19, it was stated that there can be $\Omega(k^{d-1})$ vertices in $\text{GM}(\text{VD}_k(S))$ in the worst-case. Each vertex of the Gaussian map corresponds to an edge in the $\text{VD}_k(S)$. On the other hand, an edge of the diagram corresponds to at most two vertices in the Gaussian map. Therefore, the diagram contains $\Omega(k^{d-1})$ edges. ◀

► **Theorem 20.** *The complexity of the Gaussian map of the order- k Voronoi diagram of n segments in \mathbb{R}^d is $O(\min\{k, n-k\}n^{d-1})$.*

Proof. We use the point-hyperplane duality transformation T , which establishes a 1-1 correspondence between the upper Gaussian map of the order- k Voronoi diagram and the k -th level of the arrangement of d -dimensional wedges. (The lower Gaussian map is constructed in the same manner.) Each segment is mapped to a lower wedge in the dual space, which is bounded by two half-hyperplanes. Let p be a point in dual space. Each wedge below p corresponds to a segment in primal space, which has a non-empty intersection with the open halfspace above $T^{-1}(p)$. Each wedge touching p corresponds to a segment in primal space, which is touching the closed halfspace above $T^{-1}(p)$. Each wedge above p corresponds to a segment in primal space, whose intersection with the closed halfspace above $T(p)$ is empty. Therefore, every point on the k -th level of the arrangement of the lower wedges corresponds to a hyperplane in primal space, which supports k segments. The upper or lower envelope of those wedges, each composed of two half-hyperplanes, has complexity $O(n^{d-1})$ [12].

Using the bound on the lower envelope, we can now also bound the complexity of the $\leq k$ -level of the arrangement of lower wedges. We apply Theorem 2 by Clarkson and Shor [11] to derive a complexity of $O((k+1)^d \binom{n}{k+1}^{d-1}) = O(kn^{d-1})$. We can derive a similar upper bound of $O((n-k)n^{d-1})$ by using the complexity of the upper envelope of lower wedges as a basis. The upper Gaussian map of the order- k Voronoi diagram corresponds to the k -level of the lower wedges. Combining the two bounds completes the proof. ◀

The bounds in Theorems 19 and 20 are tight for $n-k = O(1)$. In this case, the complexity of the Gaussian map of VD_k of n segments is $\Theta(n^{d-1})$ in the worst case.

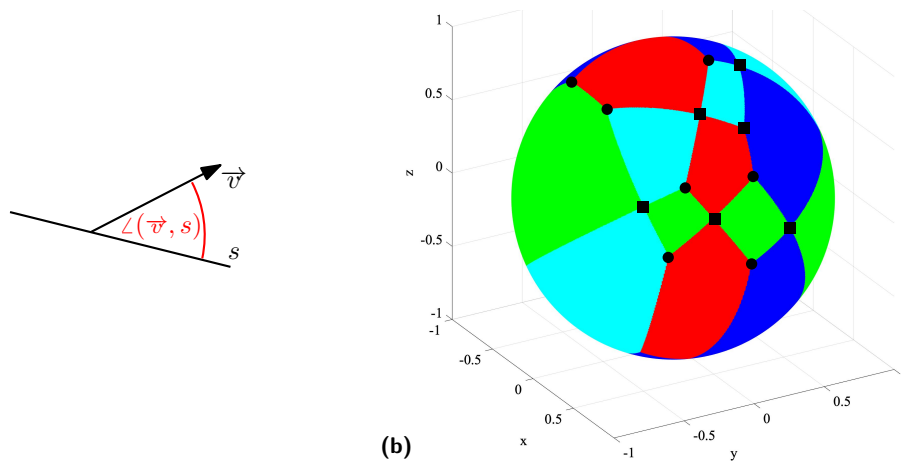
► **Theorem 21.** *Let S be a set of n line segments in \mathbb{R}^3 . Then, $\text{GM}(\text{FVD}(S))$ can be constructed in worst-case optimal $O(n^2)$ time.*

Proof. We dualize the segments into lower wedges. The upper Gaussian map of the segments corresponds to the upper envelope of the lower wedges in dual space (recall the proof of Thm. 20). The upper envelope of those wedges, each composed of two halfplanes, is constructed in $O(n^2)$ time [12]. The lower Gaussian map is constructed in the same way. ◀

The algorithm of Edelsbrunner et al. [12] for piecewise-linear functions can be extended to higher dimensions, running in $O(\alpha(n)n^{d-1})$ time [2, 15]. In fact, the complexity of the upper envelope of half-hyperplanes is only $O(n^{d-1})$ [12]. We suspect that the same algorithm runs in $O(n^{d-1})$ time when it computes the upper envelope of half-hyperplanes, as in \mathbb{R}^3 , since the complexity of the envelope does not contain the $\alpha(n)$ factor. If so, the Gaussian map of the farthest Voronoi diagram can be constructed in $O(n^{d-1})$ time.

5 Lines as Sites

Let S be a set of lines in \mathbb{R}^d . We assume that the lines are in general position, i.e., the lines are non-intersecting and the directions of any d lines are linearly independent. In this section we derive similar conditions for the order- k Voronoi diagram of lines to have unbounded cells in some direction. We omit proofs whose principles are similar to the ones of segments.



■ **Figure 8** (a) The angular distance $\angle(\vec{v}, s)$ between line s and direction \vec{v} . (b) GM(FVD) of four lines in \mathbb{R}^3 . The farthest regions of the lines are colored in different colors. Vertices of anomaly are shown with squared boxes; proper vertices with disks.

► **Definition 22.** For a line s and a direction \vec{v} , the angular distance $\angle(\vec{v}, s)$ is the smallest angle between \vec{v} and the direction of s , see Figure 8a.

► **Definition 23.** Let S be a set of lines, and let H be a subset of S . An angle β is a supporting angle of H in direction \vec{v} if

1. The angular distance between \vec{v} and any of the lines in H is at most β ; and
2. The angular distance between \vec{v} and any of the lines in $S \setminus H$ is at least β , and at least one site in $S \setminus H$ realizes the angular distance β .

► **Theorem 24.** A set of lines H , with $|H| = k$, induces an unbounded region in direction \vec{v} in $VD_k(S)$ if and only if there exists a supporting angle of H in direction \vec{v} .

The proof of the above theorem is essentially the same as that of Theorem 15, with a supporting hyperplane replaced by a supporting angle, and intersections with a halfspace replaced by angular distances.

► **Corollary 25.** A supporting angle of H , which is realized by i lines (at least one of which is in H), corresponds to an unbounded $(d-i+1)$ -cell in the order- k Voronoi diagram of S .

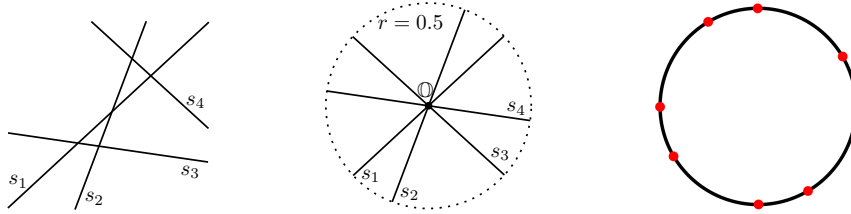
All d -cells, which are unbounded in the same direction \vec{v} , touch at a common cell. This cell is determined by the lines, which have the same angular distance to \vec{v} . A cell, which is equidistant to i lines, is $d-i+1$ -dimensional.

► **Theorem 26.** A supporting angle β of H in direction \vec{v} , which is realized by i lines (of which, at least one belongs to H), corresponds to a $(d-i)$ -cell (resp., $(d-i-1)$ -cell) in $GM(VD_k(S))$ if $\beta < \pi/2$ (resp., $\beta = \pi/2$).

Typically, i -cells of the Gaussian map correspond to $(i+1)$ -cells of the corresponding Voronoi diagram. The only exceptions are cells whose supporting angle is $\pi/2$, and, thus, they correspond to $(i+2)$ -cells of VD_k .

► **Definition 27.** The i -cells of the Gaussian map, $i < d-1$, which correspond to a supporting angle of $\pi/2$, are called cells of anomaly. All other cells are called proper.

62:12 Gaussian Map of order- k Voronoi Diagrams



■ **Figure 9** (Left) Lines S and their (center) transformed segments $\tau(S)$ have identical (right) Gaussian maps $\text{GM}(\text{VD}_2(S)) = \text{GM}(\text{VD}_2(\tau(S)))$.

In \mathbb{R}^3 , the only cells of anomaly are vertices, see Figure 8b. Such a vertex corresponds to a direction in which the bisector of two lines seems to be self-intersecting. The bisector of two lines s, s' is a hyperbolic paraboloid. Seen "from infinity" this hyperbolic paraboloid looks like two intersecting planes. The intersection of those planes is a line l , which is unbounded in two antipodal directions $-\vec{v}, \vec{v}$, which are the vertices of anomaly on the Gaussian map. One of the lines s, s' is actually strictly closer to direction \vec{v} than the other. Only "at infinity" both lines seem to have equal distance in direction \vec{v} .

In general space \mathbb{R}^d , the i -cells of anomaly on the Gaussian map correspond to $(i+2)$ -cells in the order- k Voronoi diagram. Looking at the Gaussian map, these $(i+2)$ -cells seem as if they intersect, however, they do not intersect in the actual diagram. Let \vec{v} be the direction of a cell of anomaly. The lines, which are orthogonal to \vec{v} , can actually be ordered along direction \vec{v} . Let j be the number of lines that are not orthogonal to \vec{v} . The region of those j lines, together with the closest $k-j$ orthogonal lines, is unbounded in direction \vec{v} and, moreover, is not split by an $(i+1)$ -cell in direction \vec{v} .

We define a transformation τ that maps lines to segments. Each line ℓ is mapped to a unit segment $\tau(\ell)$ that has the same direction as the line and the origin O as midpoint, see Figure 9. When applied to a set of lines, the result of the transformation is a set of segments in non-general position, but this does not affect the upper bound on the complexity of the Gaussian map.

► **Theorem 28.** *Let S be a set of lines. Then, $\text{GM}(\text{VD}_k(S)) = \text{GM}(\text{VD}_k(\tau(S)))$.*

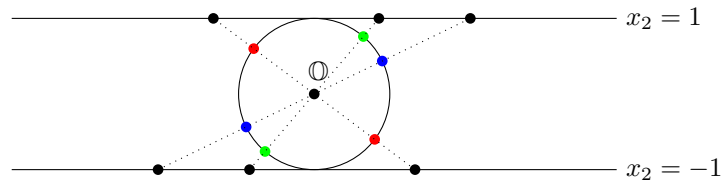
As a consequence, lower bounds on the worst-case complexity of the Gaussian map, derived for lines as sites, carry over to segments as sites. In the same manner, all upper bounds on the worst-case complexity on the Gaussian map for segments also apply to lines. In addition, the algorithm of Theorem 21 to construct the Gaussian map of the farthest Voronoi diagram extends to lines as sites. (Note that the algorithm does not require the segments to be in general position.)

► **Corollary 29.** *The Gaussian map of the order- k Voronoi diagram of n lines in \mathbb{R}^d has $O(\min\{k, n-k\}n^{d-1})$ complexity. The Gaussian map can be constructed in $O(n^{d-1}\alpha(n))$ time, while if $d = 3$, the time drops to $O(n^2)$.*

► **Theorem 30.** *Let S be a set of lines. Then, $\text{FVD}(S)$ does not have tunnels.*

A similar construction, as in Remark 18, can be used for showing that $\text{VD}_k(S)$ can have tunnels for a set of lines S and $k \leq n - d$.

The following result stands by its own and will be used to analyze the number of d -cells in the farthest Voronoi diagram of lines and its Gaussian map. We look at an arrangement of great spheres with same center and radius on a $(d-1)$ -sphere. For example, consider



■ **Figure 10** Example for $d = 2$ and $n = 3$: Three 0-dimensional unit spheres (blue, green, red) split the unit circle into 6 arcs.

the 2-dimensional unit sphere \mathbb{S}^2 in \mathbb{R}^3 and n great circles on it. We answer the following question: “Into how many 2-dimensional faces the unit sphere is split by the great circles?” We assume that no d great spheres have a point in common.

► **Theorem 31.** *Let \mathbb{S} be a set of n many $(d-2)$ -dimensional unit hyperspheres in \mathbb{R}^d , centered at the origin. Then, the arrangement of \mathbb{S} on the $(d-1)$ -dimensional unit hypersphere \mathbb{S}^{d-1} contains $\binom{n-1}{d-1} + \sum_{k=0}^{d-1} \binom{n}{k}$ many $(d-1)$ -cells.*

Theorem 31 can be proven by bijectively mapping the upper and lower hemisphere of \mathbb{S}^{d-1} to two parallel hyperplanes, see Figure 10. The $(d-2)$ -dimensional hyperspheres become hyperplanes of the same dimension. We add the $(d-1)$ -cells of each arrangement of $(d-2)$ -dimensional hyperplanes, while making sure that we do not count any cell twice.

► **Theorem 32.** *Let S be a set of n lines. The Gaussian map of $\text{FVD}(S)$ has $\Theta(n^{d-1})$ many $(d-1)$ -cells.*

Proof. We consider, for each line, the orthogonal directions. We get n many $(d-2)$ -dimensional hyperspheres in total. Each of those hyperspheres is partitioned into $\binom{n-2}{d-2} + \sum_{k=0}^{d-2} \binom{n-1}{k}$ parts by the other $(n-1)$ -hyperspheres due to Theorem 31. A direction in one of those parts is orthogonal to exactly one line in S and, hence, is also part of the farthest Voronoi region of that line. In total, all n hyperspheres are split into $n \left(\binom{n-2}{d-2} + \sum_{k=0}^{d-2} \binom{n-1}{k} \right) = \Theta(n^{d-1})$ parts. Now, consider a direction \vec{v} not on any hypersphere but in the farthest region of s . The shortest path on the Gaussian map from \vec{v} to the hypersphere corresponding to line s contains only directions of $\text{freg}(s)$. Therefore, there are no additional $(d-1)$ -cells not containing a part of a hypersphere. ◀

It is easy to prove that all cells of $\text{GM}(\text{FVD}(S))$ are convex, in the sense that the shortest path between any two points of a cell is contained in that cell.

For a set of lines S in \mathbb{R}^3 , we count the number of 2-cells of $\text{GM}(\text{FVD}(S))$ and subtract the number of vertices of anomaly to derive the exact number of 3-cells in $\text{FVD}(S)$.

► **Theorem 33.** *Let S be any set of $n \geq 2$ lines in \mathbb{R}^3 . Then, $\text{FVD}(S)$ has exactly $(n^2 - n)$ many 3-cells.*

An unbounded i -cell of a cell complex M may correspond to many $(i-1)$ -cells in the Gaussian map of M . Therefore, we need to study carefully the Gaussian map in order to derive a lower bound on the complexity of M .

► **Theorem 34.** *The worst-case complexity of FVD of n lines is $\Omega(n^{d-1})$.*

Proof. We bound the number of proper vertices (not those of anomaly) of $\text{GM}(\text{FVD}(S))$ from below. Those vertices correspond to unbounded edges of the farthest Voronoi diagram. The set of orthogonal directions to a line is a hypersphere of dimension $d-2$ in $\text{GM}(\text{FVD}(S))$. By

Theorem 31, the hyperspheres of all lines partition $\text{GM}(\text{FVD}(S))$ into $\binom{n-1}{d-1} + \sum_{k=0}^{d-1} \binom{n}{k} = \Omega(n^{d-1})$ many $(d-1)$ -dimensional parts. If $n \geq d$ (which is the case in the asymptotic analysis), each of those parts contains at least one proper vertex. Then, $\text{FVD}(S)$ has an unbounded edge in that direction. Each edge is unbounded in at most two directions. Hence, the number of edges can be bounded from below by half of the number of proper vertices of the Gaussian map. Thus, the number of edges in $\text{FVD}(S)$ is $\Omega(n^{d-1})$. ◀

6 Conclusion and Open Problems

We derived bounds on the complexity of the order- k Voronoi diagram and its Gaussian map, listed in Table 1. The results are tight for large values of k such as $k = n - 1$. Moreover we provided an algorithm to compute the Gaussian map of the farthest Voronoi diagram in three dimensional space in worst-case optimal time. It remains an open problem to determine whether or not the lower bounds on the complexity of VD_k and $\text{GM}(\text{VD}_k)$ for segments, as listed in Table 1, extend also to lines, when $k < n - 1$.

There is a gap between our lower and upper bounds on the complexity of the Gaussian map of the order- k Voronoi diagram. What is the correct bound and how can the diagram be constructed efficiently? This question is related to problem 3 in [21]: "What is the combinatorial complexity of the Voronoi diagram of a set of lines (or line segments) in three dimensions"?

We believe that knowing the structure of the Gaussian map of the order- k Voronoi diagram can help in analyzing the whole diagram. It may also be useful in constructing the full diagram. We leave this question for further research.

References

- 1 Pankaj K. Agarwal, Mark de Berg, Jirí Matousek, and Otfried Schwarzkopf. Constructing Levels in Arrangements and Higher Order Voronoi Diagrams. *SIAM J. Comput.*, 27(3):654–667, 1998. doi:10.1137/S0097539795281840.
- 2 Pankaj K. Agarwal and Micha Sharir. Arrangements and their applications. In *Handbook of computational geometry*, pages 49–119. Elsevier, 2000.
- 3 Boris Aronov. A lower bound on Voronoi diagram complexity. *Inf. Process. Lett.*, 83(4):183–185, 2002. doi:10.1016/S0020-0190(01)00336-2.
- 4 Boris Aronov. Personal communication, 2019.
- 5 Franz Aurenhammer, Robert L. S. Drysdale, and Hannes Krasser. Farthest line segment Voronoi diagrams. *Information Processing Letters*, 100(6):220–225, 2006. doi:10.1016/j.ipl.2006.07.008.
- 6 Franz Aurenhammer, Bert Jüttler, and Günter Paulini. Voronoi diagrams for parallel halflines and line segments in space. In Yoshio Okamoto and Takeshi Tokuyama, editors, *28th International Symposium on Algorithms and Computation, ISAAC 2017, December 9-12, 2017, Phuket, Thailand*, volume 92 of *LIPICs*, pages 7:1–7:10. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPICs.ISAAC.2017.7.
- 7 Franz Aurenhammer, Rolf Klein, and Der-Tsai Lee. *Voronoi diagrams and Delaunay triangulations*. World Scientific Publishing Company, 2013.
- 8 Gill Barequet and Evanthia Papadopoulou. On the Farthest-Neighbor Voronoi Diagram of Segments in Three Dimensions. In *10th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD)*, pages 31–36. IEEE, 2013.
- 9 Bernard Chazelle. An Optimal Convex Hull Algorithm and New Results on Cuttings (Extended Abstract). In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico*, pages 29–38. IEEE Computer Society, 1991. doi:10.1109/SFCS.1991.185345.

- 10 L. Paul Chew, Klara Kedem, Micha Sharir, Boaz Tagansky, and Emo Welzl. Voronoi diagrams of lines in 3-space under polyhedral convex distance functions. *J. Algorithms*, 29(2):238–255, 1998. doi:10.1006/jagm.1998.0957.
- 11 Kenneth L. Clarkson and Peter W. Shor. Applications of random sampling in computational geometry, II. *Discrete & Computational Geometry*, 4(5):387–421, 1989.
- 12 Herbert Edelsbrunner, Leonidas J. Guibas, and Micha Sharir. The Upper Envelope of Piecewise Linear Functions: Algorithms and Applications. *Discrete & Computational Geometry*, 4:311–336, 1989. doi:10.1007/BF02187733.
- 13 Herbert Edelsbrunner and Raimund Seidel. Voronoi diagrams and arrangements. *Discrete & Computational Geometry*, 1:25–44, 1986. doi:10.1007/BF02187681.
- 14 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.
- 15 Dan Halperin and Micha Sharir. Arrangements. *Handbook of discrete and computational geometry, third edition*, pages 723–762, 2017.
- 16 Michael Hemmer, Ophir Setter, and Dan Halperin. Constructing the Exact Voronoi Diagram of Arbitrary Lines in Three-Dimensional Space - with Fast Point-Location. In Mark de Berg and Ulrich Meyer, editors, *Algorithms - ESA 2010, 18th Annual European Symposium, Liverpool, UK, September 6-8, 2010. Proceedings, Part I*, volume 6346 of *Lecture Notes in Computer Science*, pages 398–409. Springer, 2010.
- 17 Victor Klee. On the complexity of d-dimensional Voronoi diagrams. *Archiv der Mathematik*, 34(1):75–80, 1980.
- 18 Vladlen Koltun and Micha Sharir. Three dimensional Euclidean Voronoi diagrams of lines with a fixed number of orientations. In Ferran Hurtado, Vera Sacristán, Chandrajit Bajaj, and Subhash Suri, editors, *Proceedings of the 18th Annual Symposium on Computational Geometry, Barcelona, Spain, June 5-7, 2002*, pages 217–226. ACM, 2002.
- 19 Vladlen Koltun and Micha Sharir. Polyhedral Voronoi Diagrams of Polyhedra in Three Dimensions. *Discrete & Computational Geometry*, 31(1):83–124, 2004. doi:10.1007/s00454-003-2950-5.
- 20 Kazimierz Kuratowski. *Topology*. Academic Press, 1966.
- 21 Joseph S. B. Mitchell and Joseph O’Rourke. Computational Geometry Column 42. *International Journal of Computational Geometry & Applications*, 11(5):573–582, 2001. doi:10.1142/S0218195901000651.
- 22 Ketan Mulmuley. On levels in arrangements and Voronoi diagrams. *Discrete & Computational Geometry*, 6(3):307–338, 1991.
- 23 Evanthia Papadopoulou and Maksym Zavershynskiy. The Higher-Order Voronoi Diagram of Line Segments. *Algorithmica*, 74(1):415–439, 2016. doi:10.1007/s00453-014-9950-0.
- 24 Raimund Seidel. On the Number of Faces in Higher-Dimensional Voronoi Diagrams. In D. Soule, editor, *Proceedings of the Third Annual Symposium on Computational Geometry, Waterloo, Ontario, Canada, 1987*, pages 181–185. ACM, 1987. doi:10.1145/41958.41977.
- 25 Micha Sharir. Almost Tight Upper Bounds for Lower Envelopes in Higher Dimensions. *Discrete & Computational Geometry*, 12:327–345, 1994. doi:10.1007/BF02574384.

Neighborhood Inclusions for Minimal Dominating Sets Enumeration: Linear and Polynomial Delay Algorithms in P_7 -Free and P_8 -Free Chordal Graphs

Oscar Defrain

Université Clermont Auvergne, France
oscar.defrain@uca.fr

Lhouari Nourine

Université Clermont Auvergne, France
lhouari.nourine@uca.fr

Abstract

In [M. M. Kanté, V. Limouzy, A. Mary, and L. Nourine. On the enumeration of minimal dominating sets and related notions. *SIAM Journal on Discrete Mathematics*, 28(4):1916–1929, 2014.] the authors give an $O(n + m)$ delay algorithm based on neighborhood inclusions for the enumeration of minimal dominating sets in split and P_6 -free chordal graphs. In this paper, we investigate generalizations of this technique to P_k -free chordal graphs for larger integers k . In particular, we give $O(n + m)$ and $O(n^3 \cdot m)$ delays algorithms in the classes of P_7 -free and P_8 -free chordal graphs. As for P_k -free chordal graphs for $k \geq 9$, we give evidence that such a technique is inefficient as a key step of the algorithm, namely the irredundant extension problem, becomes NP-complete.

2012 ACM Subject Classification Mathematics of computing → Graph enumeration

Keywords and phrases Minimal dominating sets, enumeration algorithms, linear delay enumeration, chordal graphs, forbidden induced paths

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.63

Acknowledgements The authors are supported by the ANR project GraphEn ANR-15-CE40-0009.

1 Introduction

We consider the problem of enumerating all inclusion-wise minimal dominating sets of a given graph, denoted by DOM-ENUM. A *dominating set* in a graph G is a set of vertices D such that every vertex of G is either in D or is adjacent to some vertex of D . It is said to be *minimal* if it does not contain any dominating set as a proper subset. To this date, it is open whether DOM-ENUM admits an output-polynomial time algorithm. An enumeration algorithm is said to be running in *output-polynomial* time if its running time is bounded by a polynomial in the combined size of the input and the output. It is said to be running in *incremental-polynomial* time if the running times between two consecutive outputs and after the last output are bounded by a polynomial in the combined size of the input and already output solutions. If the running times between two consecutive outputs and after the last output are bounded by a polynomial in the size of the input alone, then the algorithm is said to be running with *polynomial delay*; see [6, 10]. Recently, it has been proved in [12] that DOM-ENUM is equivalent to the problem of enumerating all inclusion-wise minimal transversals of a hypergraph, denoted by TRANS-ENUM. The best known algorithm for this problem is due to Fredman and Khachiyan [8] and runs in incremental quasi-polynomial time. Nevertheless, several classes of graphs were shown to admit output-polynomial time algorithms. For example, it has been shown that there exist output-polynomial time algorithms for $\log(n)$ -degenerate graphs [7], triangle-free graphs [3], and recently for K_t -free for any fixed $t \in \mathbb{N}$, diamond-free and paw-free graphs [2]. Incremental-polynomial time algorithms are known



© Oscar Defrain and Lhouari Nourine;

licensed under Creative Commons License CC-BY

30th International Symposium on Algorithms and Computation (ISAAC 2019).

Editors: Pinyan Lu and Guochuan Zhang; Article No. 63; pp. 63:1–63:16

Leibniz International Proceedings in Informatics



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

for chordal bipartite graphs [9] and graphs of bounded conformality [4]. Polynomial-delay algorithms are known for degenerate graphs [7], line graphs [14], and chordal graphs [13]. Linear-delay algorithms are known for permutation and interval graphs [11], graphs with bounded clique width [5], split and P_6 -free chordal graphs [12].

In this paper, we investigate the enumeration of minimal dominating sets from their intersection with redundant vertices, i.e., vertices that have an inclusion-wise non-minimal neighborhood in the graph. This technique was first introduced in [12] for the enumeration of minimal dominating sets in split and P_6 -free chordal graphs. We investigate generalizations of this technique to P_k -free chordal graphs for larger integers k . In particular, we give $O(n+m)$ and $O(n^3 \cdot m)$ delays algorithms in the classes of P_7 -free and P_8 -free chordal graphs, where n and m respectively denote the number of vertices and edges in the graph. Our algorithms rely on two main properties. The first one is that the intersections of minimal dominating sets with redundant vertices form an independence system and an accessible set system in P_7 -free and P_8 -free chordal graphs. The second is that the connected components obtained after removing redundant vertices in P_7 -free and P_8 -free chordal graphs are respectively P_3 -free and P_4 -free chordal. As for P_k -free chordal graphs for $k \geq 9$, we give evidence that such a technique is inefficient as a key step of the algorithm, namely the irredundant extension problem, becomes NP-complete.

The rest of the paper is organized as follows. In Section 2 we introduce definitions and preliminary notions. In Section 3 we describe the general algorithm that we consider throughout the paper and that can be decomposed into two distinct parts: redundant parts enumeration, and irredundant extensions enumeration. In Section 4 we prove properties on chordal graphs that depend on the size of a longest induced path in the graph. Section 5 is devoted to the complexity analysis of the first part of the algorithm, while Section 6 consider the second. We conclude in Section 7 by discussing the outlooks of such a technique.

2 Preliminaries

In this paper, all graphs are considered finite, undirected, simple, and loopless. For a graph $G = (V(G), E(G))$, $V(G)$ is its set of vertices and $E(G) \subseteq \{\{x, y\} \mid x, y \in V(G), x \neq y\}$ is its set of edges. Edges may be denoted by xy (or yx) instead of $\{x, y\}$. Two vertices x, y of G are called *adjacent* if $xy \in E(G)$. A *clique* in a graph G is a set of pairwise adjacent vertices. An *independent set* in a graph G is a set of pairwise non-adjacent vertices. The subgraph of G induced by $X \subseteq V(G)$, denoted by $G[X]$, is the graph $(X, E \cap \{\{x, y\} \mid x, y \in X, x \neq y\})$; $G - X$ is the graph $G[V(G) \setminus X]$. An *induced path* (resp. *induced cycle*) in G is a path (resp. cycle) that is an induced subgraph of G . We denote by P_k an induced path on k vertices. We call *hole* (or *chordless cycle*) an induced cycle of size at least four. A graph G is *split* if its vertex set can be partitioned into a clique and an independent set. It is *chordal* if it has no chordless cycle. It is called P_k -free if it has no induced path on k vertices.

Let G be a graph and $x \in V(G)$ be a vertex of G . The *neighborhood* of x is the set $N(x) = \{y \in V(G) \mid xy \in E(G)\}$. The *closed neighborhood* of x is the set $N[x] = N(x) \cup \{x\}$. For a subset $X \subseteq V(G)$ we define $N[X] = \bigcup_{x \in X} N[x]$ and $N(X) = N[X] \setminus X$. In case of ambiguity or when several graphs are considered, we shall note $N_G[x]$ the neighborhood of x in G . The *degree* of x is defined by $\deg(x) = |N(x)|$. We say that x is *complete* to X if $X \subseteq N(x)$, and that it is *partially adjacent* to X if it is adjacent to an element of X but not complete to X . Let $D, X \subseteq V(G)$ be two subsets of vertices of G . We say that D *dominates* X if $X \subseteq N[D]$. It is inclusion-wise *minimal* if $X \not\subseteq N[D \setminus \{x\}]$ for any $x \in D$. We say that D *dominates* x if it dominates $\{x\}$. A (minimal) *dominating set* of G is a (minimal)

dominating set of $V(G)$. The set of all minimal dominating sets of G is denoted by $\mathcal{D}(G)$, and the problem of enumerating $\mathcal{D}(G)$ given G by DOM-ENUM. Let x be a vertex of D . A *private neighbor* of x w.r.t. D in G is a vertex u of G that is only adjacent to x in D , that is, such that $N[u] \cap D = \{x\}$. Note that x can be its own private neighbor (in that case we say that x is *self-private*). The set of all private neighbors of x w.r.t. D is denoted by $Priv(D, x)$. It is well known that a subset $D \subseteq V(G)$ is a minimal dominating set of G if and only if it dominates G , and for every $x \in D$, $Priv(D, x) \neq \emptyset$.

Let x be a vertex of G . We say that x is *irredundant* if it is minimal with respect to neighborhood inclusion. In case of equality between minimal neighborhoods, exactly one vertex is considered as irredundant. We say that x is *redundant* if it is not irredundant. Then to every redundant vertex y corresponds at least one irredundant vertex x such that $N[x] \subseteq N[y]$, and no vertex y is such that $N[y] \subset N[x]$ whenever x is irredundant. The set of irredundant vertices of G is denoted by $IR(G)$, and the set of redundant vertices by $RN(G)$. We call *irredundant component* a connected component of $G[IR(G)]$. For a subset D of vertices of G we note $D_{RN} = D \cap RN(G)$ its intersection with redundant vertices, and $D_{IR} = D \cap IR(G)$ its intersection with irredundant vertices. Then D_{RN} and D_{IR} form a bipartition of D . For a subset D and a vertex $x \in D$, we call irredundant private neighbors of x w.r.t. D the elements of the set $Priv_{IR}(D, x) = Priv(D, x) \cap IR(G)$. In the remaining of the paper we shall note $\mathcal{D}_{RN}(G) = \{D_{RN} \mid D \in \mathcal{D}(G)\}$ and refer to this set as the *redundant parts* of minimal dominating sets of G . We call *irredundant extension* of $A \in \mathcal{D}_{RN}(G)$ a set $I \subseteq IR(G)$ such that $A \cup I \in \mathcal{D}(G)$, and note $DIR(A)$ the set of all such sets. Observe that $|\mathcal{D}_{RN}(G)| \leq |\mathcal{D}(G)|$ and that this inequality might be sharp (take a star graph), or strict (take a path on six vertices). We end the preliminaries stating general properties that will be used throughout the paper.

► **Proposition 1.** *Let G be a graph. Then $IR(G)$ dominates G , hence $\emptyset \in \mathcal{D}_{RN}(G)$.*

Proof. Take any vertex x of G . Either it is irredundant, or not. If it is then it is dominated by $IR(G)$. If not then by definition there exists $y \in IR(G)$ such that $N[y] \subseteq N[x]$, and it is dominated by $IR(G)$. Consequently, $IR(G)$ dominates G and thus there exists $D \subseteq IR(G)$ such that $D \in \mathcal{D}(G)$ and $D_{RN} = \emptyset$. Hence $\emptyset \in \mathcal{D}_{RN}(G)$. ◀

► **Proposition 2.** *Let G be a graph and $D \subseteq V(G)$. Then D is a minimal dominating set of G if and only if it dominates $IR(G)$ and $Priv_{IR}(D, x) \neq \emptyset$ for every $x \in D$.*

Proof. We prove the first implication. Let $D \in \mathcal{D}(G)$. Clearly D dominates $IR(G)$. Let us assume for contradiction that $Priv_{IR}(D, x) = \emptyset$ for some $x \in D$. We first exclude the case where x is self-private. If x is self-private then it is redundant and it has a neighbor $y \in IR(G)$ such that $N[y] \subseteq N[x]$. Since by hypothesis $Priv_{IR}(D, x) = \emptyset$, y is dominated by some $z \in D$, $x \neq z$. However, since $N[y] \subseteq N[x]$ then $zx \in E(G)$ and x is not self-private, a contradiction. Consequently x has a neighbor $u \in D$, and a private neighbor v in $RN(G)$. Let $w \in IR(G)$ such that $N[w] \subseteq N[v]$. Such a vertex exists since v is redundant. Two cases arise depending on whether $w = x$ or $w \neq x$. In the first case we conclude that $uv \in E(G)$, hence that v is not a private neighbor of x , a contradiction. In the other case, observe that since w is irredundant it cannot be a private neighbor of x (if ever it was adjacent to x). Hence it must be dominated by some $z \in D$, $z \neq x$ (possibly $z = w$). Since $N[w] \subseteq N[v]$, z is adjacent to v , hence v is not a private neighbor of x , a contradiction.

As for the other implication, observe that if an irredundant neighborhood $N[x]$, $x \in IR(G)$ is intersected by some set $D \subseteq V(G)$, then every neighborhood $N[y]$ such that $N[x] \subseteq N[y]$ is also intersected by D . Now if D dominates $IR(G)$, then it intersects every irredundant

63:4 Neighborhood Inclusions for Minimal Dominating Sets Enumeration

neighborhood. As for every $y \in RN(G)$ there exists $x \in IR(G)$ such that $N[x] \subseteq N[y]$ we conclude that D dominates G whenever it dominates $IR(G)$. Minimality follows from the inclusion $Priv_{IR}(D, x) \subseteq Priv(D, x)$, recalling that a dominating set D is minimal if and only if $Priv(D, x) \neq \emptyset$ for every $x \in D$. ◀

A corollary of Proposition 2 is the following, observing for $A \subseteq RN(G)$ and $I \subseteq IR(G)$ that if I dominates $IR(G) \setminus N(A)$ but not $Priv_{IR}(A, a)$ for any $a \in A$, then I can be arbitrarily reduced into a minimal such set.

► **Corollary 3.** *Let G be a graph and $A \subseteq RN(G)$. Then $A \in \mathcal{D}_{RN}(G)$ if and only if every $a \in A$ has an irredundant private neighbor, and there exists $I \subseteq IR(G)$ such that I dominates $IR(G) \setminus N(A)$ but not $Priv_{IR}(A, x) \neq \emptyset$ for any $a \in A$. Furthermore, $I \in DIR(A)$ whenever it is minimal with this property.*

3 The algorithm

We describe a general algorithm enumerating the minimal dominating sets of a graph from their intersection with redundant vertices. See Algorithm 1. The first step is the enumeration of such intersections, Line 2. The second step is the enumeration of their irredundant extensions, Line 3. The correctness of the algorithm follows from the bipartition induced by $RN(G)$ and $IR(G)$ in G .

The next sections are devoted to the complexity analysis of these two steps in the restricted case of P_7 -free and P_8 -free chordal graphs.

■ **Algorithm 1** An algorithm enumerating the minimal dominating sets of a graph G from their intersection with the set $RN(G)$ of redundant vertices of G .

```

1 Procedure DOM( $G$ )
2   for all  $A \subseteq RN(G)$  such that  $A \in \mathcal{D}_{RN}(G)$  do
3     for all  $I \subseteq IR(G)$  such that  $I \in DIR(A)$  do
4       output  $A \cup I$ ;
5     end
6   end

```

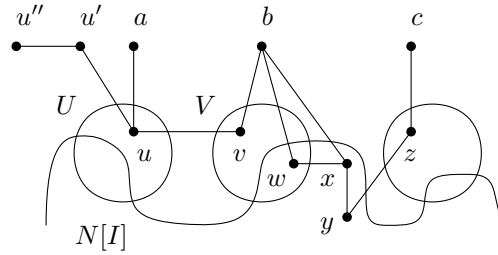
4 Properties on P_k -free chordal graphs

We give structural properties on redundant vertices and irredundant components of G whenever G is chordal, and depending on the size of a longest induced path in G .

► **Proposition 4.** *Let G be a graph and u, v be two adjacent irredundant vertices of G . Then there exist $u' \in N[u] \setminus N[v]$, $u'' \in N[u'] \setminus N[u]$, $v' \in N[v] \setminus N[u]$ and $v'' \in N[v'] \setminus N[v]$. In particular if G is chordal, then $u''u'uvv'v''$ induces a P_6 .*

Proof. Let us assume for contradiction that no such u' exists. Then either $N[u] \subset N[v]$, or $N[u] = N[v]$. In the first case v is redundant, a contradiction. In the other case only one of u and v should be irredundant by definition, a contradiction. Hence u' exists. By symmetry, v' exists. Let us now assume for contradiction that no such u'' exists. Then either $N[u'] \subset N[u]$, or $N[u'] = N[u]$. In the first case u is redundant, a contradiction. In the other case $vu' \in E(G)$, a contradiction. Hence u'' exists. By symmetry, v'' exists. Now if G is chordal, $u''u'uvv'v''$ induces a P_6 . ◀

■ **Figure 1** The situation of Proposition 5, case one. Circles denote private neighborhoods.



An *accessible set system* is a family of sets in which every non-empty set X contains an element x such that $X \setminus \{x\}$ belongs to the family. If x is of largest index in X such that $X \setminus \{x\}$ belongs to the family, then it is called *maximal generator* of X . An *independence system* is a family of sets such that for every non-empty set X of the family, and every element $x \in X$, $X \setminus \{x\}$ belongs to the family. In particular, every independence system is an accessible set system. Note that the maximal generator of X in that case is always the vertex of maximal index in X . Accessible set systems and independence systems play an important role in the design of efficient enumeration algorithms [1, 12]. The next theorem suggests that the enumeration of $\mathcal{D}_{RN}(G)$ is tractable in P_7 -free and P_8 -free chordal graphs.

► **Proposition 5.** *Let G be a chordal graph. Then $\mathcal{D}_{RN}(G)$ is an independence system whenever G is P_7 -free, and it is an accessible set system whenever G is P_8 -free.*

Proof. Let G be a chordal graph. We first assume that $\mathcal{D}_{RN}(G)$ is not an independence system to exhibit a P_7 , and then assume that $\mathcal{D}_{RN}(G)$ is not an accessible system to exhibit a P_8 . So suppose that $\mathcal{D}_{RN}(G)$ is not an independence system and let $A \in \mathcal{D}_{RN}(G)$ and $a \in A$ such that $A \setminus \{a\} \notin \mathcal{D}_{RN}(G)$. By Proposition 1, $|A| \geq 2$. Let $I \in DIR(A)$. Clearly $Priv_{IR}(A, a) \not\subseteq I$. Let $A' = A \setminus \{a\}$ and $I' = I \cup Priv_{IR}(A, a)$. Then I' dominates $IR(G) \setminus N(A')$. By Corollary 3 there must be some $b \in A'$ such that I' dominates $Priv_{IR}(A', b)$, hence $Priv_{IR}(A, b)$ as $Priv_{IR}(A, b) \subseteq Priv_{IR}(A', b)$. Let b be one such vertex. We put $U = Priv_{IR}(A, a) \setminus N[I]$ and $V = Priv_{IR}(A, b) \setminus N[I]$. Then neither of U nor V is empty, $U \cap V = \emptyset$, and U dominates V . Let $u \in U$ and $v \in V$ be such that $w \in E(G)$ (such u and v exist since U dominates V). Since u and v are private neighbors of a and b , $av, bu \notin E(G)$. Since G is chordal, $ab \notin E(G)$. Then $auvb$ induces a P_4 . By Proposition 4 since u, v are irredundant, there exists u'' and u' such that $u''u'wvb$ induces a P_5 . Consider an irredundant vertex w such that $N[w] \subseteq N[b]$. Such a vertex exists since b is redundant. Two cases arise depending on whether $w \in Priv_{IR}(A, b)$ or $w \notin Priv_{IR}(A, b)$.

Let us consider the case $w \in Priv_{IR}(A, b)$. It is illustrated in Figure 1. Since U dominates V and $N[w] \subseteq N[b]$ we know that $w \notin V$ (as otherwise b is adjacent to a vertex of U , i.e., a private neighbor of a). Hence $w \in Priv_{IR}(A, b) \cap N[I]$. Note that $w \notin I$ as $N[w] \subseteq N[b]$ (w cannot be part of an irredundant extension if it has no private neighbors). Accordingly, consider $x \in I$ such that $w \in E(G)$. Since $N[w] \subseteq N[b]$, $w \in E(G)$. Since $v \notin N[I]$, $xv \notin E(G)$. Now, since x belongs to I it has a private neighbor $y \in N[x] \setminus N[w]$. As G is chordal $u''u'wvbxxy$ induces a P_7 , concluding the first part of the proposition in this case. Let us now assume that $\mathcal{D}_{RN}(G)$ is not an accessible set system, that is $A \setminus \{c\} \notin \mathcal{D}_{RN}(G)$ for any $c \in A$. Observe that if replacing x by $Priv_{IR}(A' \cup I', x)$ in I' for all $x \in N(w) \cap I'$ does not dominate $Priv_{IR}(A', c)$ for any $c \in A' \setminus \{b\}$, then w becomes a private neighbor of b , and $A \setminus \{a\} \in \mathcal{D}_{RN}(G)$, a contradiction. Consequently there must exist $x \in I$ such that

$wx \in E(G)$ and $y \in \text{Priv}_{IR}(A' \cup I', x)$, $c \in A$ and $z \in \text{Priv}_{IR}(A', c)$ such that $yz \in E(G)$. Also $yb, zx, cy \notin E(G)$ as y and z are private neighbors of x and c . Since G is chordal, $u''u'ubvxyzc$ induces P_9 , concluding the second part of the proposition in this case.

Let us now consider the other case $w \notin \text{Priv}_{IR}(A, b)$. Then there must exist $c \in A \setminus \{b\}$ such that $wc \in E(G)$. Since $N[w] \subseteq N[b]$, we have $bc \in E(G)$. Consequently $a \neq c$. Furthermore since v is a private neighbor of b , $cv \notin E(G)$. Since $c \in A$ it has a private neighbor z , and $bz \notin E(G)$. As G is chordal $u''u'ubvcz$ induces a P_7 , concluding the first part of the proposition in this second case. Let us now assume that $\mathcal{D}_{RN}(G)$ is not an accessible set system. Then $A \setminus \{c\} \notin \mathcal{D}_{RN}(G)$. Observe that if every private neighbor z of c is such that $N[z] \subseteq N[c]$, then replacing c by every such private neighbors in $A \cup I$ yields a minimal dominating set D of G such that $D_{RN} = A \setminus \{c\}$, a contradiction. Hence there exist $z \in N[c] \setminus N[b]$ and $z' \in N[z] \setminus N[c]$. As G is chordal $u''u'ubvcz'z'$ induces a P_8 , concluding the second part of the proposition in this case, and the proof. \blacktriangleleft

► **Proposition 6.** *Let G be a chordal graph and C be an irredundant component of G . Then the graph $G[C]$ is P_{k-4} -free chordal whenever G is P_k -free, $k \geq 6$.*

Proof. We proceed by contradiction. Let G be a P_k -free graph, $k \geq 6$ and C be an irredundant component of G . Suppose that $G[C]$ is not P_{k-4} -free, and let P_{uv} be an induced path of length at least $k-4$ in $G[C]$ with endpoints u and v . Let u^* and v^* be the neighbors of u and v in P_{uv} (possibly $u^* = v$ and $v^* = u$, or $u^* = v^*$). By Proposition 4 since u, u^* and v, v^* are irredundant and adjacent, there exist u'', u', v', v'' such that $u''u'P_{uv}v''v''$ induces a path of length at least k in G , a contradiction. \blacktriangleleft

► **Proposition 7.** *Let G be a chordal graph, $a \in RN(G)$, C be an irredundant component of G , and u, v be two vertices in $C \cap N(a)$. Then $N(a)$ contains every induced path from u to v . In particular $G[N(a) \cap C]$ is connected.*

Proof. Clearly the proposition holds if $uv \in E(G)$. Let u, v be two non-adjacent vertices in $C \cap N(a)$. Let P_{uv} be an induced path from u to v in $G[C]$. One such path exists since $G[C]$ is connected. Let us assume for contradiction that there exists $x \in P_{uv}$ such that $x \notin N(a)$. Consider u^* and v^* to be the first elements of P_{uv} respectively in the way from x to u , and from x to v , such that $u^*, v^* \in N(a)$ (possibly $u^* = u$ and $v^* = v$). Consider the path $P_{u^*v^*}$ obtained from P_{uv} and shortened at endpoints u^* and v^* . Then $P_{u^*v^*}$ is an induced path with only its endpoints adjacent to a , inducing a hole in G , a contradiction. \blacktriangleleft

► **Proposition 8.** *Let G be a chordal graph and $a \in RN(G)$. Then a is partially adjacent to at most one irredundant component of G (it is either disconnected or complete to all other irredundant components of G) whenever G is P_9 -free chordal.*

Proof. We proceed by contradiction. Let us assume that G is P_9 -free chordal and that there exist two irredundant components C_1, C_2 such that $C_1 \cap N(a) \neq \emptyset$, $C_2 \cap N(a) \neq \emptyset$, and $C_1, C_2 \not\subseteq N(a)$. Let $u \in C_1 \cap N(a)$, $u' \in C_1 \setminus N(a)$, $v \in C_2 \cap N(a)$ and $v' \in C_2 \setminus N(a)$. Consider a shortest path $P_{u'u}$ in $G[C_1]$ from u' to u , and one $P_{vv'}$ in $G[C_2]$ from v to v' . These paths are induced. Let u^* and v^* be the neighbors of u' and v' in $P_{u'u}$ and $P_{vv'}$, respectively (possibly $u^* = u$ and $v^* = v$). By Proposition 4 since u', u^* and v', v^* are irredundant and adjacent, there exist u'', u''', v'' and v''' such that $u'''u''u'u^*$ and $v^*v''v''v'''$ induce paths of length four in G . Consider x the last vertex in $P_{u'u}$ starting from u which is adjacent to a , and y the last vertex in $P_{vv'}$ starting from v which is adjacent to a (possibly $x = u^*$ and $y = v^*$ but $x \neq u', y \neq v'$). Consider the paths $P_{u'x}$ and $P_{yv'}$ obtained from $P_{u'u}$ and $P_{vv'}$ and shortened at endpoints x and y . Then $u'''u''P_{u'x}aP_{yv'}v''v'''$ induces a path of length at least nine in G , a contradiction. \blacktriangleleft

In the following, for a set $A \subseteq RN(G)$ we consider the following bipartition. The part $B(A)$ contains the elements of A having an irredundant private neighbor in some irredundant component C such that $C \subseteq N(A)$. Observe that no irredundant extension of A can steal these private neighbors, as only $IR(G) \setminus N(A)$ has to be dominated by such extensions, and C is disconnected from $IR(G) \setminus N(A)$. The part $R(A)$ contains all other elements of A . We call *red* and *blue* vertices the elements of $R(A)$ and $B(A)$, respectively. If C_i is an irredundant component of G , then $R_i(A)$ denote the red elements of A having at least one private neighbor in C_i . Recall that by Proposition 8, the elements of A are partially adjacent to at most one irredundant component whenever G is P_9 -free chordal. In particular in such class, the red elements have their private neighbors in at most one irredundant component. The next theorem follows.

► **Theorem 9.** *Let G be a P_9 -free chordal graph, $A \in \mathcal{D}_{RN}(G)$ and $I \subseteq IR(G)$. Then I is an irredundant extension of A if and only if for every irredundant component C_i of G , $D_i = I \cap C_i$ is minimal such that*

- D_i dominates $C_i \setminus N(A)$, but
- D_i does not dominate $Priv_{IR}(A, x)$ for any $x \in R_i(A)$.

We immediately derive the next two corollaries, observing for the first one that a minimal set I as described in Theorem 9 can be greedily obtained from a non-minimal such set, and for the second that by Proposition 6, every irredundant component C of G is a clique whenever G is P_7 -free chordal.

► **Corollary 10.** *Let G be a P_9 -free chordal graph and $A \subseteq RN(G)$. Then $A \in \mathcal{D}_{RN}(G)$ if and only if every $a \in A$ has an irredundant private neighbor, and, for every irredundant component C_i of G there exists $D_i \subseteq C_i$ such that*

- D_i dominates $C_i \setminus N(A)$, but
- D_i does not dominate $Priv_{IR}(A, x)$ for any $x \in R_i(A)$.

► **Corollary 11.** *Let G be a P_7 -free chordal graph. Then $\mathcal{D}_{RN}(G) = \{A \subseteq RN(G) \mid \text{every } x \in A \text{ has a private neighbor in some irredundant component } C \subseteq N(A), \text{ i.e., } R(A) = \emptyset\}$.*

5 Enumerating the redundant part of minimal dominating sets

This section is devoted to the complexity analysis of Line 2 of Algorithm 1. More precisely, we show that enumerating the redundant part of minimal dominating sets can be done with linear and polynomial delays in P_7 -free and P_8 -free chordal graphs.

Recall that by Proposition 5, the set $\mathcal{D}_{RN}(G)$ is an accessible set system whenever G is P_8 -free chordal. Hence, it is sufficient to be able to decide whether (i) a given set $A \subseteq RN(G)$ belongs to $\mathcal{D}_{RN}(G)$, and (ii) a given vertex c of $A \in \mathcal{D}_{RN}$ is a maximal generator of A , in order to get an algorithm enumerating $\mathcal{D}_{RN}(G)$ without repetitions in such class. We call *irredundant extension problem* the first decision problem (denoted by IEP), and *maximal generator problem* the second (denoted by MGP). The algorithm proceeds as follows. See Algorithm 2. Given $A \in \mathcal{D}_{RN}(G)$ (starting with $A = \emptyset$ according to Proposition 1) it checks for every candidate vertex $c \in RN(G) \setminus A$ whether $A \cup \{c\}$ belongs to $\mathcal{D}_{RN}(G)$, whether c is a maximal generator of $A \cup \{c\}$, and if so, makes a recursive call on such a set. The correctness of the algorithm follows from the fact that $\mathcal{D}_{RN}(G)$ being an accessible set system, every set in $\mathcal{D}_{RN}(G)$ is accessible by such a procedure. In particular, every set A received by the algorithm belongs to $\mathcal{D}_{RN}(G)$. Repetitions are avoided by the choice of c .

■ **Algorithm 2** An algorithm enumerating the set $\mathcal{D}_{RN}(G)$ of a P_8 -free chordal graph G , relying on the fact that $\mathcal{D}_{RN}(G)$ is an accessible set system on such class.

```

1 Procedure RNDom( $G$ )
2   | RecRNDom( $G, \emptyset$ );
3 Procedure RecRNDom( $G, A$ )
4   | output  $A$ ;
5   | for all  $c \in RN(G) \setminus A$  do
6     |   if  $A \cup \{c\} \in \mathcal{D}_{RN}(G)$  and  $c$  is a maximal generator of  $A \cup \{c\}$  then
7       |   | RecRNDom( $G, A \cup \{c\}$ );
8       |   end
9   | end

```

5.1 Linear delay implementation in P_7 -free chordal graphs

We show that there is a linear-delay implementation of Algorithm 2 in P_7 -free chordal graphs. The proof is technically involved and makes use of preprocessed arrays that are maintained throughout the computation.

► **Theorem 12.** *There is an $O(n + m)$ delay, $O(n^2)$ space and $O(n^2)$ preprocessing-time implementation of Algorithm 2 whenever G is P_7 -free chordal, where n and m respectively denote the number of vertices and edges in G .*

Proof. Let C_1, \dots, C_ℓ denote the ℓ irredundant components of G . For every $a \in RN(G)$, and according to Proposition 8, we note $C^a = C_i$ the unique irredundant component C_i to which a is partially adjacent, if it exists, and $C^a = \emptyset$ otherwise. Note that the computation of such components, and the identification of C^a for every $a \in RN(G)$ can be done in $O(n^2)$ preprocessing time and takes $O(n^2)$ space. Consider $A \in \mathcal{D}_{RN}(G)$ as received by the algorithm. Let $c \in RN(G) \setminus A$. First observe that the condition of c being a maximal generator of $A \cup \{c\}$ Line 6 can be implicitly verified by selecting c of index greater than those in A . This can be done by computing the maximal index ρ in A before the loop in $O(n)$ time, and iterating on c such that $c > \rho$ with no extra cost on the complexity of the loop. We shall show using preprocessed arrays maintained at each step of the loop that testing whether $A \cup \{c\} \in \mathcal{D}_{RN}(G)$ is bounded by $O(\deg(c))$. Note that by Corollary 11, $A \cup \{c\}$ belongs to $\mathcal{D}_{RN}(G)$ if and only if (i) every $a \in A$ has a private neighbor in some irredundant component $C_j \subseteq N(A \cup \{c\})$, $j \in [\ell]$, and (ii) there exists C_i , $i \in [\ell]$ such that $C_i \not\subseteq N(A)$ and $C_i \subseteq N(A \cup \{c\})$. Also, recall that by Proposition 6 every component C_1, \dots, C_ℓ is a clique. Let T_1 be an array of size ℓ such that $T_1[i] = |C_i|$ for every $i \in [\ell]$. This array will be used to know the number of vertices that are yet to be dominated in every component. Let T_2 be an array of size n such that $T_2[y] = i$ if $y \in C_i$, and $T_2[y] = 0$ otherwise (if y is redundant). Using these two arrays, one can access in constant time to the number of vertices that are yet to be dominated in the unique clique C_i in which y belongs, by checking $T_1[T_2[y]]$. Let M_1 be a two dimensional array of size $n \times 2$ such that $M_1[a][0] = |Priv_{IR}(A, a) \cap C^a|$ if $C^a \neq \emptyset$, $M_1[a][0] = -1$ otherwise, and $M_1[a][1] = |Priv_{IR}(A, a) \setminus C^a|$. Let M_2 be an array of size n such that $M_2[y] = a$ if $y \in Priv_{IR}(A, a)$, and $M_2[y] = 0$ otherwise. Let M_3 be an array of size n such that $M_3[y] = 0$ if furthermore $y \in C^a$, $M_3[y] = 1$ otherwise. Using these three arrays, one can access in constant time to the number of irredundant private neighbors a vertex a such that $y \in Priv_{IR}(A, a)$ has by checking $M_1[M_2[y]][0]$ and $M_1[M_2[y]][1]$. The size of the set $Priv_{IR}(A, a) \cap C^a$ in case where $y \in C^a$, and $Priv_{IR}(A, a) \setminus C^a$ in case where $y \notin C^a$ can be accessed by $M_1[M_2[y]][M_3[y]]$. Finally, consider an array W of size n initialized to zero.

This array will be used to know if a vertex y is dominated by $A \cup \{c\}$, by setting $W[y] = x$ if y is connected to some $x \in A \cup \{c\}$ and $W[y] = 0$ otherwise. Note that these six arrays can be computed in $O(n^2)$ preprocessing time and $O(n^2)$ space.

We are now ready to detail each iteration of the loop Line 5. When considering a new candidate vertex $c \in RN(G) \setminus A$, we do the following. For each $y \in N(c) \cap IR(G)$, we set $W[y] := c$, $M_2[y] := c$ and $T_1[T_2[y]] := T_1[T_2[y]] - 1$ whenever $W[y] = 0$ (i.e., if y is not dominated by A). Note that $T_1[T_2[y]]$ is decreased to zero if and only if c verifies $C_i \not\subseteq N(A)$ and $C_i \subseteq N(A \cup \{c\})$ for $i = T_2[y]$. The next claim follows.

▷ **Claim 13.** Deciding whether there exists C_i , $i \in [\ell]$ such that $C_i \not\subseteq N(A)$ and $C_i \subseteq N(A \cup \{c\})$ takes $O(\deg(c))$ time.

If $W[y] \neq 0$ then y was already dominated by A , and in particular it might have been the private neighbor of some $a \in A$ given by both $M_2[y]$ and $W[y]$. In that case (i.e., whenever $M_2[y] \neq 0$) we set $M_1[M_2[y]][M_3[y]] := M_1[M_2[y]][M_3[y]] - 1$, and $M_2[y] := n + 1$ (this value is set temporarily). Note that $M_1[M_2[y]][0]$ (resp. $M_1[M_2[y]][1]$) is decreased to zero if and only if c steals all the private neighbors of $a \in A$ that are in irredundant components that are partially adjacent (resp. complete) to a . Also, observe that we still have $W[y] = a$ for all such y in that case. We prove the following

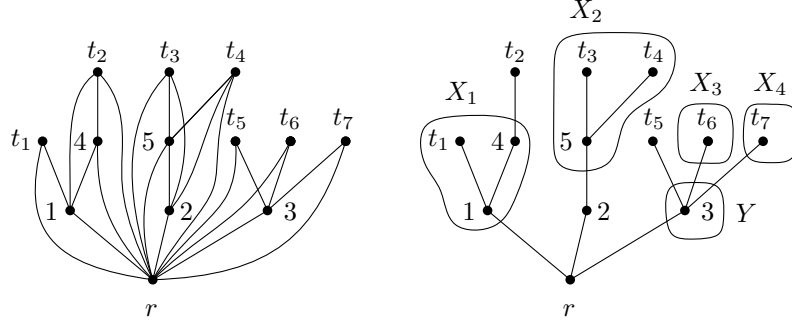
▷ **Claim 14.** Deciding whether every $a \in A$ has a private neighbor in some irredundant component $C_j \subseteq N(A \cup \{c\})$, $j \in [\ell]$ takes $O(\deg(c))$ time.

Proof. Consider some $y \in N(c) \cap IR(G)$ and let $a = M_2[y]$, $j = M_3[y]$. Observe that if both $M_1[a][0]$ and $M_1[a][1]$ have value zero after updating $M_1[a][j] := M_1[a][j] - 1$, then we answer negatively (a lost all its private neighbors). If $M_1[a][1]$ does not equal zero, then we answer positively (a has a private neighbor in a dominated irredundant component). If $M_1[a][1]$ equals zero and $M_1[a][0]$ does not equal zero, then we need to check whether C^a is dominated or not, that is whether $T_1[T_2[y]]$ equals zero or not. We answer positively if it is the case, and negatively otherwise. This covers all possibilities and the claim follows. ◁

A consequence of Claims 13 and 14 is that $A \cup \{c\} \in \mathcal{D}_{RN}(G)$ can be decided in $O(\deg(c))$ time in the condition of Line 6. Now, if $A \cup \{c\} \in \mathcal{D}_{RN}(G)$ then we set $M_2[y] = 0$ whenever $M_2[y] = n + 1$ (y is adjacent to some $a \in A$ and c , it is not a private neighbor anymore), for every $y \in N(c) \cap IR(G)$ and in a time which is also bounded by $O(\deg(c))$. Let us overview the case where c does not satisfy the conditions of Claims 13 and 14, or when a backtrack is executed. First, we undo the changes by setting $W[y] = 0$ and $T_1[T_2[y]] := T_1[T_2[y]] + 1$ for every $y \in N(c) \cap IR(G)$ such that $W[y] = c$ (such a y was not adjacent to any $a \in A$ and no other modifications occurred). If $W[y] \neq c$ and $M_2[y] = n + 1$ (in that case y was the private neighbor of some $a \in A$, $a = W[y]$) then we set $M_2[y] = W[y]$ and $M_1[M_2[y]][M_3[y]] = M_1[M_2[y]][M_3[y]] + 1$. If $W[y] \neq c$ and $M_2[y] \neq n + 1$ then y was not adjacent to c and no modification occurred. This undo process also takes $O(\deg(c))$ time. Since the sum of degrees of G is bounded by $O(n + m)$, the time spent in the loop Line 5 is bounded by $O(n + m)$.

Let us finally consider the case when consecutive backtracks are executed. Observe that in that case, it could be that n times $O(n + m)$ steps are computed without output. In order to avoid this, a common trick is to output half of the solutions while going down the recursive tree, and the other half when going up the tree. This is done by moving the output of Line 4 after the loop Line 9 on odd depths of the recursive tree. ◀

■ **Figure 2** A P_4 -free chordal graph H and the Hasse diagram of its poset tree. On such instance $p = 4$, $X_1 = \{t_1, 1, 4\}$, $X_2 = \{5, t_3, t_4\}$, $X_3 = \{t_6\}$, $X_4 = \{t_7\}$ and $Y = \{3\}$. Then $F = \{t_2, 2, t_5\}$ and a set D such that D dominates $C \setminus (X \cup Y)$ but not X_1, \dots, X_4 is given by $D = \{t_2, t_3, t_5\}$.



5.2 Polynomial delay implementation in P_8 -free chordal graphs

We show that IEP and MGP can be solved in polynomial time in P_8 -free chordal graphs. This yields a polynomial-delay implementation of Algorithm 2 in the same class.

From now on and until the end of the section, let G be a P_8 -free chordal graph. Recall that by Proposition 6, every irredundant component C of G induces a graph $H = G[C]$ that is P_4 -free chordal. It is known that every P_4 -free chordal graph is the comparability graph of a tree poset [17], where two vertices of the graph are made adjacent if they are comparable in the poset. To H we associate $T(H)$ its tree poset. Note that in particular, the root of $T(H)$ is universal in H , and that $x \leq y$ implies $N_H[y] \subseteq N_H[x]$. An example of a P_4 -free chordal graph and its tree poset is given in Figure 2.

In the following, let $A \subseteq RN(G)$ and C be an irredundant component of G . Let x_1, \dots, x_p denote the p elements of $R(A)$ having a private neighbor in C , $X_j = Priv_{IR}(A, x_j)$ for every $j \in [p]$, $X = X_1 \cup \dots \cup X_p$ and $Y = (N(A) \cap C) \setminus X$ (this last set corresponds to the vertices in C that are already dominated by A , but that are not private for any x_i , $i \in [p]$). By Corollary 10, IEP can be tested independently on every such component by checking whether X_1, \dots, X_p are non-empty, and, whether there exists $D \subseteq C$ such that

- D dominates $C \setminus (X \cup Y)$, but
- D does not dominate any of X_1, \dots, X_p .

We will show that such a test can be conducted in linear time whenever X_1, \dots, X_p and Y are given by lists and arrays, a condition that can be fulfilled at low cost as in the implementation of Theorem 21. In the remaining of this section, we note r the root of $T(H)$, and F the maximal elements of $T(H)$ which are neither in X_1, \dots, X_p nor in Y (hence no two elements of F are comparable in $T(H)$). One such instance is given in Figure 2.

► **Lemma 15.** *Let D be a subset of vertices of H . Then D dominates $C \setminus (X \cup Y)$ if and only if it dominates F .*

Proof. The first implication trivially holds since F is selected in $C \setminus (X \cup Y)$. Now since every vertex of $C \setminus (X \cup Y)$ belongs to a path in $T(H)$ from r to some $x \in F$, F dominates $C \setminus (X \cup Y)$. Consider any $x \in F$ and some dominating set D of F . Since D dominates F , either $x \in D$, or there exists $y \in D$ such that either $x < y$, or $x > y$. In all such cases, the unique path from r to x in $T(H)$ is dominated. Hence D dominates $C \setminus (X \cup Y)$. ◀

► **Lemma 16.** *There exists a set D dominating $C \setminus (X \cup Y)$ and not X_1, \dots, X_p if and only if for every $x \in F$ there exists a leaf t of $T(H)$ such that $x \leq t$ and $X_i \not\subseteq N_H[t]$, $i \in [p]$.*

Proof. We show the first implication. Let D be a dominating set of $C \setminus (X \cup Y)$ which does not dominate X_1, \dots, X_p . By Lemma 15, every $x \in F$ is dominated by some $y \in D$. Consider some such x and y . Then one of $x \leq y$ or $y < x$ holds. Let t be a leaf of $T(H)$ such that $y \leq t$ and $x \leq t$. Then t does not dominate any of X_1, \dots, X_p as y does not, and $y \leq t$ hence $N_H[y] \supseteq N_H[t]$. Since $x \leq t$ the first implication follows. As for the other implication, it is a consequence of Lemma 15 observing that every leaf t such that $x \leq t$ for some $x \in F$ dominates x . ◀

The next lemma shows that the characterization of Lemma 16 can be checked for each element $x \in F$ independently.

► **Lemma 17.** *Consider X_j , $j \in [p]$. Then, either*

- $X_j \subseteq \{y \in C \mid x < y\}$ for some unique $x \in F$, or
- $X_j \subseteq \{y \in C \mid y < x \text{ for some } x \in F\}$ and IEP can be answered negatively, or
- $X_j \not\subseteq N_H[F]$ and it can be ignored when checking the characterization of Lemma 16.

Proof. Let X_j , $j \in [p]$. First note that if $X_j \subseteq \{y \in C \mid x < y\}$ then such a x is unique or else $T(H)$ is not a tree. Let us assume that $X_j \subseteq \{y \in C \mid y < x \text{ for some } x \in F\}$. Observe that these two cases are disjoint as otherwise there exist two elements of F that are comparable, a contradiction. Recall that by Proposition 15 every dominating set D of $C \setminus (X \cup Y)$ dominates F . Now if D dominates F then it dominates every $y \in C$ such that $y < x$ for some $x \in F$, and X_j consequently. In that case IEP can be answered negatively.

Let us now assume that X_j is not of the first two cases. We first show by contradiction that there is no $x \in F$ such that $x < y$ for some $y \in X_j$. Suppose that there exist two such x and y . Since X_j is not of the first case there must be some y' , $y' \neq y$ such that $x \not< y'$. Consider $a \in R(A)$ such that $X_j = \text{Priv}_{IR}(A, a)$. Note that x belongs to a shortest path from y to y' in C (the common ancestor of y and y' in $T(H)$ must be smaller than x). By Proposition 7, x belongs to $N(a) \cap C$. Hence it either belongs to X_j , or Y , contradicting the fact that $x \in F$. Consequently, and since X_j is not of the second case, $X_j \not\subseteq N_H[F]$. Now, since the leaves of $T(H)$ selected in Lemma 16 are of neighborhood included in that of F , X_j can be ignored. ◀

► **Lemma 18.** *There is an $O(n + m)$ time algorithm solving IEP whenever G is P_8 -free chordal, where n and m respectively denote the number of vertices and edges in G , whenever*

- the leaves of $T(H = G[C])$,
- the predecessors and the successors of every $x \in T(H)$, and
- each of the sets X_1, \dots, X_p and Y

are given by lists and arrays for every irredundant component C of G .

Proof. Let us first focus on an irredundant component C of G , and $H = G[C]$ its induced subgraph. We want to decide whether F can be dominated without dominating X_1, \dots, X_p . Note that by assumption, the leaves of $T(H)$, the predecessors and the successors of every $x \in T(H)$, and each of the sets X_1, \dots, X_p and Y can be iterated in a time which is bounded by their size. Furthermore, deciding whether a vertex belongs to one given set takes constant time. The same assumptions hold for the set $Z = C \setminus (X \cup Y)$ which can be computed in $O(n_H)$ time iterating on X_1, \dots, X_p and Y .

The algorithm proceeds as follows. First it computes F by checking for every $x \in Z$ whether it has a successor in Z . It computes the set $F^- = \{y \in C \mid y < x \text{ for some } x \in F\}$ in a n -element array by adding predecessors of every $x \in F$ at a time. Since the sum of degrees of H is bounded by $O(n_H + m_H)$, this takes $O(n_H + m_H)$ time. Then it tests for

every set X_1, \dots, X_p whether it is included in F^- within the same time. At this stage if we find an inclusion then we can answer negatively according to the second item of Lemma 17, and can consider X_1, \dots, X_p to be of the first type in the following.

For every set $X_j, j \in [p]$ we check whether it has two non-adjacent vertices. This is done in $O(n_H + m_H)$ time by testing for every vertex in X_j whether it has a neighbor in X_j , recalling that every such X_j is disjoint (we iterate through vertices and their neighborhood only once). If X_j has no two non-adjacent vertices, then it is a path in $T(H)$ and we mark in a n_H -element array the indexes of leaves that are greater than its maximal element (each of these leaves dominates X_j). Similarly, computing the maximal element of every such X_j , and the indexes of leaves that are greater than their maximal element, can be done in $O(n_H + m_H)$ time. If a set $X_j, j \in [p]$ has two non-adjacent vertices then no leaf t of $T(H)$ dominates X_j and it can be ignored for the next step. We now proceed as follows according to Lemma 17. We check independently for every $x \in F$ if it has a descendant leaf t (to each $x \in F$ corresponds disjoint sets of such leaves) which was not indexed previously. If it has then we answer positively. If not then x (hence F) cannot be dominated without dominating one of X_1, \dots, X_p and we can answer negatively.

We now need to conduct this test for every irredundant component C of G independently. Since irredundant components are subgraphs of G we have that n and m are respectively bounded by the sums of n_H 's and m_H 's for every irredundant component C of G where $H = G[C]$, and the complexity follows. ◀

A corollary of Lemma 18 is the following, observing that x is a maximal generator of A if and only if $A \setminus \{y\} \notin \mathcal{D}_{RN}(G)$ for any $y \in A$ of index greater than x , and that n times $O(n + m)$ is bounded by $O(n \cdot m)$ since G is connected.

► **Corollary 19.** *There is an $O(n \cdot m)$ time algorithm solving MGP whenever G is P_8 -free chordal, where n and m respectively denote the number of vertices and edges in G , and assuming the conditions of Lemma 18.*

We can thus conclude the section with the following result.

► **Theorem 20.** *There is an $O(n^2 \cdot m)$ delay and $O(n^2)$ space implementation of Algorithm 2 whenever G is P_8 -free chordal, where n and m respectively denote the number of vertices and edges in G .*

Proof. By Lemma 18 and Corollary 19, there is an $O(n^2 \cdot m)$ delay implementation of Algorithm 2 whenever the assumptions of Lemma 18 can be fulfilled at every step of the loop Line 5. Clearly, the representation $T(H)$ of $H = G[C]$ can be computed for every irredundant component C of G in $O(n^2)$ preprocessing time, and $O(n^2)$ space. The lists and arrays containing the leaves of $T(H)$, the predecessors and the successors of every $x \in T(H)$, and that will contain the sets X_1, \dots, X_p, Y and $Z = C \setminus (X \cup Y)$ at each step of loop Line 5 can also be computed within these preprocessing-time and space complexities. Furthermore, the sets X_1, \dots, X_p and Y can be maintained at each step of loop as in the proof of Theorem 12, and in a time which is clearly upper-bounded by $O(n \cdot m)$ for each $c \in RN(G) \setminus A$. We proceed as in the proof of Theorem 12 to maintain an $O(n^2 \cdot m)$ delay in case of consecutive backtrack. The theorem follows. ◀

6 Enumerating irredundant extensions

This section is devoted to the complexity analysis of Line 3 of Algorithm 1. More precisely, we show that irredundant extensions can be enumerated with linear and polynomial delays in P_7 -free and P_8 -free chordal graphs. This allows us to conclude with the two main results of this paper.

6.1 Irredundant extensions in P_7 -free chordal graphs

Let G be a P_7 -free chordal graph and $A \in \mathcal{D}_{RN}(G)$. Recall that by Corollary 11, every $x \in A$ has a private neighbor in some irredundant component $C \subseteq N(A)$. Let C_1, \dots, C_k denote the k irredundant components of G that are not dominated by A . By Proposition 6, every such component is a clique. Consequently we have that

$$DIR(A) = \{\{x_1, \dots, x_k\} \mid x_i \in C_i, i \in [k]\}.$$

Now, such a set can clearly be enumerated with $O(n + m)$ delay given A and C_1, \dots, C_k . Furthermore, a track of these irredundant components is maintained at each step of the loop Line 5 of Algorithm 2 in the implementation of Theorem 12. We conclude with the following theorem which improves a previous result of Kanté et al. in [12] on P_6 -free chordal graphs.

► **Theorem 21.** *There is an $O(n + m)$ delay, $O(n^2)$ space and $O(n^2)$ preprocessing-time algorithm enumerating $\mathcal{D}(G)$ whenever G is P_7 -free chordal, where n and m respectively denote the number of vertices and edges in G .*

6.2 Irredundant extensions in P_8 -free chordal graphs

Let G be a P_8 -free chordal graph and $A \in \mathcal{D}_{RN}(G)$. By Theorem 9, the intersection of an irredundant extensions of A with an irredundant component C of G is a minimal set $D \subseteq C$ such that

- D dominates $C \setminus (X \cup Y)$, and
- D does not dominate any of X_1, \dots, X_p ,

where x_1, \dots, x_p denote the p elements of $R(A)$ having a private neighbor in C , where $X_j = Priv_{IR}(A, x_j)$ for every $j \in [p]$, $X = X_1 \cup \dots \cup X_p$ and $Y = (N(A) \cap C) \setminus X$. In the following, to A and C we associate $DIR(A, C)$ the set of all such minimal sets. Then, if C_1, \dots, C_k denote the k irredundant components of G that are not dominated by A , we have that

$$DIR(A) = \{D_1 \cup \dots \cup D_k \mid D_i \in DIR(A, C_i), i \in [k]\}.$$

Clearly, such a set can be enumerated with $O(n^3 \cdot m)$ delay given an algorithm enumerating $DIR(A, C_i)$ with $O(n^2 \cdot m)$ delay for every irredundant component C_1, \dots, C_k , where n and m respectively denote the number of vertices and edges in G . We shall show that such an algorithm exists.

Consider C , X_1, \dots, X_p and Y as described above. Let $H = G[C]$. Recall that by Proposition 6, H is P_4 -free chordal. In the remaining of this section, we rely on the notations of Section 5 and note r the root of $T(H)$ and F the maximal elements of $T(H)$ which are neither in X_1, \dots, X_p nor in Y . One such instance is given in Figure 2. We call *irredundant component extension problem*, denoted by ICEP, the following decision problem. Given $S, Q \subseteq C$, is there a solution $D \in DIR(A, C)$ such that $S \subseteq D$ and $D \cap Q = \emptyset$? We shall show that this problem can be solved in $O(n \cdot m)$ time, which, using the *backtrack search* technique, leads to an $O(n^2 \cdot m)$ algorithm enumerating irredundant extensions in P_8 -free chordal graphs.

► **Lemma 22.** *There is an algorithm solving ICEP in $O(n \cdot m)$ time, assuming the conditions of Lemma 18.*

Proof. Observe that IEP restricted to a single component and ICEP only differ on the fact that the set $D \subseteq C$ should in addition satisfy $D \cap Q = \emptyset$ and should not dominate $Priv_H(S, s) \setminus (X \cup Y)$ for any $s \in S$, where $Priv_H(S, s)$ denotes the private neighborhood

of $s \in S$ in H . In that case, D can be reduced to a minimal set D^* such that $S \subseteq D^*$ and $D^* \cap Q = \emptyset$. We show that these additional conditions can be handled at the cost of an increasing complexity, relying on the proof of Lemma 18. Clearly, we can first answer negatively if $\text{Priv}_H(S, s) \setminus (X \cup Y)$ is empty for some $s \in S$. Otherwise, the condition that D does not dominate $\text{Priv}_H(S, s) \setminus (X \cup Y)$ for any $s \in S$ can be handled by adding extra sets $X_{p+i} = \text{Priv}_H(S, s_i) \setminus (X \cup Y)$ for every $s_1, \dots, s_q \in S$, and updating $X := X \cup X_{p+1} \cup \dots \cup X_{p+q}$. Since these sets are connected Lemma 17 still applies. As for D satisfying $D \cap Q = \emptyset$, we proceed as follows. For every $x \in F$, and instead of only checking descendant leaves of x , we iterate through all the descendants of x and check whether it has a successor y such that $y \notin Q$, and such that y does not dominate any of X_1, \dots, X_{p+q} . This can be done in $O(n \cdot m)$ time as we iterate through every such y in $O(n + m)$ time, and check for each of these y 's whether it has some X_j in its neighborhood in $O(n)$ time. At this stage, and according to Lemmas 16, 17 and 18, we can answer yes if and only every x has such a neighbor y . ◀

We can now conclude using the backtrack search technique that we briefly recall now. Formal proofs are omitted. The enumeration is a depth-first search of a tree whose nodes are partial solutions and leaves are solutions. The algorithm constructs partial solutions by considering one vertex x_i at a time (following some linear ordering x_1, \dots, x_n of the vertices), checking at each step whether there is a final solution $D_1 \in \text{DIR}(A)$ containing $S \cup \{x_i\}$ and not intersecting Q , and one $D_2 \in \text{DIR}(A)$ containing S and not intersecting $Q \cup \{x_i\}$. This step is called the extension problem. The algorithm recursively calls on such sets each time the extension is possible. At first, S and Q are empty. The delay time complexity is bounded by the depth of the tree (the number of vertices) times the time complexity of solving the extension problem, i.e., ICEP. For further details on this technique, see for instance [15, 16].

We conclude to the next lemma and theorem, noting that the conditions of Lemma 18 can be fulfilled as in the proof of Theorems 12 and 20.

► **Lemma 23.** *There is an algorithm enumerating $\text{DIR}(A, C)$ with $O(n^2 \cdot m)$ delay, assuming the conditions of Lemma 18.*

► **Theorem 24.** *There is an $O(n^3 \cdot m)$ delay and $O(n^2)$ space algorithm enumerating $\mathcal{D}(G)$ whenever G is P_8 -free chordal, where n and m denote the number of vertices and edges in G .*

7 Discussions

We investigated the enumeration of minimal dominating sets from their intersection with redundant vertices. This technique was first introduced in [12] and led to linear-delay algorithms in split and P_6 -free chordal graphs. We investigated generalizations of this technique to P_k -free chordal graphs for larger integers k . In particular, we gave $O(n + m)$ and $O(n^3 \cdot m)$ delay algorithms in the classes of P_7 -free and P_8 -free chordal graphs, where n and m respectively denote the number of vertices and edges in the graph.

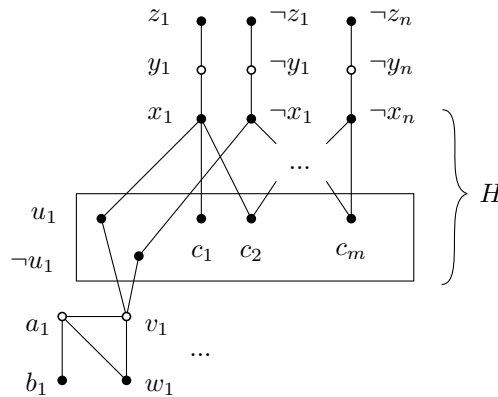
As for P_k -free chordal graphs for $k \geq 9$, we now give evidence that the enumeration of $\mathcal{D}_{RN}(G)$ might need other techniques for $k \geq 9$, as IEP becomes NP-complete.

► **Theorem 25.** *IEP is NP-complete even when restricted to P_9 -free chordal graphs.*

Proof. First notice that IEP belongs to NP: a polynomial certificate is given by an irredundant set $I \subseteq \text{IR}(G)$ such that $A \cup I \in \mathcal{D}(G)$, and which can be verified in polynomial time.

Given an instance φ of 3SAT with variables x_1, \dots, x_n and clauses C_1, \dots, C_m , we construct a P_9 -free chordal graph G and a set $A \subseteq \text{RN}(G)$ such that A admits an irredundant extension if and only if there exists a truth assignment of the variables of φ that satisfies all

■ **Figure 3** The construction of G in Theorem 25. Irredundant vertices are represented in black while redundant ones are in white. The vertices in the rectangle induce a clique and H a split graph.



the clauses. In the following, we assume that the degenerate cases where a literal intersects every clause, where two clauses are equal, or where the number of variables and clauses is lesser than three are excluded. Then, the construction is the following.

The first part concerns the construction of a split graph H which contains one vertex for each of the literals x_i and $\neg x_i$, a copy u and $\neg u_i$ of such literals, and one vertex c_j per clause C_j . The graph induced by the u_i 's, $\neg u_i$'s and c_j 's is completed into a clique, while an edge is added between u_i and x_i , between $\neg u_i$ and $\neg x_i$, and between a literal x_i (resp. $\neg x_i$) and a clause c_j whenever the literal is contained into that clause. As for the second part, it consists of a pendant path $x_i y_i z_i$ and $\neg x_i \neg y_i \neg z_i$ rooted at every literal x_i and $\neg x_i$, and of a paw $a_i b_i v_i w_i$ (a triangle $a_i v_i w_i$ with a pendant edge $a_i b_i$) made adjacent to both u_i and $\neg u_i$ only through v_i , for every $i \in [n]$. The construction is illustrated in Figure 3. It can be easily seen that the obtained graph G is P_9 -free chordal. Also that a P_8 is induced by $b_i a_i v_i u_i u_j v_j a_j b_j$ for $i \neq j \in [n]$.

Let us show that $H = G[C]$ for an irredundant component C of G . First note that every vertex outside of H has a neighbor that is not adjacent to H , so it cannot make a vertex from H redundant. Now if a vertex of H makes another one of H redundant then it cannot be a literal or some $u_i, \neg u_i$ (as it has either $y_i, \neg y_i$ or v_i as a neighbor outside of H). Also, it cannot be a clause as by assumption every two clauses differ on a literal, and no literal is complete to the clique. Hence vertices of H are all irredundant. It is easily seen that irredundant components of G include $\{z_i\}, \{\neg z_i\}, \{b_i\}$ and $\{w_i\}$ for all $i \in [n]$. Also that redundant vertices of G are a_i 's, v_i 's, y_i 's and $\neg y_i$'s. We conclude that H cannot be extended, hence that C is indeed an irredundant component of G , as claimed.

Now, let us set $A = RN(G)$ and show that $A \in \mathcal{D}_{RN}(G)$ if and only if there exists a truth assignment of the variables of φ that satisfies all the clauses. If $A \in \mathcal{D}_{RN}(G)$ then there exists an irredundant extension $D \in DIR(A)$. Observe that only the c_i 's are to be dominated by D , i.e., $IR(G) \setminus N(A) = \{c_1, \dots, c_m\}$. However, D does not intersect any element of the clique of H as otherwise it would dominate $\{u_i, \neg u_i\}$ and thus steal all the private neighbors of the v_i 's. For the same reason, it cannot contain one literal and its negation. Consequently D corresponds to a truth assignment of the variables of φ that satisfies all the clauses. Consider now any truth assignment of the variables of φ that satisfies all the clauses, and D the associated set of vertices in G . By construction D dominates all the c_i 's. Furthermore it does not steal any private neighbor to any vertex of A . By Corollary 3 we have that $A \in \mathcal{D}_{RN}(G)$ concluding the proof. ◀

References

- 1 Hiroki Arimura and Takeaki Uno. Polynomial-delay and polynomial-space algorithms for mining closed sequences, graphs, and pictures in accessible set systems. In *Proceedings of the 2009 SIAM International Conference on Data Mining*, pages 1088–1099. SIAM, 2009.
- 2 Marthe Bonamy, Oscar Defrain, Marc Heinrich, Michał Pilipczuk, and Jean-Florent Raymond. Enumerating minimal dominating sets in K_t -free graphs and variants. *arXiv preprint*, 2019. [arXiv:1810.00789](https://arxiv.org/abs/1810.00789).
- 3 Marthe Bonamy, Oscar Defrain, Marc Heinrich, and Jean-Florent Raymond. Enumerating Minimal Dominating Sets in Triangle-Free Graphs. In *36th International Symposium on Theoretical Aspects of Computer Science*, pages 16:1–16:12. Springer, 2019.
- 4 Endre Boros, Khaled Elbassioni, Vladimir Gurvich, and Leonid Khachiyan. Generating maximal independent sets for hypergraphs with bounded edge-intersections. In *Latin American Symposium on Theoretical Informatics*, pages 488–498. Springer, 2004.
- 5 Bruno Courcelle. Linear delay enumeration and monadic second-order logic. *Discrete Applied Mathematics*, 157(12):2675–2700, 2009.
- 6 Nadia Creignou, Markus Kröll, Reinhard Pichler, Sebastian Skritek, and Heribert Vollmer. A complexity theory for hard enumeration problems. *Discrete Applied Mathematics*, 2019.
- 7 Thomas Eiter, Georg Gottlob, and Kazuhisa Makino. New results on monotone dualization and generating hypergraph transversals. *SIAM Journal on Computing*, 32(2):514–537, 2003.
- 8 Michael L. Fredman and Leonid Khachiyan. On the Complexity of Dualization of Monotone Disjunctive Normal Forms. *Journal of Algorithms*, 21(3):618–628, 1996.
- 9 Petr A. Golovach, Pinar Heggernes, Mamadou M. Kanté, Dieter Kratsch, and Yngve Villanger. Enumerating minimal dominating sets in chordal bipartite graphs. *Discrete Applied Mathematics*, 199:30–36, 2016.
- 10 David S. Johnson, Mihalis Yannakakis, and Christos H. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123, 1988.
- 11 Mamadou M. Kanté, Vincent Limouzy, Arnaud Mary, and Lhouari Nourine. On the neighbourhood helly of some graph classes and applications to the enumeration of minimal dominating sets. In *International Symposium on Algorithms and Computation*, pages 289–298. Springer, 2012.
- 12 Mamadou M. Kanté, Vincent Limouzy, Arnaud Mary, and Lhouari Nourine. On the Enumeration of Minimal Dominating Sets and Related Notions. *SIAM Journal on Discrete Mathematics*, 28(4):1916–1929, 2014.
- 13 Mamadou M. Kanté, Vincent Limouzy, Arnaud Mary, Lhouari Nourine, and Takeaki Uno. A polynomial delay algorithm for enumerating minimal dominating sets in chordal graphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 138–153. Springer, 2015.
- 14 Mamadou M. Kanté, Vincent Limouzy, Arnaud Mary, Lhouari Nourine, and Takeaki Uno. Polynomial delay algorithm for listing minimal edge dominating sets in graphs. In *Workshop on Algorithms and Data Structures*, pages 446–457. Springer, 2015.
- 15 Ronald C. Read and Robert E. Tarjan. Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. *Networks*, 5(3):237–252, 1975.
- 16 Yann Strozecki and Arnaud Mary. Efficient enumeration of solutions produced by closure operations. *Discrete Mathematics & Theoretical Computer Science*, 21, 2019.
- 17 Elliot S. Wolk. The comparability graph of a tree. *Proceedings of the American Mathematical Society*, 13(5):789–795, 1962.

Dual-Mode Greedy Algorithms Can Save Energy

Barbara Geissmann 

Department of Computer Science, ETH Zürich, Switzerland
barbara.geissmann@inf.ethz.ch

Stefano Leucci 

Department of Algorithms and Complexity, Max Planck Institute for Informatics, Germany*
stefano.leucci@mpi-inf.mpg.de

Chih-Hung Liu 

Department of Computer Science, ETH Zürich, Switzerland
chih-hung.liu@inf.ethz.ch

Paolo Penna 

Department of Computer Science, ETH Zürich, Switzerland
paolo.penna@inf.ethz.ch

Guido Proietti 

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica, Università dell'Aquila, Italy
Istituto di Analisi dei Sistemi ed Informatica "A. Ruberti", CNR, Roma, Italy
guido.proietti@univaq.it

Abstract

In real world applications, important resources like energy are saved by deliberately using so-called low-cost operations that are less reliable. Some of these approaches are based on a *dual mode* technology where it is possible to choose between high-energy operations (always correct) and low-energy operations (prone to errors), and thus enable to trade energy for correctness.

In this work we initiate the study of algorithms for solving optimization problems that in their computation are allowed to choose between two types of operations: *high-energy comparisons* (always correct but expensive) and *low-energy comparisons* (cheaper but prone to errors). For the errors in low-energy comparisons, we assume the *persistent* setting, which usually makes it impossible to achieve optimal solutions without high-energy comparisons. We propose to study a natural complexity measure which accounts for the number of operations of either type separately.

We provide a new family of algorithms which, for a fairly large class of maximization problems, return a *constant approximation* using only *polylogarithmic* many high-energy comparisons and only $O(n \log n)$ low-energy comparisons. This result applies to the class of *p-extendible systems* [24], which includes several *NP-hard* problems and *matroids* as a special case ($p = 1$).

These algorithmic solutions relate to some fundamental aspects studied earlier in different contexts: (i) the approximation guarantee when only *ordinal information* is available to the algorithm; (ii) the fact that even such ordinal information may be *erroneous* because of low-energy comparisons and (iii) the ability to approximately sort a sequence of elements when comparisons are subject to *persistent errors*. Finally, our main result is quite general and can be parametrized and adapted to other error models.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases matroids, *p*-extendible systems, greedy algorithm, approximation algorithms, high-low energy

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.64

Funding Research supported by the Swiss National Science Foundation (SNFS project 200021_165524).

* Part of this work was completed while the author was affiliated with ETH Zürich.



Acknowledgements We are grateful to Peter Widmayer for many inspiring discussions.

1 Introduction

Classical computational problems have been studied under two (somewhat extreme) settings: the one in which every operation is always *correct* and the one in which operations are prone to *errors* (see, e.g., [10, 28, 11, 20]). The latter scenario represents not only faults in hardware, but also measurement errors, or even errors that are deliberately introduced in the system in order to save important resources. For instance, several approaches to save energy in computation consists in designing systems which are in part inaccurate but use substantially less energy [22, 1, 21].

Here we consider the scenario in which these two types of operations *coexist* and they may be *combined* in a clever way in order to save resources and still achieve a certain goal. Interestingly, this is already done in some practical applications. At a hardware level, the *dual mode logic* [21] allows each single gate to switch between a static mode, which uses *low energy* but suffers from some performance degradation, and a dynamic mode which uses a *higher energy* but reaches the better performance of standard CMOS gates. Similarly, in *probabilistic CMOS* [1] one can reduce the energy spent by a single gate at the price of increasing the probability of *error* in the corresponding output. In [7], the authors propose a *probabilistic adder* for image processing purposes where *high energy* is used in the most significant bits and *lower energy* in the less significant bits. In computational geometry, [12] suggests a model in which the algorithm uses either *cheap* operations (floating-point arithmetic), whose result may be erroneous in some circumstances, and *expensive* operations (exact arithmetic) whose result is always correct.

In all these examples, a good solution is obtained by combining both high-energy and low-energy (expensive and cheap) operations in a suitable way. This suggests the following *algorithmic* question regarding the *trade-offs* between high-energy and low-energy operations:

Suppose an algorithm can use both high-energy and low-energy operations, the latter being erroneous, according to some error model. How many such high-energy and low-energy operations are needed to obtain a good solution for a given problem?

1.1 Our contribution

We propose to evaluate algorithms according to a simple and natural measure that we call *high-low energy complexity*. In this model, algorithms can operate at *low energy* or at *high energy*. The former mode is cheap, but introduces some *errors* in the result of the operation, while the latter is more expensive but *always correct*. We evaluate the performance of the algorithm by essentially distinguishing between the two types of operations on inputs of size n :

- Total number $h(n)$ of *high-energy* operations;
- Total number $l(n)$ of *low-energy* operations.

In this case, we say that the algorithm has $\langle h(n), l(n) \rangle$ -*high-low energy complexity*. Because high-energy operations are more expensive, the classical notion of time complexity does not fully capture the possible *trade-offs* that may result in a lower total energy. Indeed, it may be possible to have algorithms which use *significantly fewer* high-energy operations, and essentially the same number of low-energy operations, and still obtain (nearly) optimal solutions.

In this work, we show that such trade-offs are indeed possible for a large class of problems. Specifically, we consider the setting in which the basic operations are *comparisons* between the input *weights*, which is the part of the input necessary to determine if a solution is optimal or not. For the errors in the comparisons at low energy, we assume the classical model of *persistent* errors [6, 18, 15]: comparisons between distinct pairs of elements are independent and return the wrong answer with some (small) constant probability independently across the pairs; however, comparing the same pair of elements multiple times will *always* give the same result. We consider the following setting:

- The input elements that can be part of feasible solutions have a *weight*, but the algorithm can only work with *ordinal information* meaning that it can only compare the weight of two elements, but does not know the actual weights. The computed solution is however evaluated with respect to the weights and it is compared with the optimum.
- The ordinal information is accessible to the algorithm via two types of (comparison) operations: *low-energy* operations which are *cheap* but may contain *errors*, or *high-energy* operations which are always correct, though more expensive.

We study the high-low energy complexity of a wide class of optimization problems for which greedy algorithms are guaranteed to return optimal or nearly optimal solutions: Using only high-energy operations it is possible to find such solution, but this would require already $\Theta(n \log n)$ **high-energy operations**. On the other hand, with **no high-energy operations**, errors during this sorting phase are likely to produce some *dislocation*, which in turn may cause the greedy algorithm to have an **unbounded approximation ratio**. This is true also for those problems where greedy computes the optimum like, e.g., the minimum spanning tree (as we will further discuss in Section 2). Perhaps it may be surprising if, for some problem, one could devise an algorithm with the following performance:

*Compute a **constant approximation** using only $O(\text{polylog } n)$ **high-energy operations** and $O(n \log n)$ **low-energy operations**.*

We show that this is indeed the case for the rich class of maximization problems in so called *p-extendible systems*, a generalization of matroids introduced in [24], and recently reconsidered in [8] (see Section 2 for a formal definition). We consider the case of *additive* optimization functions, where the goal is to optimize (maximize or minimize) the sum of the weights in the solution (basis). This class includes, among others, *maximum profit scheduling* ($p = 1$ or 2 depending on the version), *maximum weight b-matching* ($p = 2$), *maximum asymmetric travelling salesman problem* ($p = 3$), *weighted Δ -independent set* ($p = \Delta =$ maximum degree) [24, 8]. Interestingly, greedy (without errors) is only a constant factor away from the actual optimum: in any *p-extendible system* greedy is *p-approximate* [24], and it even returns the *optimum* if the input instance is *p-stable* [8], that is, if the *p-extendible system* admits an unique optimal solution which remains unique even when the elements' weights are perturbed by a factor between 1 and p . The special case of *p-extendible systems* with $p = 1$ are *matroids* [24], where greedy also computes the optimum (the minimum spanning tree is a classical example).

Our results show a direct connection between the number of *high-energy* operations required to obtain a provably good solution and the ability to *approximately sort* a sequence of n elements at *low energy*. For a given error model, suppose we can compute a sequence of the input elements so that the *maximum dislocation is at most* $d = d(n)$: each element appears at most d positions away from its position in the sorted sequence (depending on the weights). The results of this work are summarized in Table 1 (upper part) where for the moment we do not account for the number of operations used to approximately sort

■ **Table 1** A summary of the results for a generic input with initial dislocation at most d (**upper part**), and their instantiation for the case of *persistent* comparison errors (**lower part**). In the latter case, an additional $O(n \log n)$ low-energy operations are needed to produce the sequence with dislocation $d = \Theta(\log n)$ w.h.p. [15]. The approximation guarantee of the lower part holds w.h.p.

Problem	High-Energy	Low-Energy	Approximation
Matroids (min/max)	$O(d \log^2 d)$	$O(n)$	2 (Thm 4)
Matroids (min/max)	$O(d^2/\epsilon)$	$O(n)$	$1 + \epsilon$ (Thm 4)
p -Ext. Sys. (max, $p \geq 2$)	$O(d + \frac{d^2}{p})$	$O(n)$	p times greedy $\leq p^2$ (Thm 15)
Matroids (min/max)	$O(\log n \cdot (\log \log n)^2)$	$O(n \log n)$	2
Matroids (min/max)	$O(\frac{\log^2 n}{\epsilon})$	$O(n \log n)$	$1 + \epsilon$
p -Ext. Sys. (max, $p \geq 2$)	$O(\log n + \frac{\log^2 n}{p})$	$O(n \log n)$	p times greedy $\leq p^2$

the sequence (i.e., assume that such a sequence is provided in input). If such a sequence can be constructed in time $t_{\text{sort}}(n)$ using only unreliable (low-energy) comparisons, then the low-energy operations of the whole algorithm (first approximately sort and then run our algorithms) is $O(n + t_{\text{sort}}(n))$. For the case of *persistent* comparison errors, [15] showed that it is possible to approximately sort a sequence in $t_{\text{sort}}(n) = O(n \log n)$ time so that the maximum dislocation is $d = O(\log n)$ with high probability [15].¹ This immediately yields the bounds in Table 1 (lower part) showing that *polylogarithmic* high-energy operations suffice to compute w.h.p. an approximate solution, where the approximation guarantee depends on the problem version as shown in Table 1 (lower part).

Interestingly, since the greedy algorithm has approximation guarantee p for maximization problems restricted to p -extendible systems [24], our results for $p \geq 2$ yield a constant approximation for any $p = O(1)$, including the aforementioned NP-hard optimization problems. Moreover, for the special case of *matroids* ($p = 1$) where greedy returns the actual optimum, we show that $O(\frac{\log^2 n}{\epsilon})$ high-energy operations suffice to compute a $(1 + \epsilon)$ -approximation, and extend this result in two ways: we consider *minimization* problems as well and algorithms which use even *fewer high energy* operations and still achieve an approximation factor of 2.

Regarding problems where greedy is guaranteed to return the optimum solution, we recall that this is also the case on p -stable instances of p -extendible systems [8]. There we show that our algorithm will in some cases return a solution which is a factor p worse than the optimal (greedy) solution, thus showing that the analysis in Theorem 15 is tight (see Theorem 19).

Due to space limitations, the analysis of our result concerning maximization in matroids is omitted and will appear in the full version of the paper.

1.2 Related work

The standard greedy algorithm performs optimally or nearly optimally in several interesting classes of problems.

- Maximization of *submodular* optimization functions under *cardinality constraint*. In this case, greedy has an approximation guarantee of $\frac{e}{e-1}$ [27, 29]. An even better guarantee for greedy holds for *modular* functions and for functions that are “close” to being modular

¹ Technically this requires a comparisons error probability $p_{\text{err}} < 1/16$ for which the $O(n \log n)$ -time algorithm in [15] applies. Alternatively, if low-energy operations account only for comparisons, the same high-low energy complexity can be achieved for larger $p_{\text{err}} < 1/2$ using [6] which uses $O(n \log n)$ comparisons.

[9, 30]. Finally, constant approximations also hold for functions that are close to be submodular [5].

- Maximization in problems with *more complex constraints* has been also considered. For p -extendible systems, greedy has an approximation of $p + 1$ for submodular functions and p for additive ones (our case). For the additive case, a small constant approximation can be achieved using only *ordinal information*, i.e., without knowing the actual weights and solely based on comparisons [3, 4]. Another line of research deals with *stable* instances of p -extendible systems where greedy *recovers the optimal solution* [8].
- Minimization problems are generally harder, with the exception of matroids, including *minimum spanning tree*. Other examples are those problems where the minimum spanning tree itself is a good approximation of the optimum, and thus greedy automatically provides the same guarantee (see for example, the 2-approximation for the metric *travelling salesman problem*, and [17, 2] for connectivity problems in wireless networks). Bicriteria results for supermodular functions with cardinality constraints are given in [23], which considers extending a given solution in a greedy fashion. Finally, also for the metric travelling salesman problem, greedy recovers the *optimum* in the case of *stable* instances [26].

Theoretical models for algorithms that use *two-level* operations are not completely new, though they have been studied with different objectives. In particular, [12] distinguishes between *cheap* and *expensive* comparisons, and errors occur only in the cheap comparisons according to a *threshold error model*: a cheap comparison between two elements whose position in the sorted sequence differ by at most τ is prone to errors, and all other comparisons are correct. They suggested to use a suitable “concatenation” of two algorithms to *sort perfectly* in this error model. In our terminology, their approach has $\langle O(\tau n), O(n \log n) \rangle$ -high-low energy complexity (see also [16] for further results on this setting). Note that the techniques and the results in [12, 16] are not applicable to our setting as their error model is *different* (see next paragraph) and because our primary objective is not to sort. Indeed, our results say that sorting exactly is *not* energy-optimal for many optimization problems (while our algorithms use $O(\log^2 n)$ high-energy operations, sorting exactly w.h.p. requires $\Omega(n)$ high-energy complexity).

The error model with *persistent* errors is different from the threshold error model [12, 16], and somewhat more difficult. In the model with persistent errors, there is no bound on τ , and thus every comparison result is wrong with some probability $p_{err} > 0$, independently of the other results. In this case, the best bound on the maximum dislocation which is possible to guarantee (with high probability) is $d = \Theta(\log n)$ and, among the various known algorithms that achieve this performance [6, 18, 13, 14, 15], the fastest has running time $\Theta(n \log n)$ [15].

2 Preliminaries

Independence systems and matroids

An *independence system* is a pair $M = (E, \mathcal{I})$ where E is a collection of n elements called *ground set*; $\mathcal{I} \subseteq \mathcal{P}(E)$ is a family of *independent sets*² which is *downward closed* (also known as the *hereditary property*): if $B \in \mathcal{I}$ and $A \subseteq B$, then $A \in \mathcal{I}$; and $\emptyset \in \mathcal{I}$. A maximal (w.r.t. inclusion) independent set is called a *base* or *feasible solution*.

² Where $\mathcal{P}(E)$ denotes the power-set of E , i.e., the set of all subsets of E .

64:6 Dual-Mode Greedy Algorithms Can Save Energy

We consider *maximization* and *minimization* problems involving independence systems where each element $e \in E$ has a non-negative weight $w(e) \geq 0$. In maximization problems (resp., minimization problems) the goal is to compute a base B maximizing (resp., minimizing) the total weight $w(B) = \sum_{b \in B} w(b)$, the so called additive case. We shall restrict our attention to the following two important classes of independence systems, for which the simple greedy algorithm (see below) returns either a constant approximation or even the optimum.

Matroids: A *matroid* is an independence system (E, \mathcal{I}) that satisfies the *augmentation property*: If $A, B \in \mathcal{I}$ and $|A| < |B|$, then $\exists x \in B \setminus A$ such that $A \cup \{x\} \in \mathcal{I}$.

p -extendible systems : A *p -extendible system* is an independence system (E, \mathcal{I}) such that if $A \subseteq B \in \mathcal{I}$ and $A \cup \{e\} \in \mathcal{I}$, then $(B \setminus R) \cup \{e\} \in \mathcal{I}$ for some $R \subseteq B \setminus A$ of cardinality $|R| \leq p$.

It follows from the above definitions that the cardinalities of any two bases of a p -extendible system are at most a factor p apart. Since any matroid M is a 1-extendible system [25], it follows that all bases of M have the same cardinality, denoted by $\text{rank}(M)$. Finally, a *circuit* C of an independence system is a minimal (w.r.t. inclusion) non-independent set, i.e., $C \in \mathcal{P}(E) \setminus \mathcal{I}$.

Greedy algorithm

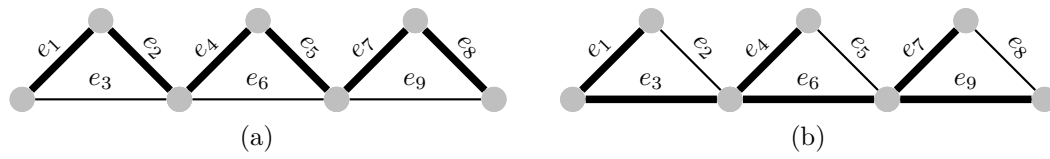
A well-known algorithm is the greedy algorithm. Starting from the empty set, it iteratively extends the current solution $A \in \mathcal{I}$ with the “best” element $x \in E \setminus A$ such that $A \cup \{x\}$ is still independent; For maximization problems, greedy considers the elements from the largest weight to the smallest one, while in minimization it follows the opposite order. The algorithm stops and returns the current set A when no such x element can be added. In the following we denote by $\text{greedy}(M)$ the set returned by (an unspecified implementation of) the greedy algorithm on M , and by $\text{greedy}(M, \tilde{S})$ the set returned by the variant of the greedy algorithm that considers the elements of M as they appear in a (non necessarily sorted) sequence \tilde{S} .

Our setting (high and low energy operations)

The greedy algorithm needs access to the ground set of M , to the function w , and to an *independence oracle* \mathcal{O} that reports whether a subset of elements of E is independent. Notice, however, that only *ordinal* information about w are needed, i.e., we can replace w with a *comparison oracle* $C_H : E \times E \rightarrow \{\leq, >\}$: a query $C_H(x, y)$ to C_H reports whether x is smaller than or equal to y .

In this paper we consider the scenario in which algorithms have access to an additional comparison oracle C_L that can sometimes return *incorrect answers*. More precisely, there is a (small) constant probability $p_{err} < 1/16$ that C_L returns the wrong answer to a query. Errors between comparisons that involve different pairs of elements are *independent*. However the answers of C_L are *persistent*, i.e., they do not change if the pair of elements is queried multiple times. Notice that the algorithm is still allowed to query C_H in order to determine with certainty the correct order relation between two elements (even if a query to C_L involving the same pair of elements has already been performed).

► **Definition 1.** We say that an algorithm has high-low energy complexity $\langle h(n), \ell(n) \rangle$ if it performs at most $h(n)$ queries to C_H , and at most $\ell(n)$ other operations (including queries to C_L).



■ **Figure 1** The graph G along with (a) its minimum spanning tree, and (b) the spanning tree returned by $\text{greedy}(M, \tilde{S})$.

► **Example 2.** The greedy algorithm has high-low energy complexity $\langle O(n \log n), O(n \log n) \rangle$ as it performs $O(n \log n)$ queries to C_H in order to sort the elements of E according to w .³

Greedy on nonsorted sequences

As already mentioned, the standard greedy algorithm first orders the elements in E according to their weights, i.e., it constructs a sorted sequence

$$S = (e_1, e_2, \dots, e_n) \quad \text{where} \quad w(e_1) \geq w(e_2) \geq \dots \geq w(e_n) \quad (1)$$

for *maximization* problems (the opposite order is considered for *minimization problems*). The greedy algorithm then considers the elements of S in order and iteratively maintains an independent set. Because sorting exactly the elements requires $\Omega(n)$ high-energy operations (see Example 2 above), we consider running greedy with respect to a different *almost-sorted* sequence $\tilde{S} = (\tilde{e}_1, \tilde{e}_2, \dots, \tilde{e}_n)$. The sequence \tilde{S} is a permutation of the elements which has some bound d on the *dislocation*: we say that \tilde{S} has dislocation at most d if there exists a sorted sequence S as in (1) such that⁴ $|t(e, S) - t(e, \tilde{S})| \leq d$, where $t(e, S)$ and $t(e, \tilde{S})$ denote the positions of e in S and in \tilde{S} , respectively.

One might wonder whether running greedy on the almost-sorted sequence \tilde{S} already results in good approximation guarantees. Unfortunately, this turns out not to be the case: the solution computed by the greedy algorithm on input a sequence with *very small dislocation* can be *arbitrarily far from the optimum*, as the following example for the minimum spanning tree problem shows.

► **Example 3 (minimum spanning tree).** Let $\epsilon \in (0, \frac{1}{8})$ and consider the graph G shown in Figure 1 (a) along with its corresponding *graphic matroid* M .⁵ The edges of G are weighted as follows: $w(e_i) = i\epsilon$ for $i = 1, \dots, 8$ and $w(e_9) = 1$, so that the sorted version of the ground set of M w.r.t. w is $S = \langle e_1, e_2, \dots, e_9 \rangle$. The minimum-weight base of M has weight 27ϵ and consists of the edges in the (unique) minimum spanning tree of G (in bold). For a sequence \tilde{S} of dislocation 1, in which we swap the order between e_2 and e_3 , e_5 and e_6 , and e_8 and e_9 , i.e., $\tilde{S} = \langle e_1, e_3, e_2, e_4, e_6, e_5, e_7, e_9, e_8 \rangle$, $\text{greedy}(M, \tilde{S})$ would select the suboptimal tree shown in Figure 1 (b). The cost of the resulting tree can be significantly higher than cost of a MST (i.e., more than $1 = w_9$ as opposed to 27ϵ) and, for tiny ϵ , there is no approximation guarantee.

Even if $\text{greedy}(M, S)$ returns the optimal solution for a minimization/maximization problems in matroids, the above example above shows that $\text{greedy}(M, \tilde{S})$ cannot approximate

³ Here, the bound on the low-energy complexity accounts for up to $O(n \log n)$ non-comparison operations.
⁴ For distinct elements (weights) the sorted sequence is unique. In general, we consider S and \tilde{S} to agree on the relative order between elements with identical weight.
⁵ The ground-set of M is the set of edges of G , while a subset of edges is independent in M if it induces a forest in G .

the greedy solution within any factor (to apply the previous example to the problem of computing a maximum-weight base of M it suffices to exchange the roles of S and \tilde{S}).

3 The general scheme for matroids

In this section we describe a general scheme which leads to different approximation algorithms for computing a minimum- or maximum-weight base in a matroid. In particular, we will then be able to prove the following theorem.

► **Theorem 4.** *Consider any maximization/minimization problem in a matroid, where the input elements are given as a sequence with dislocation at most d . There exists an algorithm which returns a 2-approximate solution and that has $\langle O(d \log^2 d), O(n) \rangle$ -high-low energy complexity. Moreover, for every $\epsilon \in (0, 1)$, there exists an algorithm which returns a $(1 + \epsilon)$ -approximate solution and that has $\langle O(d^2/\epsilon), O(n) \rangle$ -high-low energy complexity.*

Our algorithm and analysis are based on standard notions of “submatroid”. If $M = (E, \mathcal{I})$ is a matroid and $X \subseteq E$, the *restriction* $M|X$ of M to X is the matroid having X as its ground set and $\{Y \in \mathcal{I} : Y \subseteq X\}$ as its independent sets. If $X \in \mathcal{I}$, the *contraction* M/X of M by X is the matroid having $E \setminus X$ as its ground set and $\{Y \in \mathcal{P}(E \setminus X) : Y \cup X \in \mathcal{I}\}$ as its independent sets. A *minor* of M is matroid that can be obtained from M by a sequence of restrictions and contractions. We denote by $\text{Opt}(M) = w(\text{greedy}(M))$ the weight of a base of M having minimum (resp. maximum) weight in the case of minimization (resp. maximization) problems.

The algorithm

The inputs of our algorithm are a matroid M , given in the form of a set E of n elements and an independence oracle, and an approximately-sorted sequence \tilde{S} of the elements in E having dislocation at most d .⁶ If such a sequence is not readily available, one with $d = O(\log n)$ can be computed in a pre-processing step using $O(n \log n)$ low-energy operations [15].

Our algorithm, whose pseudocode is shown in Algorithm 1, performs the following steps:

1. First, we run the greedy algorithm by considering the elements as they appear in \tilde{S} . Let $A = \{a_1, a_2, \dots, a_k\} = \text{greedy}(M, \tilde{S})$ be the resulting (now possibly suboptimal) base, where a_i represents the i -th element added during the execution of the algorithm and $k = \text{rank}(M)$.
2. Next, select a suitable subset F of elements of A that will be part of our final solution. The exact details of this step will be specified later.
3. Let E' be the set of all elements $x \in E \setminus F$ such that $|t(x, \tilde{S}) - t(y, \tilde{S})| \leq 2d$ for some $y \in A \setminus F$. We define M' as the matroid having E' as its ground set, and all the sets $X \in \mathcal{P}(E')$ such that $X \cup F$ is independent in M as its independent sets. Notice that M' is a minor of M as it can be obtained by first contracting M by F , and then restricting the resulting matroid M/F to E' , i.e., $M' = (M/F)|E'$.
4. Finally, we compute a minimum-weight base A' of M' using the greedy algorithm and high-energy queries to C_H . We return $F \cup A'$.

When minimization (resp. maximization) problems are concerned, the high level intuition is that the initial greedy solution A , which can be far from optimal, contains a “large” subset

⁶ Elements are (approximately) sorted in non-decreasing or non-increasing order of weights depending on whether we are interested in a minimum-weight or maximum-weight base of M , respectively.

■ **Algorithm 1** Dual-Mode Greedy Scheme(M, \tilde{S}, d).

```

1  $A \leftarrow \text{greedy}(M, \tilde{S});$ 
2  $F \leftarrow$  Select a suitable subset of  $A$ ;
3  $E' \leftarrow \{x \in E \setminus F : \exists y \in A \setminus F, |t(x, \tilde{S}) - t(y, \tilde{S})| \leq 2d\};$ 
4  $A' \leftarrow \text{greedy}((M/F)|E');$  // high energy part
5 return  $F \cup A'$ ;

```

F of elements whose weight is comparable to the optimum. We fix these elements and isolate a “small” set E' of candidates to complete the solution. As this set is “small” we can run greedy at high energy, and hope that it will only contribute a small (resp. large enough) additional weight to the final solution, which is the union of the solutions of the two parts.

Some properties

For the sake of the analysis, we define S to be the (correctly) sorted sequence containing the elements in E . Whenever ties between two elements arise they are broken by preserving their relative order in \tilde{S} . Let $B = \{b_1, b_2, \dots, b_k\} = \text{greedy}(M, S)$ be an optimal base of M as computed by the greedy algorithm that considers the elements in the same order as S .

► **Lemma 5.** For all $i = 1, \dots, k$ we have $|t(b_i, S) - t(a_i, \tilde{S})| \leq d$.

Proof. Let S_t (resp. \tilde{S}_t) be the sequence consisting of the first t elements of S (resp. \tilde{S}). Similarly, let $A_t = \text{greedy}(M, \tilde{S}_t)$ (resp. $B_t = \text{greedy}(M, S_t)$) be the set of elements included in the independent set maintained by $\text{greedy}(M, \tilde{S})$ (resp. $\text{greedy}(M, S)$) at time t , i.e., immediately after the t -th element of \tilde{S} (resp. S) is considered.

Notice that, for every $t = 0, \dots, n$, each element in S_t must also be contained in $\tilde{S}_{\min\{t+d, n\}}$ due to the bound on the dislocation of \tilde{S} . This implies that $|B_t| \leq |A_{\min\{t+d, n\}}|$. By choosing $t = t(b_i, S)$ we obtain $i = |B_{t(b_i, S)}| \leq |A_{\min\{t(b_i, S)+d, n\}}|$ and therefore the i -th element a_i of A must have been added at a time (i.e., position in S) of at most $t(b_i, S) + d$, i.e., $t(a_i, \tilde{S}) \leq t(b_i, S) + d$, or equivalently $t(b_i, S) \geq t(a_i, \tilde{S}) - d$.

Similarly, for $t = 0, \dots, n$, S_t is a superset of $\tilde{S}_{\max\{0, t-d\}}$, implying $|B_t| \geq |A_{\max\{0, t-d\}}|$. By choosing $t = t(b_i, S)$ we obtain $i = |B_{t(b_i, S)}| \geq |A_{\max\{0, t(b_i, S)-d\}}|$, implying that the i -th element a_i in A has been added at a time of at least $t(b_i, S) - d$, i.e., $t(a_i, \tilde{S}) \geq t(b_i, S) - d$, or equivalently $t(b_i, S) \leq t(a_i, \tilde{S}) + d$. ◀

The above lemma, together with the bound on the dislocation of \tilde{S} , immediately implies:

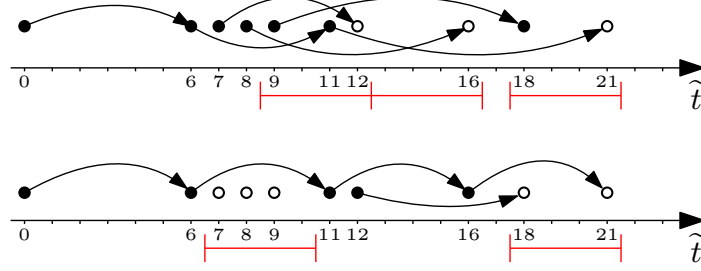
► **Corollary 6.** $|t(b_i, S) - t(a_i, S)| \leq 2d$ and $|t(b_i, \tilde{S}) - t(a_i, \tilde{S})| \leq 2d$.

Next, we show that, regardless of the choice of F , Algorithm 1 returns a base of M .

► **Lemma 7.** The set $A' \cup F$ returned by Algorithm 1 is a base of M .

Proof. We start by defining M'' as the matroid having E as its ground set and such that $X \in \mathcal{P}(E \setminus F)$ is an independent set of M'' iff $X \cup F$ is independent in M . Notice how M'' is closely related to the *contraction* of M by F (which is an independent set of M since $F \subseteq A$): the only difference is that the ground set of M'' still contains the elements in F , even though they do not belong to any independent set.

Since F and B are, respectively, an independent set and a base of M , we can iteratively invoke the augmentation property of matroids to select a set $B'' \subset B \setminus F$ of $k - |F|$ elements



■ **Figure 2** Top: an example of a $\langle \tau, \lambda \rangle$ -min-mapping with $\tau = 4$ and $\lambda = 3$. The elements in the set X are depicted as dots and the horizontal axis represents the function \tilde{t} . Filled (resp. hollow) dots correspond to mapped (resp. unmapped) elements. Three intervals of size τ whose union contains the values $\tilde{t}(x)$ of all unmapped elements x are shown in red. The mapping has been obtained by greedily assigning each element of X to the first suitable element, in increasing order of $\tilde{t}(\cdot)$. Bottom: A $\langle 4, 2 \rangle$ -min-mapping for the same set of elements X and values of $\tilde{t}(\cdot)$.

such that $F \cup B''$ is an independent set of M . This implies that B'' is also an independent set in M'' . In particular, since all the independent sets X of M'' are such that $X \cap F = \emptyset$ and $X \cup F$ is independent in M , it follows that $|X| \leq k - |F| = |B''|$ therefore that B'' is maximal w.r.t. inclusion in M'' and hence it is a base.

Notice that M' is exactly the *restriction* of M'' to the set E' , and that the set $\text{greedy}(M'', \tilde{S})$ coincides with $A \setminus F$. By Corollary 6, the position in \tilde{S} of each element in $B' = \text{greedy}(M'', S)$ differs by at most $2d$ from the position of a suitable element in $A \setminus F$, implying that $B' \subseteq E'$. To summarize, we have: (i) $B' = \text{greedy}(M'', S) = \text{greedy}(M', S) = A'$, (ii) $B' \cup F$ is independent in M , (iii) $|B' \cup F| = |B'| + |F| = \text{rank}(M') + |F| = |B''| + |F| = (k - |F|) + |F| = k$. ◀

4 Minimization in Matroids

In this section, we instantiate our general scheme in order to prove Theorem 4 in the case of *minimization in matroids* (the maximization counterpart will appear in the full version of the paper). Recall that \tilde{S} has dislocation d w.r.t. a sequence S in which elements are sorted in *non-decreasing* order of weight.

We start by proving the following lemma, which will hold for all our choices of F .

► **Lemma 8.** $w(A') \leq \sum_{i=|F|+1}^k w(b_i) \leq \text{Opt}(M)$.

Proof. Let M'' , and B'' be defined as in Lemma 7. We have that $|A' \cup F| = k$ and hence $|A'| = k - |F|$ as $A \cap F = \emptyset$. Since $A' = \text{greedy}(M', S)$ is a minimum-weight base of M' and M'' while $B'' \subseteq B$ is a base of M'' , we have $w(A') = \text{Opt}(M'') \leq w(B'')$.

By definition, B'' is a subset of B and, since $|B''| = \text{rank}(M'') = |A'| = k - |F|$, we can upper bound the total weight of the $k - |F|$ elements in B'' with that of the $k - |F|$ elements of B of largest weight, i.e., $w(B'') \leq \sum_{i=|F|+1}^k w(b_i)$. Combining all the previous inequalities, we can write: $w(A') \leq w(B'') \leq \sum_{i=|F|+1}^k w(b_i) \leq w(B) = \text{Opt}(M)$. ◀

4.1 A 2-Approximation (for Minimization in Matroids)

To instantiate the general scheme of Algorithm 1, we need to specify how the subset F of elements of Step 2 is selected. To this aim we *map* some of the elements of the initially computed solution A into some other elements of A . The set of mapped elements will be our set F , while the mapping shall satisfy the following definition.

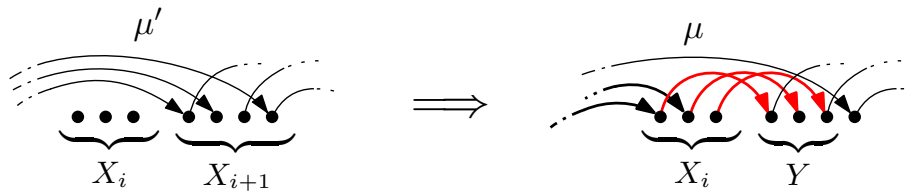


Figure 3 A graphical representation of the local transformation that allows us to assume that $|X_i| > |X_{i+1}|$. Function μ' (whose domain does not include the elements in X_i) is shown on the left, while function μ is shown on the right. Recall that $Y \subseteq X_{i+1}$ is chosen arbitrarily such that $|Y| = |X_i|$. The differences between μ' and μ are highlighted in bold. The newly-added mappings $\mu(x)$ for $x \in X_i$ are shown in red.

► **Definition 9** ($\langle \tau, \lambda \rangle$ -min-mapping). A $\langle \tau, \lambda \rangle$ -min-mapping of a set of elements X w.r.t. an injective function $\tilde{t} : X \rightarrow \mathbb{N}$ is an injective partial function $\mu : X \rightarrow X$ such that:

- For every element x in the domain $\mathcal{D}(\mu)$ of μ it holds $\tilde{t}(\mu(x)) \geq \tilde{t}(x) + \tau$;
- The integers in $\{\tilde{t}(x) : x \in X \setminus \mathcal{D}(\mu)\}$ are all contained in the union of at most λ intervals of contiguous integers, each of size at most τ .

In other words, if we think of $\tilde{t}()$ as a function that associates a *time* to each element of X , the above definition guarantees that an element x of X is either mapped to some other element $y \in X$ that appears sufficiently later in time, or it belongs to a small set of at most λ *time intervals*, each of size τ . Figure 2 shows an example of a $\langle 4, 3 \rangle$ - and a $\langle 4, 2 \rangle$ -min-mapping.

Finding a $\langle \tau, 4 \log \tau + 4 \rangle$ -min-mapping

We now show how a $\langle \tau, \lambda \rangle$ -min-mapping with $\lambda = O(\log \tau)$ can be found, which will turn out to be the best asymptotic trade-off between τ and λ one can hope to obtain. In particular, we will set $\lambda = 4 \log \tau + 4$.

To this aim, it is useful to consider a relaxed variant of the problem. Intuitively, we get rid of $\tilde{t}()$ by grouping the elements of X into a collection of sets $\langle X_1, X_2, \dots, X_m \rangle$. Instead of requiring $\tilde{t}(\mu(x)) \geq \tilde{t}(x) + \tau$, it will be enough for $x \in X_i$ to be mapped to some element $\mu(x) \in X_j$ with $j > i$. Moreover, we allow up to $2 \log \tau + 2$ sets X_i to contain unmapped elements.

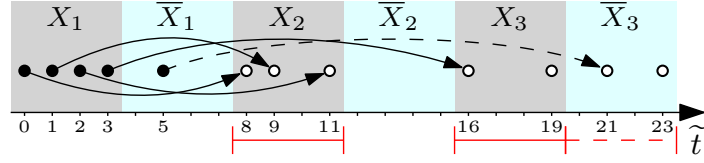
Formally, we are given a sequence of m pairwise-disjoint sets $\langle X_1, X_2, \dots, X_m \rangle$ each containing at most τ elements, as we want to find:

- A subset N of $\{X_1, X_2, \dots, X_m\}$ of size at most $2 \log \tau + 2$.
- An injective function $\mu : \bigcup_{X_i \notin N} X_i \rightarrow \bigcup_i X_i$ such that $x \in X_i \implies \mu(x) \in X_j$ for a $j > i$.

W.l.o.g. we can restrict ourselves to the case in which the cardinalities of the sets X_i are monotonically decreasing and $|X_m| > 0$. Indeed, if i is an index such that $|X_i| \leq |X_{i+1}|$, we can consider the modified instance consisting of the sets $\{X_1, \dots, X_{i-1}, X_{i+1}, \dots, X_m\}$ instead. Any solution (N', μ') to this modified instance yields a solution (N, μ) for the original instance. Indeed, if $X_i = \{x_1, x_2, \dots\}$, it suffices to pick $N = N'$, select an arbitrary subset $Y = \{y_1, y_2, \dots\}$ of $|X_i|$ elements from X_{i+1} , and define μ as follows (see also Figure 3):

$$\mu(x) = \begin{cases} y_j & \text{if } x \in X_i, \text{ where } j \text{ is such that } x = x_j, \\ x_j & \text{if } x \notin X_i \text{ and } \mu'(x) \in Y, \text{ where } j \text{ is such that } \mu'(x) = y_j, \\ \mu'(x) & \text{otherwise.} \end{cases}$$

64:12 Dual-Mode Greedy Algorithms Can Save Energy



■ **Figure 4** Decomposition of a set X into the families of sets X_1, X_2, \dots and $\bar{X}_1, \bar{X}_2, \dots$ used in the relaxed version of our mapping problem for $\tau = 4$. The elements in the set X are depicted as dots and the horizontal axis represents the function \tilde{t} . Filled (resp. hollow) dots correspond to elements that belong to (resp. do not belong to) $\mathcal{D}(\mu)$. The solution (N_1, μ_1) is shown with solid lines, while the solution (N_2, μ_2) is shown with dashed lines. Their combination yields the $\langle \tau, \lambda \rangle$ -min-mapping μ with $\lambda = 4 \log \tau + 4$. The (at most λ) red intervals span all the elements in $N_1 \cup N_2 = X \setminus \mathcal{D}(\mu)$.

We henceforth assume that $|X_i| > |X_{i+1}|$ for all $i = 1, \dots, m-1$. Our algorithm starts by letting $i = 1$ and $N = \emptyset$, then it iteratively looks for the largest index $j \geq i$ such that $|X_j| \geq |X_i|/2$ and performs one of the following two steps, depending on the value of j :

- If $j = i$, then X_i is added to N .
- If $j > i$, we assign consecutive indices $1, 2, \dots$, to the elements in X_i, \dots, X_j (in order). Let x_h be the element with index h . We define $\mu(x_h) = x_{h+|X_i|}$ for all $h = 1, \dots, \sum_{\ell=i}^{j-2} |X_\ell|$. Notice that, for every $x_h \in \bigcup_{\ell=i}^{j-2} X_\ell$, we have that $\mu(x_h) \in \bigcup_{\ell=i+1}^j X_\ell$, that $\mu(x_h)$ is necessarily in a successive set, and that the mapping is injective. Finally, we add X_j and X_{j-1} to N .

If $j < m$ we set $i = j + 1$ and continue with the next iteration, otherwise we return the pair (N, μ) .

Observe that, in every iteration, the cardinality of N increases by at most 2. Moreover, if $j < m$, the set X_i has cardinality least twice the one of the set X_{j+1} , i.e., the set considered at the next iteration. This means that there will be at most $\log |X_1| + 1$ iterations, thus $|N| \leq 2 \log |X_1| + 2 \leq 2 \log \tau + 2$.

We now argue that a $\langle \tau, 4 \log \tau + 4 \rangle$ -min-mapping of a set X w.r.t. $\tilde{t} : X \rightarrow \mathbb{N}$ can be found by solving *two instances* of the relaxed problem. Namely, for $i = 1, 2, \dots$, we define $X_i = \{x \in X : 2(i-1)\tau \leq \tilde{t}(x) < (2i-1)\tau\}$ and $\bar{X}_i = \{x \in X : (2i-1)\tau \leq \tilde{t}(x) < 2i\tau\}$. Let (N_1, μ_1) and (N_2, μ_2) be two solutions to the instances of the above problem consisting of the sets X_i and \bar{X}_i , respectively. Then, the mapping μ defined as $\mu(x) = \mu_1(x)$ if $x \in \mathcal{D}(\mu_1)$ and $\mu(x) = \mu_2(x)$ if $x \in \mathcal{D}(\mu_2)$ is a $\langle \tau, 4 \log \tau + 4 \rangle$ -min-mapping. Notice indeed that each element in $X \setminus \mathcal{D}(\mu)$ is contained in one of the at most $4 \log \tau + 4$ sets in $N_1 \cup N_2$, and – by our definition of X_i and \bar{X}_i – all the elements in a set $Y \in N_1 \cup N_2$ are such that all $\tilde{t}(x)$ for $x \in Y$ belong to a single interval of size τ . Moreover, since each element in $x \in X_i \notin N_1$ is mapped to an element $\mu(x)$ in some X_j with $j \geq i + 1$, we know that $\tilde{t}(x) < (2i-1)\tau$ while $\tilde{t}(\mu(x)) \geq 2i\tau$, i.e., $\tilde{t}(\mu(x)) - \tilde{t}(x) > \tau$, as desired (a symmetrical arguments holds for $x \in \bar{X}_i \notin N_2$). See Figure 4 for an example.

It is not hard to see that such a mapping can be computed in $O(|X|)$ time if a sorted version of X w.r.t. $\tilde{t}(\cdot)$ is known, as it will be the case in the sequel.

There is no $\langle \tau, o(\log \tau) \rangle$ -min-mapping

We point out that the above construction of a $\langle \tau, \lambda \rangle$ -min-mapping essentially achieves the best attainable trade-off between τ and λ ,

► **Lemma 10.** *In general, there exists no $\langle \tau, o(\log \tau) \rangle$ -min-mapping.*

Proof. Let $h = \lfloor \log \tau \rfloor$ and consider a set X of $2^{h+1} - 1$ elements which is partitioned into $h + 1$ sets X_0, \dots, X_h where $X_i = \{x_1^{(i)}, x_2^{(i)}, \dots\}$ contains 2^{h-i} elements. We define $\tilde{t}(x_j^{(i)}) = 2^i + j + i\tau - 1$.

Let μ be any $\langle \tau, h \rangle$ -min-mapping of X w.r.t. \tilde{t} . We say that a set X_i is covered by μ if $X_i \subseteq \mathcal{D}(\mu)$. We claim that, in μ , no set can be covered. Indeed, assume towards a contradiction that at least one set X_i is covered. Since the value of \tilde{t} for any two elements in X_i differs by at most $|X_i| - 1 \leq 2^{h-i} - 1 \leq 2^h - 1 \leq \tau - 1$, we have that, for every $x_j^{(i)} \in X_i$, $\mu(x_j^{(i)}) \in X_\ell$ for some $\ell > i$. However, $|\bigcup_{\ell=i+1}^h X_\ell| = \sum_{\ell=0}^{h-i-1} 2^\ell = 2^{h-i} - 1$ and, since $|X_i| = 2^{h-i}$ this contradicts the fact that μ is an injective function.

To conclude the proof it suffices to notice that $h + 1 = \lfloor \log \tau \rfloor + 1 > \log \tau$ intervals of length at most τ are necessary for their union to include all the integers in $X \setminus \mathcal{D}(\mu)$. ◀

Analysis for the 2-Approximation

In order to obtain a 2-approximate minimum-weight base of M , we compute a $\langle 2d, 4 \log d + 8 \rangle$ -min-mapping μ of A w.r.t. $\tilde{t}(x) = t(x, \tilde{S})$ and we choose F as the domain $\mathcal{D}(\mu)$ of μ . Intuitively, for $\tau = 2d$, the first condition of Definition 9 gives an *implicit* partial injective mapping into elements of B of non smaller weight, thus yielding $w(F) \leq \text{Opt}(M)$. The second condition can be used to show that the set F can be extended in an optimal way by looking at a “small” subset of elements (Steps 3 and 4 of Algorithm 1), and thus the number of high-energy operations is not too large. As this part of the solution also contributes at most another factor $\text{Opt}(M)$, we get a 2-approximation. We formalize this intuition in the following lemmas:

► **Lemma 11.** *The set returned by Algorithm 1 with $F = \mathcal{D}(\mu)$ has weight at most $2 \text{Opt}(M)$.*

Proof. We associate each $a_i \in A \cap \mathcal{D}(\mu) = \mathcal{D}(\mu)$ to an element $b_j \in B$, where j is the index such that $a_j = \mu(a_i)$. By definition of $\langle 2d, 4 \log d + 8 \rangle$ -min-mapping we know that $\tilde{t}(a_j) \geq \tilde{t}(a_i) + 2d$ and, by Lemma 5, $\tilde{t}(a_j) = t(a_j, \tilde{S}) \leq t(b_j, S) + d$. Combining the previous inequalities we have $t(b_j, S) \geq \tilde{t}(a_j) - d \geq \tilde{t}(a_i) + d = t(a_i, \tilde{S}) + d \geq t(a_i, S)$, thus implying that $w(b_j) \geq w(a_i)$, as b_j appears no earlier than a_j in S (which is sorted in nondecreasing order of weights). Moreover, since μ is injective, our mapping between $\mathcal{D}(\mu)$ and B is also injective. Therefore: $w(\mathcal{D}(\mu)) \leq w(B) = \text{Opt}(M)$. We can now use the above inequality together with Lemma 8 to conclude that $w(F \cup A') = w(\mathcal{D}(\mu)) + w(A') \leq 2 \text{Opt}(M)$. ◀

► **Lemma 12.** *The high-low energy complexity of Algorithm 1 when $F = \mathcal{D}(\mu)$ and the elements of E are given in a almost-sorted sequence \tilde{S} having dislocation at most d is $\langle O(d \cdot \log^2 d), O(n) \rangle$.*

Proof. Once the base A of M is found using $O(n)$ low-energy operations, both the function μ and the set $F = \mathcal{D}(\mu)$ can be computed in linear time w.r.t. $|A|$ without requiring access to the oracle C_H , as we discussed in Section 4.1. This, in turn, allows to construct E' using $O(n)$ additional low-energy operations.

Finally, finding the set $A' = \text{greedy}(M')$ requires $O(d \cdot \log^2 d)$ high- and low-energy operations. Indeed, since μ is a $\langle 2d, 4 \log d + 8 \rangle$ -min-mapping, the set E' will contain at most $6d+1$ elements for each of the $4 \log d + 8$ unmapped intervals, i.e., $|E'| \leq (4 \log d + 8) \cdot (2d + 1) =$

$O(d \log d)$ implying that E' can be sorted using $O(d \log d \cdot \log(d \log d)) = O(d \cdot \log^2 d)$ high-energy queries to C_H . ◀

4.2 A $(1 + \epsilon)$ -Approximation (for Minimization in Matroids)

In order to improve the approximation guarantee from 2 to $1 + \epsilon$, for any $\epsilon \in (0, 1)$, we shall define the set F in Step 2 of Algorithm 1 as the first $k - cd$ elements of the initially computed solution A , for $c = \lceil 2/\epsilon \rceil$. The approximation guarantee and the high-low energy complexity are given by the next two lemmas, respectively. Recall that a_i (resp. b_i) is the i -th element added to the independent set maintained by $\text{greedy}(M, \tilde{S})$ (resp. $\text{greedy}(M, S)$).

► **Lemma 13.** *The set returned by Algorithm 1 with $F = \{a_1, \dots, a_{k-cd}\}$ has weight at most $(1 + \epsilon) \text{Opt}(M)$.*

Proof. By Corollary 6, $t(a_i, S) \leq t(b_i, S) + 2d \leq t(b_{i+2d}, S)$, which implies $w(a_i) \leq w(b_{i+2d})$ (notice that $c \geq 2$, therefore $i + 2d \leq k - (c - 2)d \leq k$ and b_{i+2d} always exists). We thus get the following bound: $w(F) = \sum_{i=1}^{k-cd} w(a_i) \leq \sum_{i=1+2d}^{k-(c-2)d} w(b_i)$.

From Lemma 8 we also have $w(A') \leq \sum_{i=k-cd+1}^k w(b_i)$ and, combining the above inequalities, we obtain:

$$\begin{aligned} w(F \cup A') &\leq \sum_{i=1+2d}^{k-(c-2)d} w(b_i) + \sum_{i=k-cd+1}^k w(b_i) = \sum_{i=1+2d}^k w(b_i) + \sum_{i=k-cd+1}^{k-(c-2)d} w(b_i) \\ &\leq \text{Opt}(M) + \frac{2}{c} \sum_{i=k-cd+1}^k w(b_i) \leq (1 + \epsilon) \text{Opt}(M). \end{aligned} \quad \blacktriangleleft$$

► **Lemma 14.** *For any $\epsilon > 0$, the high-low energy complexity of Algorithm 1 when $F = \{a_1, \dots, a_{k-cd}\}$ and the elements of E are given in a almost-sorted sequence \tilde{S} having dislocation at most d is $\langle O(\epsilon^{-1}d^2), O(n) \rangle$.*

Proof. Similarly to the proof of Lemma 12, A can be computed using at most $O(n)$ low-energy operations. Notice that F can trivially be found in linear time in $|A|$ and that the set E' contains at most $|A \setminus F| \cdot d = O(\frac{1}{\epsilon}d^2)$ elements.

We now simulate the greedy algorithm in order to compute an optimal base A' of M' as follows: We start with $A' = \emptyset$ and we consider the elements x in $A \setminus F$ in increasing order of $t(x, \tilde{S})$ until we find an element x' such that $A' \cup \{x'\}$ is independent. We then perform a linear search for the minimum-weight element x^* among the ones in $\{y \in E : A' \cup \{y\} \text{ is independent in } M' \text{ and } t(x', \tilde{S}) \leq t(y, \tilde{S}) \leq t(x', \tilde{S}) + d\}$ using $O(d)$ high-energy queries to C_H , we add x^* to A' and we resume considering the elements x in $A \setminus D(\mu)$ such that $t(x, \tilde{S}) \geq t(x', \tilde{S})$.⁷

Since $\text{rank}(M') = k - |F| = cd$, the total number of high-energy operations is $O(cd^2) = O(\frac{d^2}{\epsilon})$. ◀

To conclude this section, we remark that Lemmas 7, 11, 12, 13, and 14 together prove Theorem 4 when minimization problems are concerned.

⁷ Notice how the next element to be considered will again be x' .

■ **Algorithm 2** Dual-Mode p -Extendible-System Maximization(M, p, \tilde{S}, d).

-
- 1 $\gamma \leftarrow 1 + \lceil \frac{2d}{p-1} \rceil$;
 - 2 $B^* \leftarrow$ First γ elements included in the independent set maintained by $\text{greedy}(M)$;
 - 3 $A \leftarrow \text{greedy}(M/B^*, \tilde{S})$;
 - 4 **return** $A \cup B^*$;
-

5 Maximization in p -Extendible Systems

In this section we show the following theorem which yields a p^2 -approximation for the general problem of computing a maximum-weight base of a p -extendible system M , and a p -approximation if M is p -stable.

► **Theorem 15.** *Consider any maximization problem in p -extendible systems, with $p \geq 2$, where the input elements are given as a sequence with maximum dislocation at most d . There exists an algorithm which returns a p -approximation of the base returned by the greedy algorithm and that has $\langle O(d + \frac{d^2}{p}), O(n) \rangle$ -high-low energy complexity.*

Similarly to matroids, we define the contraction M/X of $M = (E, \mathcal{I})$ by $X \in \mathcal{I}$ as the independence system having $E \setminus X$ as its ground set and all $Y \in \mathcal{P}(E \setminus X)$ such that $Y \cup X \in \mathcal{I}$ as its independent sets. It is easy to check that M/X is a p -extendible system.

Our Algorithm, whose pseudocode is shown in Algorithm 2, computes an independent set B^* consisting of the first $\gamma = 1 + \lceil \frac{2d}{p-1} \rceil$ elements that $\text{greedy}(M)$ would select, and then completes the solution with the base A of $M' = M/B^*$ obtained by greedily adding the elements in \tilde{S} . We start our analysis by proving a generalization of Lemma 5 to p -extendible systems.

► **Lemma 16.** *Let M be a p -extendible system and $k = |\text{greedy}(M, S)|$. Let S_t (resp. \tilde{S}_t) be the sequence containing the first t elements of S (resp. \tilde{S}), $A_t = \{a_1, a_2, \dots\} = \text{greedy}(M, \tilde{S}_t)$, and $B_t = \{b_1, b_2, \dots\} = \text{greedy}(M, S_t)$. For all $i = 1, \dots, \lfloor k/p \rfloor$ it holds $t(a_i, \tilde{S}) \leq t(b_{i \cdot p}, S) + d$.*

Proof. For any time $t = 0, \dots, n$ we have $S_t \subseteq \tilde{S}_{\min\{n, t+d\}}$. This implies that $|A_{\min\{t+d, n\}}| \geq |B_t|/p$. By choosing $t = t(b_{i \cdot p}, S)$ we obtain $|A_{\min\{t(b_{i \cdot p}, S)+d, n\}}| \geq |B_{t(b_{i \cdot p}, S)}|/p = ip/p = i$, meaning that $\text{greedy}(M, \tilde{S})$ must have already considered a_i by the time it finished considering the $(t(b_{i \cdot p}, S) + d)$ -th element of \tilde{S} , i.e., $t(a_i, \tilde{S}) \leq t(b_{i \cdot p}, S) + d$. ◀

We can now lower bound the weight of the base returned by Algorithm 2. Since if $p = 1$ the results of the previous sections apply, we henceforth assume $p \geq 2$.

► **Lemma 17.** *Algorithm 2 returns a base $A \cup B^*$ of M of weight at least $\frac{1}{p}w(\text{greedy}(M))$.*

Proof. Let $B = \text{greedy}(M, S)$, $M' = M/B^*$, and $B' = \{b'_1, \dots, b'_k\} = \text{greedy}(M', S)$. Since a contraction of a p -extendible system is again a p -extendible system and $|A| \geq k/p$, we can invoke Lemma 16 on M' to write:

$$w(A) = \sum_{a_i \in A} w(a_i) \geq \sum_{i=1}^{\lfloor \frac{k-2d}{p} \rfloor} w(a_i) \geq \sum_{i=1}^{\lfloor \frac{k-2d}{p} \rfloor} w(b'_{ip+2d}) \geq \frac{1}{p} \sum_{i=p+2d}^k w(b'_i).$$

64:16 Dual-Mode Greedy Algorithms Can Save Energy

Notice that $B = B^* \cup B'$ and that all the elements in B^* weigh at least $w(b'_1)$, therefore:

$$\sum_{i=1}^{p+2d-1} w(b'_i) \leq (p+2d-1)w(b'_1) \leq (p+2d-1) \cdot \frac{1}{\gamma} w(B^*) \leq \frac{p+2d-1}{1 + \frac{2d}{p-1}} w(B^*) = (p-1)w(B^*).$$

Combining the above inequalities:

$$\begin{aligned} w(A \cup B^*) &= w(A) + w(B^*) \geq \frac{1}{p} \sum_{i=p+2d}^k w(b'_i) + w(B^*) \\ &= \frac{1}{p} \sum_{i=1}^k w(b'_i) - \frac{1}{p} \sum_{i=1}^{p+2d-1} w(b'_i) + w(B^*) \geq \frac{1}{p} w(B') - \frac{p-1}{p} w(B^*) + w(B^*) \\ &= \frac{1}{p} (w(B') + w(B^*)) = \frac{1}{p} w(B). \end{aligned}$$

To conclude the proof we need to show that $A \cup B^*$ is a base of M . Since $A \cup B^*$ is an independent set of M by construction, we only need to show that it is maximal. In order to do so suppose towards a contradiction that there exists an element $x \in E \setminus (A \cup B^*)$ such that $A \cup B^* \cup \{x\}$ is independent. If we let A' be the independent set maintained by $\text{greedy}(M/B^*, \tilde{S})$ immediately before x is considered, then we have that $A' \cup \{x\} \subseteq A \cup \{x\} \subseteq A \cup B^* \cup \{x\}$ must also be independent, contradicting $x \neq A$. \blacktriangleleft

We now bound the high-low energy complexity of Algorithm 2 which, when combined with Lemma 17, immediately yields Theorem 15.

► **Lemma 18.** *The high-low energy complexity of Algorithm 2 when the elements of E are given in a almost-sorted sequence \tilde{S} having dislocation at most d is $\langle O(d + \frac{d^2}{p}), O(n) \rangle$.*

Proof. Since the overall number of the low-energy operations is $O(n)$ we only need to bound the number of high-energy operations, i.e., the ones needed to select B^* . By using a technique similar to the one described in the proof of Lemma 14, we can select the first $\gamma = O(1 + \frac{d}{p})$ elements of $\text{greedy}(M, S)$ by performing $O(d)$ high-energy queries to C_H per element. The overall high-energy complexity is therefore $O(d + \frac{d^2}{p})$. \blacktriangleleft

Since in any p -extendible system, the greedy algorithm recovers the optimum whenever the instance is p -stable, Lemma 17 implies that Algorithm 2 computes a p -approximation. The next result, whose proof will appear in the full version of this paper, shows that our analysis is actually tight whenever $d \geq p$ (we recall that the best dislocation that sorting algorithms can achieve with high probability is $\Omega(\log n)$).

► **Theorem 19.** *For every $d \geq p \geq 2$, there exists a p -stable instance of a p -extendible system for which Algorithm 2 is no better than p -approximate.*

One might wonder whether our techniques can be extended to larger classes of independence system, e.g., to p -systems [19] (i.e., independence system such that the ratio between the cardinality of any two maximal independent sets is at most p). Unfortunately, the answer is negative: our analysis uses the fact that p -extendible systems are closed under restrictions and contractions. This is no longer true when p -systems are considered, as the following counterexample shows: Fix any $n \geq 2$ and let $A = \{a_1, \dots, a_n\}$ and $B = \{b_1, \dots, b_n\}$ be two disjoint sets. We define the independence system $M = (\mathcal{E}, \mathcal{I})$ where $\mathcal{E} = A \cup B$ and $\mathcal{I} = \mathcal{P}(A) \cup \mathcal{P}(B)$. Notice that M is a 1-system as the only two maximal independent sets of M are A and B . Consider now $M' = M|(A \cup \{b_1\})$. The only two maximal independent sets of M' are A and $\{b_1\}$, showing that M' is a n -system but not a $(n-1)$ -system.

References

- 1 Bilge ES Akgul, Lakshmi N Chakrapani, Pinar Korkmaz, and Krishna V Palem. Probabilistic CMOS technology: A survey and future directions. In *Very Large Scale Integration, 2006 IFIP International Conference on*, pages 1–6. IEEE, 2006.
- 2 Christoph Ambühl. An optimal bound for the MST algorithm to compute energy efficient broadcast trees in wireless networks. In *Proc. of International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 1139–1150. Springer, 2005.
- 3 Elliot Anshelevich and Shreyas Sekar. Blind, Greedy, and Random: Algorithms for Matching and Clustering Using Only Ordinal Information. In *Proc. of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 390–396, 2016.
- 4 Elliot Anshelevich and Shreyas Sekar. Truthful mechanisms for matching and clustering in an ordinal world. In *International Conference on Web and Internet Economics*, pages 265–278. Springer, 2016.
- 5 Andrew An Bian, Joachim M. Buhmann, Andreas Krause, and Sebastian Tschiatschek. Guarantees for Greedy Maximization of Non-submodular Functions with Applications. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 498–507, International Convention Centre, Sydney, Australia, 06–11 August 2017. PMLR.
- 6 Mark Braverman and Elchanan Mossel. Noisy Sorting Without Resampling. In *Proc. of the 19th Annual Symposium on Discrete Algorithms (SODA)*, pages 268–276, 2008. [arXiv:0707.1051](https://arxiv.org/abs/0707.1051).
- 7 Lakshmi N. B. Chakrapani, Jason George, Bo Marr, Bilge E. S. Akgul, and Krishna V. Palem. Probabilistic Design: A Survey of Probabilistic CMOS Technology and Future Directions for Terascale IC Design. In Giovanni De Micheli, Salvador Mir, and Ricardo Reis, editors, *VLSI-SoC: Research Trends in VLSI and Systems on Chip*, pages 101–118, Boston, MA, 2008. Springer US.
- 8 Vaggos Chatziafratis, Tim Roughgarden, and Jan Vondrák. Stability and Recovery for Independence Systems. In *Proc. of the 25th Annual European Symposium on Algorithms (ESA)*, volume 87 of *LIPICs*, pages 26:1–26:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. [doi:10.4230/LIPICs.ESA.2017.26](https://doi.org/10.4230/LIPICs.ESA.2017.26).
- 9 Michele Conforti and Gérard Cornuéjols. Submodular set functions, matroids and the greedy algorithm: tight worst-case bounds and some generalizations of the Rado-Edmonds theorem. *Discrete applied mathematics*, 7(3):251–274, 1984.
- 10 Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. Computing with Noisy Information. *SIAM Journal on Computing*, 23(5):1001–1018, 1994. [doi:10.1137/S0097539791195877](https://doi.org/10.1137/S0097539791195877).
- 11 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](https://doi.org/10.1016/j.tcs.2009.07.026).
- 12 Stefan Funke, Kurt Mehlhorn, and Stefan Näher. Structural filtering: a paradigm for efficient and exact geometric programs. *Comput. Geom.*, 31(3):179–194, 2005.
- 13 Barbara Geissmann, Stefano Leucci, Chih-Hung Liu, and Paolo Penna. Sorting with Recurrent Comparison Errors. In *ISAAC*, volume 92 of *LIPICs*, pages 38:1–38:12. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.
- 14 Barbara Geissmann, Stefano Leucci, Chih-Hung Liu, and Paolo Penna. Optimal Dislocation with Persistent Errors in Subquadratic Time. In *Proc. of the 35th Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 96 of *LIPICs*, pages 36:1–36:13, 2018.
- 15 Barbara Geissmann, Stefano Leucci, Chih-Hung Liu, and Paolo Penna. Optimal Sorting with Persistent Comparison Errors. In *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany.*, pages 49:1–49:14, 2019. [doi:10.4230/LIPICs.ESA.2019.49](https://doi.org/10.4230/LIPICs.ESA.2019.49).
- 16 Barbara Geissmann and Paolo Penna. Inversions from Sorting with Distance-Based Errors. In *Proc. of the 44th International Conference on Current Trends in Theory and Practice of*

- Computer Science (SOFSEM)*, volume 10706 of *Lecture Notes in Computer Science*, pages 508–522. Springer, 2018. doi:10.1007/978-3-319-73117-9_36.
- 17 Lefteris M Kirousis, Evangelos Kranakis, Danny Krizanc, and Andrzej Pelc. Power consumption in packet radio networks. *Theoretical Computer Science*, 243(1-2):289–305, 2000.
 - 18 Rolf Klein, Rainer Penninger, Christian Sohler, and David P. Woodruff. Tolerant Algorithms. In *Proc. of the 19th Annual European Symposium on Algorithm (ESA)*, pages 736—747, 2011.
 - 19 Bernhard Korte and Dirk Hausmann. An Analysis of the Greedy Heuristic for Independence Systems. In B. Alspach, P. Hell, and D.J. Miller, editors, *Algorithmic Aspects of Combinatorics*, volume 2 of *Annals of Discrete Mathematics*, pages 65–74. Elsevier, 1978. doi:10.1016/S0167-5060(08)70322-4.
 - 20 Stefano Leucci, Chih-Hung Liu, and Simon Meierhans. Resilient Dictionaries for Randomly Unreliable Memory. In *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany.*, pages 70:1–70:16, 2019. doi:10.4230/LIPIcs.ESA.2019.70.
 - 21 Itamar Levi and Alexander Fish. Dual mode logic—Design for energy efficiency and high performance. *IEEE access*, 1:258–265, 2013.
 - 22 Sven Leyffer, Stefan M. Wild, Mike Fagan, Marc Snir, Krishna V. Palem, Kazutomo Yoshii, and Hal Finkel. Doing Moore with Less - Leapfrogging Moore’s Law with Inexactness for Supercomputing. *CoRR*, abs/1610.02606, 2016. arXiv:1610.02606.
 - 23 Edo Liberty and Maxim Sviridenko. Greedy Minimization of Weakly Supermodular Set Functions. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM-17)*, volume 81 of *LIPIcs*, pages 19:1–19:11, 2017. doi:10.4230/LIPIcs.APPROX-RANDOM.2017.19.
 - 24 Julián Mestre. Greedy in approximation algorithms. In *European Symposium on Algorithms*, pages 528–539. Springer, 2006.
 - 25 Julián Mestre. Greedy in Approximation Algorithms. In *Algorithms - ESA 2006, 14th Annual European Symposium, Zurich, Switzerland, September 11-13, 2006, Proceedings*, pages 528–539, 2006. doi:10.1007/11841036_48.
 - 26 Matúš Mihalák, Marcel Schöngens, Rastislav Šrámek, and Peter Widmayer. On the complexity of the metric tsp under stability considerations. In *Proc. of International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, pages 382–393. Springer, 2011.
 - 27 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.
 - 28 Andrzej Pelc. Searching games with errors - fifty years of coping with liars. *Theor. Comput. Sci.*, 270(1-2):71–109, 2002.
 - 29 Jan Vondrák. Optimal approximation for the submodular welfare problem in the value oracle model. In *Proc. of the fortieth Annual ACM Symposium on Theory of computing (STOC)*, pages 67–74. ACM, 2008.
 - 30 Jan Vondrák. Submodularity and curvature: The optimal algorithm (combinatorial optimization and discrete algorithms). Mathematical Analysis Laboratory of Kyoto University, 2010.