24th Annual European Symposium on Algorithms

ESA 2016, August 22–24, 2016, Aarhus, Denmark

Edited by

Piotr Sankowski

Christos Zaroliagis

LIPICS

Editors

Piotr Sankowski	Christos Zaroliagis
Institute of Infortmatics	Department of Computer Engineering & Informatics
University of Warsawa, Poland	University of Patras, Greece
`sank@mimuw.edu.pl`	`zaro@ceid.upatras.gr`

ACM Classification 1998
E.1 Data Structures, F.2.2 Nonnumerical Algorithms and Problems, G.1.6 Optimization, G.2 Discrete Mathematics, G.4 Mathematical Software, I.1.2 Algorithms, I.2.8 Problem Solving, Control Methods, and Search, I.3.5 Computational Geometry and Object Modeling.

ISBN 978-3-95977-015-6

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-95977-015-6.

Publication date
August, 2016

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 http://dnb.d-nb.de.

Digital Object Identifier: 10.4230/LIPIcs.ESA.2016.0

ISSN 1868-8969

http://www.dagstuhl.de/lipics

To all algorithmicists

Contents

Preface

This volume contains the extended abstracts selected for presentation at ESA 2016, the 24th European Symposium on Algorithms, held in Aarhus, Denmark, on 22–24 August 2016, as part of ALGO 2016. The ESA symposia are devoted to fostering and disseminating the results of high-quality research on algorithms and data structures. ESA seeks original algorithmic contributions for problems with relevant theoretical and/or practical applications and aims at bringing together researchers in the computer science and operations research communities. Ever since 2002, it has had two tracks, the Design and Analysis Track (Track A), intended for papers on the design and mathematical analysis of algorithms, and the Engineering and Applications Track (Track B), for submissions dealing with real-world applications, engineering, and experimental analysis of algorithms. Information on past symposia, including locations and proceedings, is maintained at http://esa-symposium.org.

In response to the call for papers, ESA 2016 attracted a rather high number of 282 submissions, 230 for Track A and 52 for Track B. Paper selection was based on originality, technical quality, and relevance. Considerable effort was devoted to the evaluation of the submissions, with at least three reviews per paper. With the help of more than 1040 expert reviews and more than 470 external reviewers, the two committees selected 76 papers for inclusion in the scientific program of ESA 2016, 63 in Track A and 13 in Track B, yielding an acceptance rate of about 27%. In addition to the accepted contributions, the symposium featured two invited lectures by Guiseppe Italiano (University of Roma "Tor Vergata", Italy) and Ola Svensson (EPFL, Switzerland). Contributions of the invited lectures are also included in this volume.

The European Association for Theoretical Computer Science (EATCS) sponsored a best paper award and a best student paper award. A submission was eligible for the best student paper award if all authors were doctoral, master, or bachelor students at the time of submission.

The best student paper award was shared by two papers: one by Michele Borassi and Emanuele Natale for their contribution "KADABRA is an ADaptive Algorithm for Betweenness via Random Approximation", and the other by Adam Kunysz "The Strongly Stable Roommates Problem".

The best paper award went to two papers: one by Stefan Kratsch for his contribution "A randomized polynomial kernelization for Vertex Cover with a smaller parameter" and the other one by Thomas Bläsius, Tobias Friedrich, Anton Krohmer and Sören Laue for their contribution "Efficient Embedding of Scale-Free Graphs in the Hyperbolic Plane". Our warmest congratulations to all of them for these achievements!

We wish to thank all the authors who submitted papers for consideration, the invited speakers, the members of the Program Committees for their hard work, and all the external reviewers who assisted the Program Committees in the evaluation process. Special thanks go to the Local Organizing Committee, who helped us with the organization of the conference.

June 2016

Piotr Sankowski
Christos Zaroliagis

Program Committee

Design and Analysis (Track A) Program Committee

Piotr Sankowski (Chair)	University of Warsaw, Poland
Alexandr Andoni	Columbia University, USA
Chen Avin	Ben Gurion University of The Negev, Israel
Sergio Cabello	University of Ljubljana, Slovenia
Parinya Chalermsook	Max Planck Institute for Informatics, Germany
Shiri Chechik	Tel-Aviv University, Israel
Holger Dell	Saarland University, Germany
Friedrich Eisenbrand	EPFL, Switzerland
Pierre Fraigniaud	CNRS and University Paris Diderot, France
Naveen Garg	Indian Institute of Technology Delhi, India
Pawel Gawrychowski	University of Wrocław, Poland
Bernd Gärtner	ETH Zurich, Switzerland
Bart M. P. Jansen	Eindhoven University of Technology, Netherlands
Piotr Krysta	University of Liverpool, UK
Lap Chi Lau	University of Waterloo, Canada
Pinyan Lu	Shanghai University of Finance and Economics, China
Ulrich Meyer	Goethe-Universität Frankfurt am Main, Germany
Danupon Nanongkai	KTH Royal Institute of Technology, Sweden
Michal Pilipczuk	University of Warsaw, Poland
Harald Räcke	Technische Universität München, Germany
Thomas Sauerwald	University of Cambridge, UK
Mohit Singh	Microsoft Research, USA
Christian Sohler	Technische Universität Dortmund, Germany
Paul Wollan	Sapienza University of Rome, Italy
Grigory Yaroslavtsev	University of Pennsylvania, USA

Engineering and Applications (Track B) Program Committee

Christos Zaroliagis (Chair)	CTI & University of Patras, Greece
Gianlorenzo D'Angelo	Gran Sasso Science Institute, Italy
Yann Disser	Technical University of Berlin, Germany
Daniele Frigioni	University of Aquila, Italy
Spyros Kontogiannis	CTI & University of Ioannina, Greece
Leszek Gasieniec	University of Liverpool, UK
Fabrizio Grandoni	IDSIA, University of Lugano, Switzerland
Giuseppe Italiano	University of Roma "Tor Vergata", Italy
Andreas Karrenbauer	Max Planck Institute for Informatics, Germany
Marco Luebbecke	RWTH Aachen University, Germany
Henning Meyerhenke	Karlsruhe Institute of Technology, Germany
Liam Roditty	Bar-Ilan University, Israel
Stefan Schirra	Otto-von-Guericke University Magdeburg, Germany
Nodari Sitchinava	University of Hawaii, Manoa, USA
Yuichi Yoshida	National Institute of Informatics, Japan

External Reviewers

Abboud, Amir
Abraham, Ittai
Acharya, Jayadev
Adamczyk, Marek
Agarwal, Pankaj
Aggarwal, Divesh
Ahn, Kook Jin
Ajwani, Deepak
Albers, Susanne
Aliasgari, Mehrdad
Alonso, Laurent
Alt, Helmut
Ambainis, Andris
Annamalai, Chidambaram
Assadi, Sepehr
Augustine, John
Aurenhammer, Franz
Backurs, Arturs
Balliu, Alkida
Bandyapadhyay, Sayan
Banik, Aritra
Bansal, Nikhil
Barba, Luis
Basit, Abdul
Batra, Jatin
Bei, Xiaohui
Belazzougui, Djamal
Bergamini, Elisabetta
Berkholz, Christoph
Bhattacharya, Sayan
Biedl, Therese
Bienkowski, Marcin
Bingmann, Timo
Biniaz, Ahmad
Bliznets, Ivan
Bodlaender, Hans L.
Bokal, Drago
Bonacina, Ilario
Bonnet, Edouard
Bosek, Bartłomiej
Bousquet, Nicolas
Brand, Cornelius
Brandes, Ulrik
Brendel, Ronny
Bringmann, Karl
Briskorn, Dirk
Bruhn, Henning
Buchin, Kevin
Buchin, Maike
Bulatov, Andrei
Bulian, Jannis
Bury, Marc
Byrka, Jaroslaw
Cai, Shaowei
Cai, Yang
Cao, Yixin
Carlucci, Lorenzo
Carmesin, Johannes
Cevallos, Alfonso
Chan, T-H. Hubert
Chandoo, Maurice
Chastain, Erick

Chen, Fei
Chen, Wei
Chrobak, Marek
Cicalese, Ferdinando
Cicerone, Serafino
Colella, Feliciano
Comandur, Seshadhri
Cormode, Graham
Coudert, David
Cummings, Rachel
Curticapean, Radu
Cygan, Marek
Czyzowicz, Jurek
D'Emidio, Mattia
Dadush, Daniel
Daeubel, Karl
Daltrophe, Hadassa
Damian, Mirela
Daniely, Amit
Danner, Andrew
Das, Gautam K
Das, Syamantak
De Carufel, Jean-Lou
de Haan, Ronald
De Keijzer, Bart
De Mesmay, Arnaud
de Zeeuw, Frank
De, Anindya
Deligkas, Argyrios
Devanur, Nikhil
Didier, Laurent-Stephane
Dietzfelbinger, Martin
Ding, Jian
Dinitz, Michael
Doty, David
Duan, Ran
Dudek, Bartlomiej
Dughmi, Shaddin
Dulęba, Maciej
Durocher, Stephane
Dvořák, Wolfgang
Edwards, Katherine
Efentakis, Alexandros
Efthymiou, Charilaos
Elbassioni, Khaled
Elsässer, Robert
Ene, Alina
Englert, Matthias
Eppstein, David
Erlebach, Thomas
Esfandiari, Hossein
Even, Guy
Faenza, Yuri
Farach-Colton, Martin
Farczadi, Linda
Fearnley, John
Feige, Uriel
Feldman, Moran
Felmdan, Dan
Felsner, Stefan
Fernau, Henning
Ferraioli, Diodato
Ferres, Leo
Fichtenberger, Hendrik
Filos-Ratsikas, Aris
Filtser, Arnold

Fineman, Jeremy
Fischer, Eldar
Fischer, Frank
Fischer, Johannes
Fleischer, Rudolf
Fomin, Fedor
Fox, Kyle
Fox-Epstein, Eli
Freedman, Ofer
Friedler, Ophir
Friedrich, Tobias
Fulek, Radoslav
Funke, Stefan
Gagie, Travis
Galvez, Waldo
Ganian, Robert
Gao, Jie
Gaspers, Serge
Georgiadis, Loukas
Giannopoulos, Panos
Glantz, Roland
Golan, Shay
Golovach, Petr
Goodrich, Michael
Goswami, Mayank
Gołębiewski, Mateusz
Green Larsen, Kasper
Groß, Martin
Grønlund, Allan
Gudmundsson, Joachim
Guo, Heng
Guo, Jiong
Gupta, Anupam
Göös, Mika
Hackfeld, Jan
Hajiaghayi, Mohammad Taghi
Halldórsson, Magnús
Han, Xin
Hassidim, Avinatan
Hirai, Hiroshi
Hoffmann, Michael
Holley, Guillaume
Holzer, Stephan
Hočevar, Tomaž
Huang, Chien-Chung
Huang, Sangxia
Huang, Zhiyi
Iacono, John
Im, Sungjin
Inenaga, Shunsuke
Ingala, Salvatore
Irving, Robert
Issac, Davis
Iwama, Kazuo
Iwata, Yoichi
Jacob, Riko
Jindal, Gorav
Joos, Felix
Jowhari, Hossein
Kakimura, Naonori
Kalaitzis, Christos
Kanellopoulos, Panagiotis
Kanté, Mamadou Moustapha
Kapralov, Michael
Karczmarz, Adam
Karsin, Ben

Kaski, Petteri
Kaufmann, Michael
Kawahara, Jun
Kawarabayashi, Ken-Ichi
Kell, Nathaniel
Keller, Orgad
Kempa, Dominik
Kerber, Michael
Kesselheim, Thomas
Khan, Arindam
Kim, Eun Jung
Kindermann, Philipp
Knauer, Christian
Kociumaka, Tomasz
Kodric, Bojana
Koivisto, Mikko
Komosa, Pawel
Konrad, Christian
Kopelowitz, Tsvi
Korman, Matias
Korshunov, Anton
Kothari, Pravesh
Kothari, Robin
Koucky, Michal
Kovacs, Annamaria
Kowalik, Lukasz
Kralovic, Rastislav
Krasnopolsky, Nadav
Kratsch, Stefan
Krinninger, Sebastian
Krishnaswamy, Ravishankar
Kuhnert, Sebastian
Kulkarni, Janardhan
Kumar, Amit
Kumar, Nikhil
Kunysz, Adam
Kwok, Tsz Chiu
Kwon, O-Joung
Laarhoven, Thijs
Laekhanukit, Bundit
Lampis, Michael
Lapinskas, John
Larisch, Lukas
Laue, Soeren
Lazarus, Francis
Le Gall, Francois
Le, Tien-Nam
Lee, Yin Tat
Leniowski, Dariusz
Leucci, Stefano
Li, Minming
Li, Shi
Li, Yang
Liaghat, Vahid
Limaye, Nutan
Liu, Chun-Hung
Liu, Jingcheng
Lokshtanov, Daniel
Lotker, Zvi
Loukas, Andreas
Luccio, Fabrizio
Lund, Benjamin
Löffler, Maarten
Łopuszański, Jakub
Łącki, Jakub
Makarychev, Konstantin

Mallmann-Trenn, Frederik
Manea, Florin
Mansour, Yishay
Manzini, Giovanni
Maria, Clément
Marino, Andrea
Martin, Russell
Martin-Recuerda, Francisco
Marx, Dániel
Meeks, Kitty
Mehlhorn, Kurt
Meir, Reshef
Meissner, Julie
Mertzios, George
Mestre, Julian
Meunier, Pierre-Étienne
Meyer Auf der Heide, Friedhelm
Milatz, Malte
Mitsou, Valia
Mnich, Matthias
Monemizadeh, Morteza
Moseley, Benjamin
Mukherjee, Koyel
Mulzer, Wolfgang
Munteanu, Alexander
Mészáros, Viola
Müller, Tobias
Navarra, Alfredo
Naves, Guyslain
Nayyeri, Amir
Nederlof, Jesper
Nelson, Jelani
Neumann, Stefan
Nicholson, Patrick K.
Nikita, Ivkin
Nikolov, Aleksandar
Nilsson, Bengt J.
Nimbhorkar, Prajakta
Noori Zehmakan, Abdolahad
Nummenpalo, Jerri
Nutov, Zeev
Okamoto, Yoshio
Olivetti, Dennis
Onak, Krzysztof
Ordyniak, Sebastian
Orecchia, Lorenzo
Oren, Sigal
Otachi, Yota
Oum, Sang-Il
Oveis Gharan, Shayan
Ozeki, Kenta
Palios, Leonidas
Panagiotou, Konstantinos
Panigrahi, Debmalya
Panigrahy, Rina
Panolan, Fahad
Papadopoulos, Fragkiskos
Parnas, Michal
Parotsidis, Nikos
Parter, Merav
Pasquale, Francesco
Patáková, Zuzana
Paul, Christophe
Peng, Pan
Peng, Richard
Penschuck, Manuel

Persiano, Giuseppe
Pettie, Seth
Philip, Geevarghese
Pilipczuk, Marcin
Pilz, Alexander
Pothitos, Nikolaos
Pountourakis, Emmanouil
Pruhs, Kirk
Prutkin, Roman
Puglisi, Simon
Pérez-Lantero, Pablo
Radoszewski, Jakub
Radzik, Tomasz
Raghavendra, Prasad
Raman, Rajeev
Rawitz, Dror
Razenshteyn, Ilya
Reidl, Felix
Richerby, David
Roth, Marc
Roy Choudhury, Anamitra
Rubinstein, Aviad
Rutter, Ignaz
Röglin, Heiko
Sabharwal, Yogish
Sagraloff, Michael
Saha, Barna
Salazar, Gelasio
Saranurak, Thatchaphol
Sau, Ignasi
Saurabh, Saket
Scalosub, Gabriel
Schaefer, Marcus
Scheffer, Christian
Schewior, Kevin
Schlag, Sebastian
Schlöter, Miriam
Schmid, Andreas
Schmid, Stefan
Schnider, Patrick
Schoenebeck, Grant
Schott, René
Schubert, Matthias
Schulz, André
Schulz, Christian
Schwartz, Roy
Schweitzer, Pascal
Schwiegelshohn, Chris
Segal, Michael
Severini, Lorenzo
Shepherd, Bruce
Shinkar, Igor
Sidiropoulos, Anastasios
Silas, Shashwat
Silvestri, Francesco
Simhadri, Harsha Vardhan
Sinnl, Markus
Skiena, Steven
Skopalik, Alexander
Skrepetos, Dimitrios
Skutella, Martin
Slivovsky, Friedrich
Smid, Michiel
Smolny, Frieder
Smorodinsky, Shakhar
Solomon, Shay

Sommer, Christian
Sorge, Manuel
Starikovskaya, Tatiana
Staudt, Christian
Stehn, Fabian
Strozecki, Yann
Su, Hsin-Hao
Sun, He
Szedlak, May
Takaguchi, Taro
Tang, Bo
Tang, Pingzhong
Tangwongsan, Kanat
Tarhio, Jorma
Telle, Jan Arne
Thaler, Justin
Thomas, Antonis
Tillmann, Stephan
Todinca, Ioan
Toledo, Sivan
Toma, Laura
Tov, Roei
Turkoglu, Duru
Tyagi, Hemant
Tönnis, Andreas
Türkoğlu, Duru
Umboh, Seeun William
Uniyal, Sumedha
Uno, Yushi
Uznański, Przemysław
Vahrenhold, Jan
van Iersel, Leo
van Leeuwen, Erik Jan
van Zuylen, Anke
Vassilevska Williams, Virginia
Vassilvitskii, Sergei
Vattani, Andrea
Vaz, Daniel
Vegh, Laszlo
Velaj, Yllka
Ventre, Carmine
Verbeek, Kevin
Verschae, José
Vijayaraghavan, Aravindan
Vinci, Cosimo
Vladu, Adrian
Vogtenhuber, Birgit
von Looz, Moritz
Vredeveld, Tjark
Wahlström, Magnus
Walter, Michael
Wang, Haitao
Wang, Xiao
Wang, Yusu
Ward, Justin
Wasa, Kunihiro
Watson, Thomas
Wegner, Michael
Weinberg, Matt
Wettstein, Manuel
Whitesides, Sue
Wiese, Andreas
Wismath, Steve
Wong, Prudence W.H.
Wong, Sam Chiu-Wai
Wood, David R.

Wrochna, Marcin

Wu, Zhiwei Steven

Wulff-Nilsen, Christian

Wötzel, Maximilian

Węgrzycki, Karol

Xia, Ge

Xia, Mingji

Xiao, Tao

Yekhanin, Sergey

Yu, Huacheng

Yukun, Cheng

Zadimoghaddam, Morteza

Zanetti, Luca

Zehavi, Meirav

Zenklusen, Rico

Zhang, Jialin

Zhang, Qiang

Zhang, Yumeng

2-Connectivity in Directed Graphs

Loukas Georgiadis[1], Giuseppe F. Italiano[*2], and Nikos Parotsidis[3]

1 **University of Ioannina, Ioannina, Greece**
`loukas@cs.uoi.gr`
2 **University of Rome Tor Vergata, Rome, Italy**
`giuseppe.italiano@uniroma2.it`
3 **University of Rome Tor Vergata, Rome, Italy**
`nikos.parotsidis@uniroma2.it`

Abstract

We survey some recent results on 2-edge and 2-vertex connectivity problems in directed graphs. Despite being complete analogs of the corresponding notions on undirected graphs, in digraphs 2-vertex and 2-edge connectivity have a much richer and more complicated structure. It is thus not surprising that 2-connectivity problems on directed graphs appear to be more difficult than on undirected graphs. For undirected graphs it has been known for over 40 years how to compute all bridges, articulation points, 2-edge- and 2-vertex-connected components in linear time, by simply using depth-first search. In the case of digraphs, however, the very same problems have been much more challenging and required the development of new tools and techniques.

1998 ACM Subject Classification E.1 [Graphs and Networks] Trees, F.2.2 [Nonnumerical Algorithms and Problems] Computations on Discrete Structures, G.2.2 [Graph Algorithms] Trees

Keywords and phrases 2-edge and 2-vertex connectivity on directed graphs, graph algorithms, dominator trees

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.1

Category Invited Talk

1 Preliminaries

Let $G = (V, E)$ be an *undirected* (resp., *directed*) graph, with m edges and n vertices. Throughout the paper, we use interchangeably the term directed graph and digraph. Edge and vertex connectivity are fundamental concepts in graph theory with numerous practical applications [2, 32]. As an example, we mention the computation of disjoint paths in routing and reliable communication, both in undirected and directed graphs [21, 24].

We assume that the reader is familiar with the standard graph terminology, as contained for instance in [7]. An *undirected path* (resp., *directed path*) in G is a sequence of vertices v_1, v_2, …, v_k, such that edge $(v_i, v_{i+1}) \in E$ for $i = 1, 2, \ldots, k-1$. An undirected graph G is *connected* if there is an undirected path from each vertex to every other vertex. The *connected components* of an undirected graph are its maximal connected subgraphs. A directed graph G is *strongly connected* if there is a directed path from each vertex to every other vertex. The *strongly connected components* of a directed graph are its maximal connected subgraphs.

* Partially supported by MIUR, the Italian Ministry of Education, University and Research, under Project AMANDA (Algorithmics for MAssive and Networked DAta).

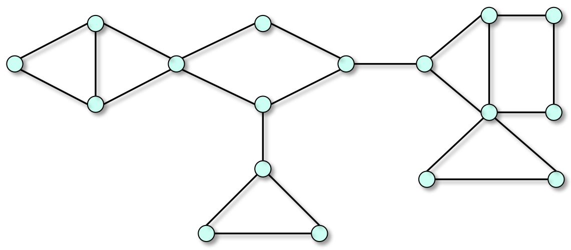


Figure 1 An undirected graph G.

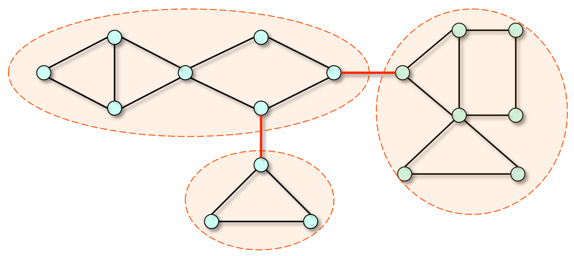


Figure 2 The bridges and 2-edge-connected components of the graph G in Figure 1. (Better viewed in color).

1.1 2-Connectivity in Undirected Graphs

Given an undirected graph $G = (V, E)$, an edge is a *bridge* if its removal increases the number of connected components of G. A graph G is 2-edge-connected if it has no bridges. The *2-edge-connected components* of G are its maximal 2-edge-connected subgraphs. Figure 1 shows an undirected graph, and Figure 2 highlights its bridges and 2-edge-connected components.

Two vertices v and w are 2-edge-connected if there are two edge-disjoint paths between v and w: we denote this relation by $v \leftrightarrow_{2\mathrm{e}} w$. Equivalently, by Menger's Theorem [31], v and w are 2-edge-connected if the removal of any edge leaves them in the same connected component.

Analogous definitions can be given for 2-vertex connectivity. In particular, a vertex is an *articulation point* if its removal increases the number of connected components of G. Figure 3 shows the articulation points of the graph in Figure 1. A graph G is 2-vertex-connected if it has at least three vertices and no articulation points. The 2-*vertex-connected components* of G are its maximal 2-vertex-connected subgraphs. Note that the condition on the minimum number of vertices in a 2-vertex-connected graph disallows degenerate 2-vertex-connected components consisting of one single edge. Figure 4 shows the 2-vertex-connected components of the graph in Figure 1.

Two vertices v and w are 2-vertex-connected if there are two internally vertex-disjoint paths between v and w: we denote this relation by $v \leftrightarrow_{2\mathrm{v}} w$. If v and w are 2-vertex-connected then Menger's Theorem implies that the removal of any vertex different from v and w leaves them in the same connected component. The converse does not necessarily hold, since v and w may be adjacent but not 2-vertex-connected. It is easy to show that $v \leftrightarrow_{2\mathrm{e}} w$ (resp., $v \leftrightarrow_{2\mathrm{v}} w$) if and only if v and w are in a same 2-edge-connected (resp., 2-vertex-connected) component.

All bridges, articulation points, 2-edge- and 2-vertex-connected components of undirected graphs can be computed in linear time essentially by the same algorithm, which is simply based on depth-first search [34].

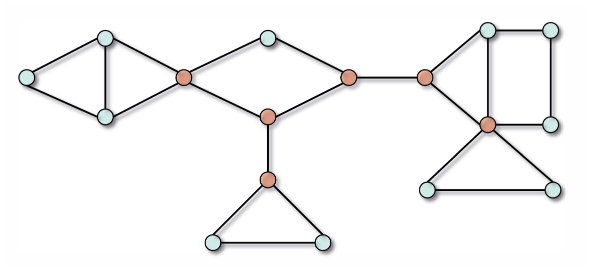


Figure 3 The articulation points of the graph G in Figure 1. (Better viewed in color).

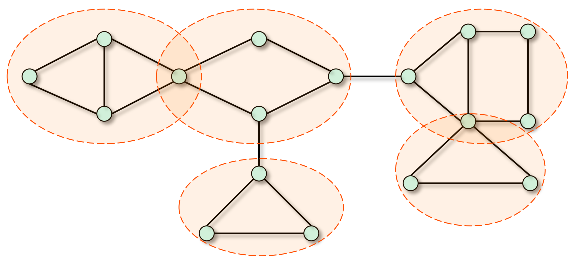


Figure 4 The 2-vertex-connected components of the graph G in Figure 1. (Better viewed in color).

1.2 2-Connectivity in Directed Graphs

The notions of 2-edge and 2-vertex connectivity can be naturally extended to directed graphs. The main idea is that now the role of connected components is played by strongly connected components. Given a digraph G, an edge (resp., a vertex) is a *strong bridge* (resp., a *strong articulation point*) if its removal increases the number of strongly connected components of G. A digraph G is 2-edge-connected if it has no strong bridges; G is 2-vertex-connected if it has at least three vertices and no strong articulation points. The 2-edge-connected (resp., 2-vertex-connected) components of G are its maximal 2-edge-connected (resp., 2-vertex-connected) subgraphs. Again, the condition on the minimum number of verti ces disallows for degenerate 2-vertex-connected components consisting of two mutually adjacent vertices (i.e., two vertices v and w and the two edges (v, w) and (w, v)).

Similarly to the undirected case, we say that two vertices v and w are 2-edge-connected (resp., 2-vertex-connected), and we denote again this relation by $v \leftrightarrow_{2\mathrm{e}} w$ (resp., $v \leftrightarrow_{2\mathrm{v}} w$), if there are two edge-disjoint (resp., internally vertex-disjoint) directed paths from v to w and two edge-disjoint (resp., internally vertex-disjoint) directed paths from w to v. (Note that a path from v to w and a path from w to v need not be edge-disjoint or vertex-disjoint). It is easy to see that $v \leftrightarrow_{2\mathrm{e}} w$ if and only if the removal of any edge leaves v and w in the same strongly connected component. Similarly, $v \leftrightarrow_{2\mathrm{v}} w$ implies that the removal of any vertex different from v and w leaves v and w in the same strongly connected component. We define a 2-*edge-connected block* (resp., 2-*vertex-connected block*) of a digraph $G = (V, E)$ as a maximal subset $B \subseteq V$ such that $u \leftrightarrow_{2\mathrm{e}} v$ (resp., $u \leftrightarrow_{2\mathrm{v}} v$) for all $u, v \in B$. Figure 5 illustrates (a) a strongly connected digraph G together with its strong articulation points and strong bridges, (b) the 2-vertex-connected components of G, (c) the 2-vertex-connected blocks of G, (d) the 2-edge-connected components of G, and (e) the 2-edge-connected blocks of G.

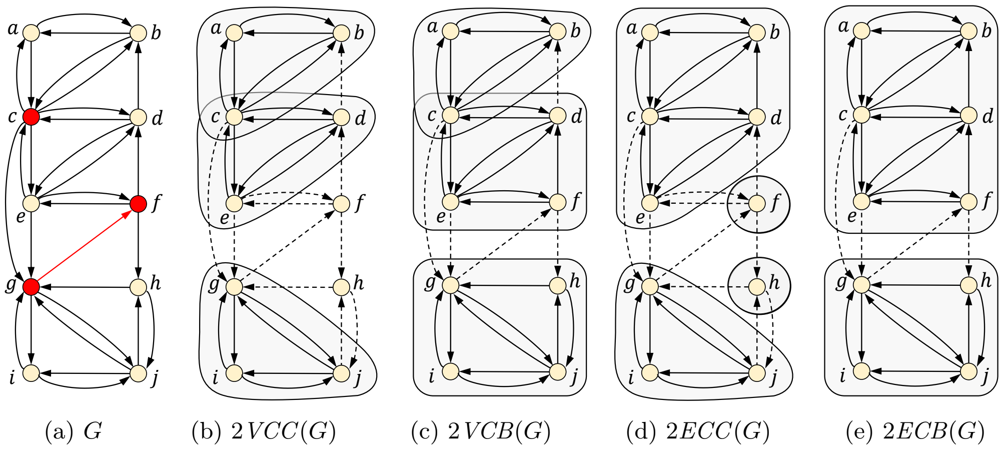


Figure 5 (a) A strongly connected digraph G, with strong articulation points and strong bridges shown in red (better viewed in color). (b) The 2-vertex-connected components of G. (c) The 2-vertex-connected blocks of G. (d) The 2-edge-connected components of G. (e) The 2-edge-connected blocks of G. Note that vertices e and f are in the same 2-vertex- (resp., 2-edge-) connected block of G since there are two internally vertex-disjoint (resp., edge-disjoint) paths from e to f and from f to e. However, e and f are not in the same 2-vertex (resp., 2-edge-) connected component of G. (Better viewed in color).

1.3 Differences between 2-Connectivity in Undirected and Directed Graphs

Connectivity-related problems for digraphs are notoriously harder than for undirected graphs, and indeed many notions for undirected connectivity do not translate to the directed case. Differently from undirected graphs, in digraphs 2-edge- and 2-vertex-connected blocks do not correspond to 2-edge- and 2-vertex-connected components, as it is clearly illustrated in Figure 5. Namely, two vertices may be 2-edge-connected (resp., 2-vertex-connected) but lie in different 2-edge-connected (resp., 2-vertex-connected) components. Furthermore, these notions seem to have a much richer and more complicated structure in digraphs, as depicted in Figure 6. Just to give an example, we observe that while in the case of undirected connected graphs the 2-edge-connected components (which correspond to the 2-edge-connected blocks) are exactly the connected components left after the removal of all bridges, for directed strongly connected graphs the 2-edge-connected components, the 2-edge-connected blocks, and the strongly connected components left after the removal of all strong bridges are not necessarily the same (see Figure 7).

Finally, we observe that an undirected graph is naturally decomposed by bridges (resp., articulation points) into a tree of 2-edge- (resp., 2-vertex-) connected components, known as the bridge-block (resp., block) tree (see, e.g., [36]). In digraphs, the decomposition induced by strong bridges and strong articulation points becomes much more complicated (see Figure 8): in general, it was shown by Benczúr that in digraphs there can be no “cut” tree for various connectivity concepts [3].

It is thus not surprising that, despite being complete analogs of the corresponding notions on undirected graphs, 2-edge and 2-vertex connectivity problems appear to be much more difficult on digraphs.

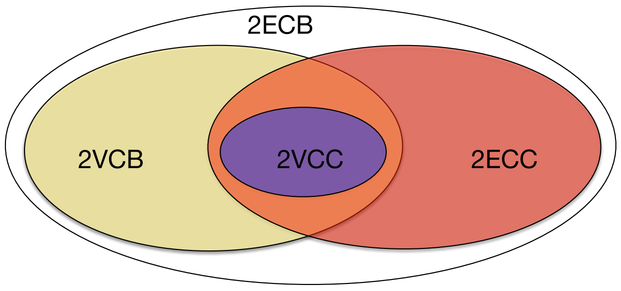


Figure 6 The relation among various notions of 2-connectivity in directed graphs. Two vertices that are 2-edge-connected (resp., 2-vertex-connected) are in the same 2-edge-connected (resp., 2-vertex-connected) block but not necessarily in the same 2-edge-connected (resp., 2-vertex-connected) component. Also, a 2-vertex-connected component is included in a 2-edge-connected component. (Better viewed in color).

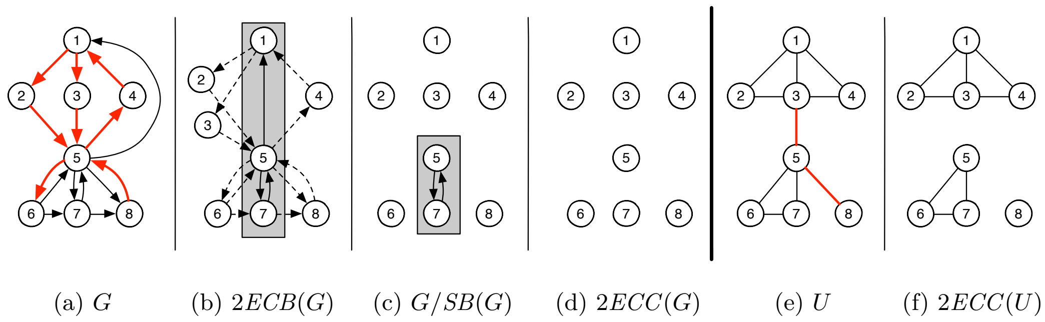
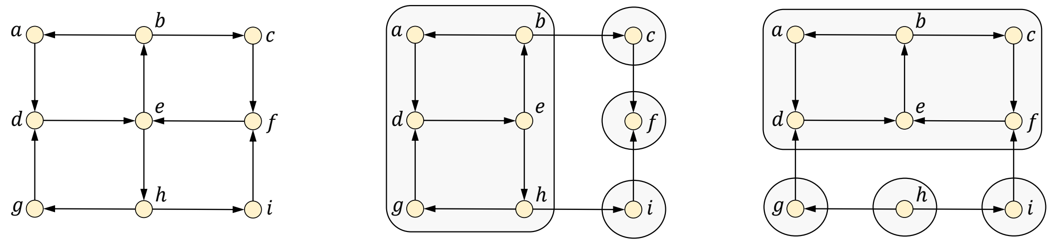


Figure 7 (a) A digraph G with strong bridges shown in red; (b) The 2-edge-connected blocks of G; (c) The strongly connected components left after removing all the strong bridges from G; (d) The 2-edge-connected components of G. (e) An undirected graph U with bridges shown in red; (f) The 2-edge-connected components of U, corresponding to the 2-edge-connected blocks and to the connected components left after the removal of all bridges of U. (Better viewed in color).

2 Simple-minded Algorithms for 2-edge and 2-vertex Connectivity in Directed Graphs

A simple algorithm for computing the 2-edge-connected components can be obtained by repeatedly removing all the strong bridges in the graph (and repeating this process until no strong bridges are left). At each round all the strong bridges can be computed in $O(m+n)$ time [26] and since there can be at most $O(n)$ rounds, the total time taken by this algorithm is $O(mn)$. The same bound was previously achieved by Nagamochi and Watanabe [33]. As for 2-vertex connectivity, Erusalimskii and Svetlov [9] proposed an algorithm that reduces the problem of computing the 2-vertex-connected components of a digraph to the computation of the 2-vertex-connected components in an undirected graph, but did not analyze the running time of their algorithm. Jaberi [28] showed that the algorithm of Erusalimskii and Svetlov has $O(nm^2)$ running time, and proposed two different algorithms with running time $O(mn)$. Both algorithms follow substantially the same high-level approach as the simple algorithm for computing the 2-edge-connected components of a digraph sketched before.

A simple algorithm for computing the 2-edge- or 2-vertex-connected blocks of a digraph G takes $O(mn)$ time: given a vertex v, one can find in linear time all the vertices that are



(a) G (b) $G \setminus (f, e)$ (c) $G \setminus (e, h)$

Figure 8 An example illustrating the complicated structure of 1-edge cuts in digraphs. (a) A strongly connected digraph G. (b) The strongly connected components in $G \setminus (f, e)$. (c) The strongly connected components in $G \setminus (e, h)$. Note that a strongly connected component in $G \setminus (f, e)$ and a strongly connected component in $G \setminus (e, h)$ are neither disjoint nor nested. In fact, all edges are strong bridges, and the deletion of each edge creates many non-disjoint and non-nested sets in the resulting partitions.

2-edge- or 2-vertex-connected with v with the help of dominator trees [11]. Since in the worst case this step must be repeated for all vertices v in G, the total time required by this simple algorithm is $O(mn)$. Very recently, Jaberi [27] presented algorithms for computing the 2-vertex-connected and 2-edge-connected blocks. His algorithms require $O(n \cdot \min\{m, b^*n\})$ time for computing the 2-edge-connected blocks and $O(n \cdot \min\{m, (a^* + b^*)n\})$ time for computing the 2-vertex-connected blocks, where a^* and b^* are respectively the number of strong articulation points and strong bridges in the digraph G. Since both a^* and b^* can be as large as $O(n)$, both bounds are $O(mn)$ in the worst case.

3 Flow Graphs and Dominators

In this section, we introduce some of the main tools that provided to be useful for solving 2-connectivity problems. Let $G = (V, E)$ be a strongly connected graph. Throughout, we denote by $G^R = (V, E^R)$ the *reverse digraph* of G, i.e., the digraph obtained by reversing the direction of all edges.

A *flow graph* is a digraph with a distinguished *start vertex* s such that every vertex is reachable from s. Let s be a fixed but arbitrary start vertex of a strongly connected digraph G. Since G is strongly connected, all vertices are reachable from s and reach s, so we can view both G and G^R as flow graphs with start vertex s. To avoid ambiguities, throughout the paper we will denote those flow graphs respectively by G_s and G_s^R. Vertex u is a *dominator* of vertex v (u *dominates* v) in G_s if every path from s to v in G_s contains u. Vertex u is a *proper dominator* of v if u dominates v and $u \neq v$. Let $dom(v)$ be the set of dominators of v. Clearly, $dom(s) = \{s\}$ and for any $v \neq s$ we have that $\{s, v\} \subseteq dom(v)$: we say that s and v are the *trivial dominators* of v in the flow graph G_s. The dominator relation is reflexive and transitive. Its transitive reduction is a rooted tree, the *dominator tree* D: u dominates v if and only if u is an ancestor of v in D. For any $v \neq s$, we denote by $d(v)$ the parent of v in D. Similarly, we can define the dominator relation in the flow graph G_s^R, and let D^R denote the dominator tree of G_s^R, and $d^R(v)$ the parent of v in D^R. Throughout the paper, we let N (resp., N^R) denote the set of nontrivial dominators of G_s (resp., G_s^R). Lengauer and Tarjan [30] presented an algorithm for computing dominators in $O(m\alpha(m, n))$ time for a flow graph with n vertices and m edges, where α is a functional inverse of Ackermann's function [35]. Subsequently, several linear-time algorithms were discovered [1, 4, 11, 12].

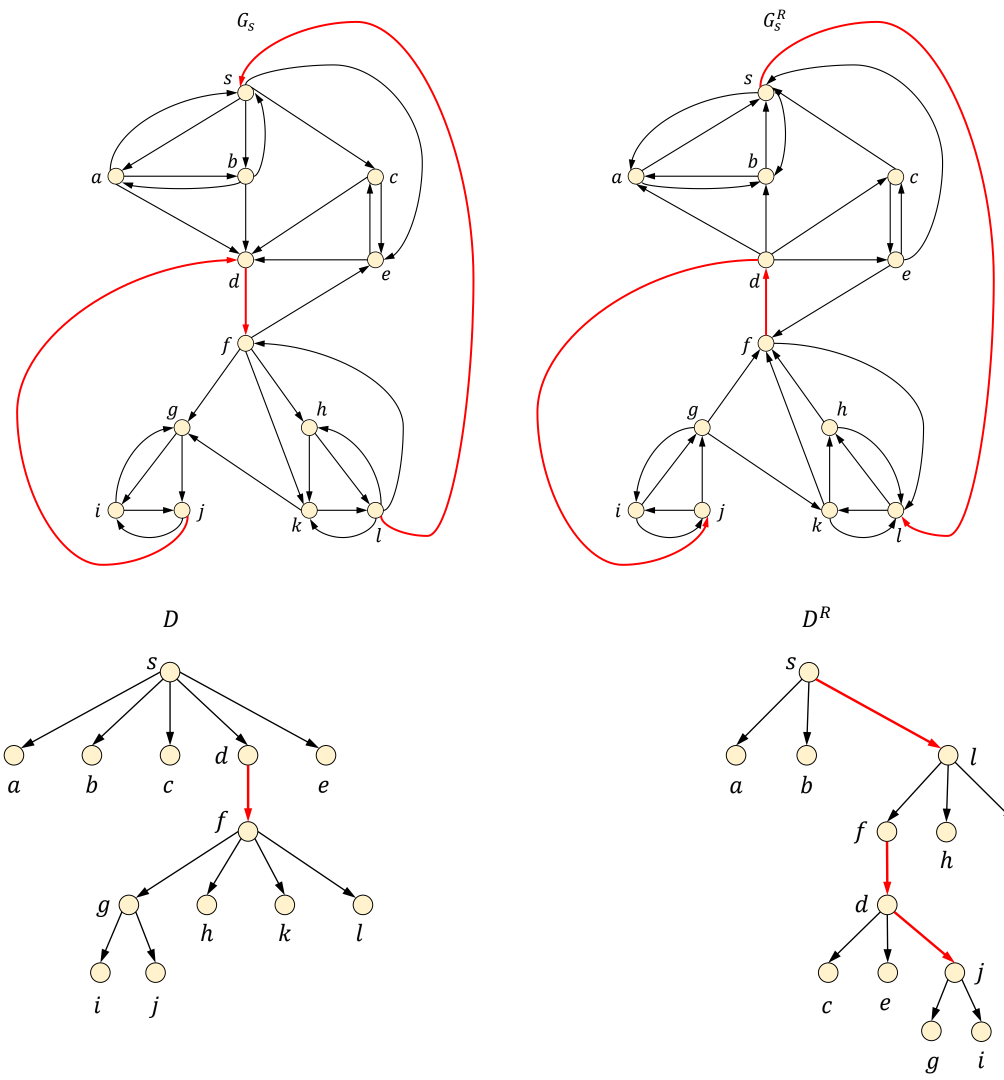


Figure 9 A flow graph G_s and its reverse G_s^R, and their dominator trees D and D^R. The corresponding digraph G is strongly connected. Strong bridges of G and G^R and bridges of G_s and G_s^R in D and D^R are shown red. (Better viewed in color.)

Figure 9 shows a flow graph G_s, its reverse G_s^R, and their dominator trees D and D^R.

An edge (u, v) is a *bridge* of a flow graph G_s if all paths from s to v include (u, v).[1] Let s be an arbitrary start vertex of G. As shown in [26], an edge $e = (u, v)$ is strong bridge of G if and only if it is either a bridge of G_s or a bridge of G_s^R. As a consequence, all the strong bridges of G can be obtained from the bridges of the flow graphs G_s and G_s^R, and thus there can be at most $2(n-1)$ strong bridges overall.

[1] Throughout the paper, to avoid confusion we use consistently the term *bridge* to refer to a bridge of a flow graph and the term *strong bridge* to refer to a strong bridge in the original graph.

4 Efficient Algorithms for 2-Connectivity in Directed Graphs

In this section, we show how to exploit the basic tools described in Section 3 to obtain fast algorithms for 2-connectivity in digraphs. We start by showing a connection between strong articulation points and dominators in flowgraphs. Consider the problem of finding all strong articulation points of a strongly connected digraph $G = (V, E)$. Let s be any vertex in G. Since G is strongly connected, every vertex in G is reachable from s: thus for every vertex $s \in V$, G_s is a flowgraph. Note that there can be n flowgraphs for each strongly connected graph. The following lemmas show a close relationship between strong articulation points in strongly connected graphs and non-trivial dominators in flow graphs.

▶ **Lemma 1** ([26]). *Let $G = (V, E)$ be a strongly connected graph, and let s be any vertex in G. Let G_s be the flowgraph with start vertex s. If a vertex u is a non-trivial dominator of a vertex v in G_s, then u is a strong articulation point in G.*

Proof. If u is a non-trivial dominator of v in the flowgraph G_s, then $u \neq s$, $u \neq v$ and all the paths in G from s to v must include u. Consequently, $G \setminus \{u\}$ is not strongly connected and thus u must be a strong articulation point in G. ◀

▶ **Lemma 2** ([26]). *Let $G = (V, E)$ be a strongly connected graph. If u is a strong articulation point in G, then there must be a vertex $s \in V$ such that u is a non-trivial dominator of a vertex v in the flowgraph G_s.*

Proof. If u is a strong articulation point of G, then there must exist two vertices s and v in G, $s \neq u$, $v \neq u$, such that every path from s to v contains vertex u. This implies that u must be a non-trivial dominator of vertex v in the flowgraph G_s. ◀

We note that Lemmas 1 and 2 are still not sufficient to achieve a linear-time algorithm for our problem: indeed, to compute all the strong articulation points of a strongly connected graph G, we need to compute all the non-trivial dominators in the flowgraphs $G(s)$, for each vertex s in V. Since the dominators of a flowgraph can be computed in $O(m + n)$ time [1, 4, 11, 12] and there are exactly n flowgraphs to be considered, the running time of this algorithm is $O(n(m + n))$. We show next how a more careful exploitation of the relationship between strong articulation points and dominators yields a linear-time algorithm for computing the strong articulation points of a directed graph.

▶ **Theorem 3** ([26]). *Let $G = (V, E)$ be a strongly connected graph, and let $s \in V$ be any vertex in G. Let G_s and G_s^R be respectively the flowgraphs with start vertex s, D and D^R their dominator trees, and N and N^R the non-trivial dominators in D and D^R. Then vertex $v \neq s$ is a strong articulation point in G if and only if $v \in N \cup N^R$.*

Proof. We first prove that if v is a strong articulation point in G, $v \neq s$, then v must be a non-trivial dominator either in D or in D^R. Assume not: namely, assume that v is a strong articulation point in G, $v \neq s$, but $v \notin N \cup N^R$. Since v is a strong articulation point in G, then $G \setminus \{v\}$ is not strongly connected. As a consequence, there must be a vertex w in G, $w \neq s$, $w \neq v$, such that the following is true: w is in the same strongly connected component as s in G, but w is not in the same strongly connected component as s in $G \setminus \{v\}$. Namely, $v \neq s$, $v \neq w$, and either (a) there is a path from s to w in G, but there is no path from s to w in $G \setminus \{v\}$, or (b) there is a path from w to s in G, but there is no path from w to s in $G \setminus \{v\}$. If we are in case (a), then all the paths from s to w in G must contain vertex v. This is equivalent to saying that v is a non-trivial dominator of w in the flowgraph G_s, which clearly contradicts our assumption that $v \notin N$. If we are in case (b), then all the paths from w to s in G must contain vertex v. This is equivalent to saying that v is a non-trivial dominator of w in the flowgraph G_s^R, which contradicts our assumption that $v \notin N^R$. This shows that if v is a strong articulation point in G, then v must be in N or in N^R.

To prove the converse, let v be any vertex such that $v \in N$ or $v \in N^R$. If $v \in N$, v is a non-trivial dominator in G_s, and thus v must be a strong articulation point in G by Lemma 1. Analogously, if $v \in N^R$, again by Lemma 1 v must be a strong articulation point in G^R, and thus in G. This completes the proof of the theorem. ◀

Note that Theorem 3 provides no information on whether vertex s is a strong articulation point. However, this can be easily checked in linear time, yielding the following theorem.

▶ **Theorem 4** ([26]). *All the strong articulation points of a directed graph G can be computed in $O(m+n)$ time in the worst case.*

The strong bridges of a directed graph can be found in an analogous fashion, giving rise to the following theorem:

▶ **Theorem 5** ([26]). *All the strong bridges of a directed graph G can be computed in $O(m+n)$ time in the worst case.*

A more sophisticated usage of dominator trees, combined with other properties, gives rise to efficient algorithms for computing the 2-edge-connected and 2-vertex-connected blocks and components of a directed graph, as stated in the remainder of this section.

In particular, Georgiadis et al. [16] gave a linear-time algorithms for computing the 2-edge-connected blocks of a digraph. Their approach hinges on two different algorithms. The first is a simple iterative algorithm that builds the 2-edge-connected blocks by removing one strong bridge at a time. The second algorithm is more involved and recursive: the main idea is to consider simultaneously how different strong bridges partition vertices with the help of dominator trees. Although both algorithms run in $O(mn)$ time in the worst case, Georgiadis et al. [16] showed that a sophisticated combination of the iterative and the recursive method is able to achieve a linear-time bound, as shown in the following theorem.

▶ **Theorem 6** ([16]). *The 2-edge-connected blocks of a directed graph G can be computed in $O(m+n)$ time in the worst case.*

Using the linear-time algorithm for computing the 2-edge-connected blocks, one can preprocess a digraph in linear time, and then can answer in constant time queries on whether any two vertices are 2-edge-connected. Additionally, when two query vertices v and w are not 2-edge-connected, one can produce in constant time a "witness" of this property, by exhibiting an edge that is contained in all paths from v to w or in all paths from w to v. As a consequence of the linear-time algorithm of Theorem 6, one can also compute in linear time a sparse certificate for 2-edge-connected blocks, i.e., a subgraph of the input graph that has $O(n)$ edges and maintains the same 2-edge-connected blocks as the input graph. The interested reader is referred to [16] for all the details.

Following the high-level approach of [16] for finding the 2-edge-connected blocks, Georgiadis et al. [17] were able to prove that also the 2-vertex-connected blocks of a digraph can be computed in linear time. The algorithm for computing the 2-vertex-connected blocks is much more involved than the 2-edge connectivity algorithm required several novel ideas and more sophisticated techniques to achieve the claimed bounds. Moreover, differently from 2-edge connectivity, 2-vertex connectivity in digraphs is plagued with several degenerate special cases, which are not only more tedious but also more cumbersome to deal with. For instance, the algorithm in [16] exploits implicitly the property that two vertices v and w are 2-edge-connected if and only if the removal of any edge leaves v and w in the same strongly connected component. Unfortunately, this property no longer holds for 2-vertex connectivity, as for instance two mutually adjacent vertices are always left in the same strongly connected component by the removal of any other vertex, but they are not necessarily 2-vertex-connected. This is summarized in the following theorem.

▶ **Theorem 7** ([17]). *The 2-vertex-connected blocks of a directed graph G can be computed in $O(m+n)$ time in the worst case.*

Similary to the case of 2-edge connectivity, other side results can be obtained as an application of this algorithm. In particular, one can construct an $O(n)$-space data structure that reports in constant time if two vertices are 2-vertex-connected. by exhibiting a vertex (i.e., a strong articulation point) or an edge (i.e., a strong bridge) that separates them. Once again, one can also compute in linear time a sparse certificate for 2-vertex connectivity, i.e., a subgraph of the input graph that has $O(n)$ edges and maintains the same 2-vertex connectivity properties.

We now turn to the problem of computing the 2-edge- and 2-vertex-connected components of a digraph. In this case, Henzinger et al. [23], presented fast algorithms for computing the 2-edge- and 2-vertex-connected components of a directed graph. The main idea behind their algorithm is a hierarchical graph sparsification that was introduced by Henzinger et al. [22] for undirected graphs and extended to directed graphs in [5]. Roughly speaking, this sparsification technique allows one to replace the m in the $O(mn)$ running times by an n, yielding $O(n^2)$ running times in place of $O(mn)$. Henzinger et al. [23] were able to find structural properties of 2-edge and 2-vertex connectivity in directed graphs that allow one to apply this technique starting from the simple-minded $O(mn)$ algorithms and a clever use of dominators. Those bounds are summarized in the following theorem.

▶ **Theorem 8** ([23]). *The 2-edge- and 2-vertex-connected components of a directed graph G can be computed in $O(n^2)$ time in the worst case.*

Additionally, Henzinger et al. [23] presented an $O(m^2/\log n)$ time algorithm for computing the 2-edge-connected components, which provides a small improvement for sparse graphs, i.e., $m = O(n)$. The same approach can be extended to k-edge- and k-vertex-connected components, for any constant k, with a running time of $O(n^2 \log n)$ for k-edge connectivity and $O(n^3)$ for k-vertex connectivity.

Finally, we mention that Georgiadis et al. [19] initiated the study of the dynamic maintenance of 2-edge-connectivity relationships in directed graphs. In particular, they presented an algorithm that can update the 2-edge-connected blocks of a digraph G with n vertices through a sequence of m edge insertions in a total of $O(mn)$ time. After each insertion, one can answer the following queries in asymptotically optimal time:

- Test in constant time if two query vertices v and w are 2-edge-connected. Moreover, if v and w are not 2-edge-connected, one can produce in constant time a “witness” of this property, by exhibiting an edge that is contained in all paths from v to w or in all paths from w to v.
- Report in $O(n)$ time all the 2-edge-connected blocks of G.

▶ **Theorem 9** ([19]). *The 2-edge-connected blocks of a digraph with n vertices can be maintained through a sequence of edge insertions in $O(mn)$ time, where m is the total number of edges in G after all insertions.*

We remark that this is the first known dynamic algorithm for 2-connectivity problems on digraphs, and it matches the best known bounds for simpler problems, such as incremental transitive closure [25].

5 Sparse Subgraphs Preserving 2-Connectivity in Directed Graphs

Other problems that were considered in the area of 2-connectivity for directed graphs are related to the computation of a minimum spanning subgraph (i.e., a subgraph with minimum number of edges) that maintains certain 2-connectivity requirements in addition to strong connectivity. More specifically, one problem that has been investigated is finding a smallest strongly connected spanning subgraph of a digraph G that has the same 2-edge- (respectively, 2-vertex-) connectivity properties as G. Both for 2-edge- and for 2-vertex connectivity, this problem is known to be NP-hard [13, 18]. We next review some of the algorithms proposed in the literature respectively for edge and for vertex connectivity.

Laekhanukit et al. [29] gave a randomized $(1+1/k)$-approximation algorithm for computing the smallest k-edge-connected spanning subgraph of a k-edge-connected graph. Georgiadis et al. [18] used the algorithm in [29] to compute a 3/2-approximate minimum spanning subgraph that has the same 2-edge-connected components and additionally presented a faster 2-approximation algorithm that runs in linear time. Let G be a strongly connected graph. Jaberi [27] considered the problem of computing a smallest subgraph that has the same 2-edge-connected blocks (or the same 2-vertex-connected blocks) as G. Unfortunately, the approximation ratio in Jaberi's algorithms is $O(n)$ in the worst case. Georgiadis et al. [18] improved this result by presenting a linear-time 4-approximation algorithm for computing the smallest strongly connected spanning subgraph that has the same 2-edge-connected blocks as G. Additionally, they presented a linear-time algorithm for the problem of computing the smallest subgraph that has both the same 2-edge-connected components and the same 2-edge-connected blocks as G. The algorithms in [18] that compute spanning subgraphs with the same 2-edge-connected components as G run in linear time once the 2-edge-connected components of G are available (we remark that the currently best known bound for computing the 2-edge-connected components is $O(n^2)$ [23]).

For the smallest k-vertex-connected spanning subgraph, Cheriyan and Thurimella [6], gave a $(1+1/k)$-approximation algorithm that runs in $O(km^2)$ time. For $k=2$, Georgiadis [14] presented a linear time algorithm with approximation ratio 3. Based on the algorithm from [14], the running time of Cheriyan and Thurimella's algorithm was improved to $O(m\sqrt{n}+n^2)$ for $k=2$. Let G be a strongly connected graph. Georgiadis et al. [15] presented a constant-factor approximation algorithm for the problem of computing the smallest subgraph that preserves the 2-vertex-connected blocks of G. More specifically, they gave a linear-time 6-approximation algorithm for this problems, and further extended this algorithm to compute a sparse subgraph with the same approximation gurantee that has both the same 2-vertex-connected components and the same 2-vertex-connected blocks as G. The algorithm that computes a sparse subgraph that preserves both the 2-vertex-connected blocks and the 2-vertex-connected components of the input graph G runs in linear time, once the 2-vertex-connected components of G are available (we remark that the currently best known bound for computing the 2-vertex-connected components is $O(n^2)$ [23]). Finally, in [15] Georgiadis et al. presented a 6-approximation algorithm for computing a strongly connected spanning subgraph of G that preserves all the 2-connectivity relations, i.e., both the 2-edge- and the 2-vertex-connected components and the 2-edge- and the 2-vertex-connected blocks. Once again, this algorithm runs in linear time, provided that the 2-edge- and the 2-vertex-connected components of G are available.

We remark that references [15, 18] provide efficient implementations of all those approximation algorithms that run very fast in practice. Additionally, they also present several heuristics that improve the quality (i.e., the number of edges) of the computed spanning subgraphs, and assess how all these algorithms perform in practical scenarios by conducting a thorough experimental study.

6 Conclusions and Open Problems

We have surveyed some very recent results on 2-edge and 2-vertex connectivity problems in directed graphs, which revealed to be harder than their counterparts on undirected graphs. Experimental studies for algorithms that compute dominators, strong bridges, strong articulation points, 2-edge- and 2-vertex-connected blocks are presented in [8, 10, 20]. Those experimental results are very promising, as they show that the corresponding fast algorithms given in [11, 16, 17, 26] perform very well in practice even on very large graphs.

This recent bulk of work has raised some interesting and perhaps intriguing questions. In particular, the main open problem is whether the 2-edge-connected or the 2-vertex-connected components of a digraph can be computed in linear time. Moreover, the dynamic maintenance of 2-edge and 2-vertex connectivity in directed graphs deserves further investigation. Finally, we have described in Section 5 linear-time constant-factor approximation algorithms for computing minimum spanning subgraphs that preserve the 2-edge- and 2-vertex-connected blocks of a graph [15, 18]. The trade-offs between running times and approximation guarantees need further study. In particular, can the approximation guarantees in [15, 18] be improved while still maintaining linear running times? Can they match the corresponding approximation ratios for computing the 2-edge- and 2-vertex-connected spanning subgraphs of 2-edge- and 2-vertex-connected graphs [6, 29], respectively?

References

1 S. Alstrup, D. Harel, P. W. Lauridsen, and M. Thorup. Dominators in linear time. *SIAM Journal on Computing*, 28(6):2117–32, 1999.

2 J. Bang-Jensen and G. Gutin. *Digraphs: Theory, Algorithms and Applications (Springer Monographs in Mathematics)*. Springer, 1st ed. 2001. 3rd printing edition, 2002.

3 A. A. Benczúr. Counterexamples for directed and node capacitated cut-trees. *SIAM J. Comput.*, 24:505–510, 1995.

4 A. L. Buchsbaum, L. Georgiadis, H. Kaplan, A. Rogers, R. E. Tarjan, and J. R. Westbrook. Linear-time algorithms for dominators and other path-evaluation problems. *SIAM Journal on Computing*, 38(4):1533–1573, 2008.

5 K. Chatterjee and M. Henzinger. Efficient and dynamic algorithms for alternating büchi games and maximal end-component decomposition. *J. ACM*, 61(3):15:1–15:40, 2014. `doi:10.1145/2597631`.

6 J. Cheriyan and R. Thurimella. Approximating minimum-size k-connected spanning subgraphs via matching. *SIAM J. Comput.*, 30(2):528–560, 2000.

7 T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. The MIT Press, 2001.

8 W. Di Luigi, L. Georgiadis, G. F. Italiano, L. Laura, and N. Parotsidis. 2-connectivity in directed graphs: An experimental study. In *Proc. 17th Workshop on Algorithm Engineering and Experiments*, pages 173–187, 2015. `doi:10.1137/1.9781611973754.15`.

9 Ya. M. Erusalimskii and G. G. Svetlov. Bijoin points, bibridges, and biblocks of directed graphs. *Cybernetics*, 16(1):41–44, 1980. `doi:10.1007/BF01099359`.

10 D. Firmani, G. F. Italiano, L. Laura, A. Orlandi, and F. Santaroni. Computing strong articulation points and strong bridges in large scale graphs. In *Proc. 10th Int'l. Symp. on Experimental Algorithms*, pages 195–207, 2012.

11 W. Fraczak, L. Georgiadis, A. Miller, and R. E. Tarjan. Finding dominators via disjoint set union. *Journal of Discrete Algorithms*, 23:2–20, 2013. doi:10.1016/j.jda.2013.10.003.

12 H. N. Gabow. The minset-poset approach to representations of graph connectivity. *ACM Transactions on Algorithms*, 12(2):24:1–24:73, February 2016. doi:10.1145/2764909.

13 M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1979.

14 L. Georgiadis. Approximating the smallest 2-vertex connected spanning subgraph of a directed graph. In *Proc. 19th European Symposium on Algorithms*, pages 13–24, 2011.

15 L. Georgiadis, G. F. Italiano, A. Karanasiou, C. Papadopoulos, and N. Parotsidis. Sparse subgraphs for 2-connectivity in directed graphs. In *Proc. 15th Int'l. Symp. on Experimental Algorithms*, pages 150–166, 2016. doi:10.1007/978-3-319-38851-9_11.

16 L. Georgiadis, G. F. Italiano, L. Laura, and N. Parotsidis. 2-edge connectivity in directed graphs. In *Proc. 26th ACM-SIAM Symp. on Discrete Algorithms*, pages 1988–2005, 2015.

17 L. Georgiadis, G. F. Italiano, L. Laura, and N. Parotsidis. 2-vertex connectivity in directed graphs. In *Proc. 42nd Int'l. Coll. on Automata, Languages, and Programming*, pages 605–616, 2015.

18 L. Georgiadis, G. F. Italiano, C. Papadopoulos, and N. Parotsidis. Approximating the smallest spanning subgraph for 2-edge-connectivity in directed graphs. In *ESA 2015*, pages 582–594, 2015.

19 L. Georgiadis, G. F. Italiano, and N. Parotsidis. Incremental 2-edge connectivity in directed graphs. In *Proc. 43rd Int'l. Coll. on Automata, Languages, and Programming*, 2016. To appear.

20 L. Georgiadis, L. Laura, N. Parotsidis, and R. E. Tarjan. Loop nesting forests, dominators, and applications. In *Proc. 13th Int'l. Symp. on Experimental Algorithms*, pages 174–186, 2014.

21 Y. Guo, F. Kuipers, and P. Van Mieghem. Link-disjoint paths for reliable QoS routing. *International Journal of Communication Systems*, 16(9):779–798, 2003. doi:10.1002/dac.612.

22 M. Henzinger, V. King, and T. J. Warnow. Constructing a tree from homeomorphic subtrees, with applications to computational evolutionary biology. *Algorithmica*, 24(1):1–13, 1999. doi:10.1007/PL00009268.

23 M. Henzinger, S. Krinninger, and V. Loitzenbauer. Finding 2-edge and 2-vertex strongly connected components in quadratic time. In *Proc. 42nd Int'l. Coll. on Automata, Languages, and Programming*, pages 713–724, 2015.

24 A. Itai and M. Rodeh. The multi-tree approach to reliability in distributed networks. *Information and Computation*, 79(1):43–59, 1988.

25 G. F. Italiano. Amortized efficiency of a path retrieval data structure. *Theor. Comput. Sci.*, 48(3):273–281, 1986. doi:10.1016/0304-3975(86)90098-8.

26 G. F. Italiano, L. Laura, and F. Santaroni. Finding strong bridges and strong articulation points in linear time. *Theoretical Computer Science*, 447:74–84, 2012. doi:10.1016/j.tcs.2011.11.011.

27 R. Jaberi. Computing the 2-blocks of directed graphs. *RAIRO-Theor. Inf. Appl.*, 49(2):93–119, 2015. doi:10.1051/ita/2015001.

28 R. Jaberi. On computing the 2-vertex-connected components of directed graphs. *Discrete Applied Mathematics*, 204:164–172, 2016. doi:10.1016/j.dam.2015.10.001.

29 B. Laekhanukit, S. O. Gharan, and M. Singh. A rounding by sampling approach to the minimum size k-arc connected subgraph problem. In *ICALP 2012*, pages 606–616, 2012.

30 T. Lengauer and R. E. Tarjan. A fast algorithm for finding dominators in a flowgraph. *ACM Transactions on Programming Languages and Systems*, 1(1):121–41, 1979.

31 K. Menger. Zur allgemeinen kurventheorie. *Fund. Math.*, 10:96–115, 1927.

32 H. Nagamochi and T. Ibaraki. *Algorithmic Aspects of Graph Connectivity.* Cambridge University Press, 2008. 1st edition.

33 H. Nagamochi and T. Watanabe. Computing k-edge-connected components of a multigraph. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E76–A(4):513–517, 1993.

34 R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

35 R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22(2):215–225, 1975.

36 J. Westbrook and R. E. Tarjan. Maintaining bridge-connected and biconnected components on-line. *Algorithmica*, 7(5&6):433–464, 1992. doi:10.1007/BF01758773.

Algorithms with Provable Guarantees for Clustering

Ola Svensson

School of Computer and Communication Sciences, EPFL, Zürich, Switzerland
`ola.svensson@epfl.ch`

Abstract

In this talk, we give an overview of the current best approximation algorithms for fundamental clustering problems, such as k-center, k-median, k-means, and facility location. We focus on recent progress and point out several important open problems.

For the uncapacitated versions, a variety of algorithmic methodologies, such as LP-rounding and primal-dual method, have been applied to a standard linear programming relaxation. This has given a uniform way of addressing these problems resulting in small constant approximation guarantees.

In spite of this impressive progress, it remains a challenging open problem to give tight guarantees. Moreover, this collection of powerful algorithmic techniques is not easily applicable to the capacitated setting. In fact, there is no simple strong convex relaxation known for the capacitated versions. As a result, our understanding of these problems is significantly weaker and several fundamental questions remain open.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.2

Category Invited Talk

Beating Ratio 0.5 for Weighted Oblivious Matching Problems

Melika Abolhassani[*1], **T.-H. Hubert Chan**[†2], **Fei Chen**[‡3], **Hossein Esfandiari**[§4], **MohammadTaghi Hajiaghayi**[¶5], **Hamid Mahini**[‖6], **and Xiaowei Wu**[7]

1 **Department of Computer Sicence, University of Maryland, College Park, USA**
`melika@cs.umd.edu`
2 **Department of Computer Sicence, The University of Hong Kong, Hong Kong**
`hubert@cs.hku.hk`
3 **KTH Royal Institute of Technology, Stockholm, Sweden**
`feichen@kth.se`
4 **Department of Computer Sicence, University of Maryland, College Park, USA**
`hossein@cs.umd.edu`
5 **Department of Computer Sicence, University of Maryland, College Park, USA**
`hajiagha@cs.umd.edu`
6 **Department of Computer Sicence, University of Maryland, College Park, USA**
`hmahini@cs.umd.edu`
7 **Department of Computer Sicence, The University of Hong Kong, Hong Kong**
`xwwu@cs.hku.hk`

Abstract

We prove the first non-trivial performance ratios strictly above 0.5 for weighted versions of the oblivious matching problem. Even for the unweighted version, since Aronson, Dyer, Frieze, and Suen first proved a non-trivial ratio above 0.5 in the mid-1990s, during the next twenty years several attempts have been made to improve this ratio, until Chan, Chen, Wu and Zhao successfully achieved a significant ratio of 0.523 very recently (SODA 2014). To the best of our knowledge, our work is the first in the literature that considers the node-weighted and edge-weighted versions of the problem in arbitrary graphs (as opposed to bipartite graphs).

(1) For arbitrary node weights, we prove that a weighted version of the Ranking algorithm has ratio strictly above 0.5. We have discovered a new structural property of the ranking algorithm: if a node has two unmatched neighbors at the end of algorithm, then it will still be matched even when its rank is demoted to the bottom. This property allows us to form LP constraints for both the node-weighted and the unweighted oblivious matching problems. As a result, we prove that the ratio for the node-weighted case is at least 0.501512. Interestingly via the structural

[*] Partially supported by NSF CAREER award CCF-1053605, NSF BIGDATA grant IIS-1546108, NSF AF:Medium grant CCF-1161365, DARPA GRAPHS/AFOSR grant FA9550-12-1-0423, and another DARPA SIMPLEX grant.
[†] Partially supported by a grant from Hong Kong RGC under the contract 17200214.
[‡] Partially supported by Swedish Research Council grant 2015-04659 "Algorithms and Complexity for Dynamic Graph Problems".
[§] Partially supported by NSF CAREER award CCF-1053605, NSF BIGDATA grant IIS-1546108, NSF AF:Medium grant CCF-1161365, DARPA GRAPHS/AFOSR grant FA9550-12-1-0423, and another DARPA SIMPLEX grant.
[¶] Partially supported by NSF CAREER award CCF-1053605, NSF BIGDATA grant IIS-1546108, NSF AF:Medium grant CCF-1161365, DARPA GRAPHS/AFOSR grant FA9550-12-1-0423, and another DARPA SIMPLEX grant.
[‖] Partially supported by NSF CAREER award CCF-1053605, NSF BIGDATA grant IIS-1546108, NSF AF:Medium grant CCF-1161365, DARPA GRAPHS/AFOSR grant FA9550-12-1-0423, and another DARPA SIMPLEX grant.

property, we can also improve slightly the ratio for the unweighted case to 0.526823 (from the previous best 0.523166 in SODA 2014).

(2) For a bounded number of distinct edge weights, we show that ratio strictly above 0.5 can be achieved by partitioning edges carefully according to the weights, and running the (unweighted) Ranking algorithm on each part. Our analysis is based on a new primal-dual framework known as *matching coverage*, in which dual feasibility is bypassed. Instead, only dual constraints corresponding to edges in an optimal matching are satisfied. Using this framework we also design and analyze an algorithm for the edge-weighted online bipartite matching problem with free disposal. We prove that for the case of bounded online degrees, the ratio is strictly above 0.5.

1998 ACM Subject Classification G.1.2 Approximation, G.1.6 Optimization, G.2.1 Combinatorics, G.2.2 Graph Theory

Keywords and phrases weighted matching, oblivious algorithms, Ranking, linear programming

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.3

1 Introduction

While the classical maximum matching problem [14] is well understood, the oblivious version is motivated by exchange settings [15] and online advertising [9, 1], in which information about the underlying graphs might be unknown. For instance, in the kidney exchange problem [15], donor-recipient pairs are probed and greedily matched when two pairs are compatible. Another example is pay-per-click online advertising, in which the revenue for a click on a particular ad showing on a particular page is known, but it is unknown whether the user will actually click on that ad. In this paper, we analyze two weighted versions of the oblivious matching problem (ObMP). To be more specific, we first state the edge-weighted (Ew) ObMP (and the node-weighted (Nw) version as a special case) formally as follows.

EwObMP. An *adversary* commits to a simple undirected graph $G = (V, E)$, where **every** unordered pair of nodes $e = \{u, v\}$ (even if $e \notin E$) has non-negative weight w_e. The *unweighted* case is the special case in which all pairs have the same weight. The nodes V (where $n = |V|$) and the weights of all pairs are revealed to the (randomized) *algorithm*, while the edges E are kept secret. The algorithm returns a list L that gives a permutation of the set $\binom{V}{2}$ of unordered pairs of nodes. Each pair of nodes in G is probed according to the order specified by L to form a matching greedily. In the round when a pair $e = \{u, v\}$ is probed, if both nodes are currently *unmatched* and the edge e is in E, then the two nodes will be *matched* to each other; otherwise, we skip to the next pair in L until all pairs in L are probed. The goal is to maximize the ***performance ratio*** of the (expected) sum of weights of edges in the matching produced by the algorithm to that of a maximum weight matching in G. The node-weighted version is related to the edge-weighted version as follows.

NwObMP. The node-weighted version is a special case of EwObMP in which each node $u \in V$ has a non-negative weight w_u and the weight of each pair $e = \{u, v\}$ is $w_e = w_u + w_v$.

Greedy Algorithms. Greedy algorithms can achieve ratio 0.5 for both the edge-weighted and node-weighted versions. For the edge-weighted version, the probing order is given by sorting pairs in non-increasing order of weight. For the node-weighted version, the nodes are sorted in non-increasing order of weight to induce a lexicographical order on the pairs. As far as we know, this work is the first in the literature to achieve algorithms for both weighted versions with ratios strictly greater than 0.5.

To achieve non-trivial ratios, different variants of the Ranking algorithm have been investigated for various matching problems [12, 1, 6, 5]. We analyze the following variant that is relevant to NwObMP on arbitrary graphs.

Weighted Ranking Algorithm for NwObMP. Given the node weights w, the algorithm determines a distribution $\mathcal{D}_w$ on permutations of V. It samples a permutation π from $\mathcal{D}_w$, and returns a list L of unordered pairs according to the lexicographical order induced by π, where nodes appearing earlier in the permutation have higher priority. Specifically, for a permutation $\pi : V \to [n]$, given two pairs e_1 and e_2 (where for each j, $e_j = \{u_j, v_j\}$ and $\pi(u_j) < \pi(v_j)$), the pair e_1 has higher priority than e_2 if (i) $\pi(u_1) < \pi(u_2)$, or (ii) $u_1 = u_2$ and $\pi(v_1) < \pi(v_2)$.

Sampling a permutation. Previous works [1, 6] have considered the following way to sample a permutation of nodes. The algorithm uses an *adjustment function* $\varphi(t) := 1 - e^{t-1}$ for $t \in [0,1]$, and samples a *configuration* $\sigma \in \Omega_\infty := [0,1]^V$ uniformly at random, i.e., each node u receives independently a random number $\sigma(u)$ in $[0,1]$ uniformly at random. A permutation is given by sorting the nodes in non-increasing order of the *adjusted weight* $w(\sigma, u) := \varphi(\sigma(u)) \cdot w_u$. Observe that for the unweighted case (i.e., all nodes have the same weight), this is equivalent to sampling a permutation uniformly at random. We consider different adjustment functions φ in this paper.

1.1 Summary of Our Results

Extending previous linear programming (LP) approaches [1, 13, 11, 5], we prove that a weighted Ranking algorithm has ratio greater than 0.5 for NwObMP with arbitrary node weights in general graphs.

▶ **Theorem 1** (Weighted Ranking for NwObMP). *For $m = 10000$, weighted* Ranking *using the discrete sample space $[0,1]_m^V$ (where $[0,1]_m := \{\frac{i}{m} : i \in [m]\}$ is a discretization of $[0,1]$) and adjustment function $\varphi(t) := 1 - \frac{e^{17t}-1}{e^{17}-1}$ has performance ratio at least 0.501505.*

In the analysis, we have discovered new structural properties of the Ranking algorithm. For instance, if a node has two unmatched neighbors, then it will still be matched even when its rank is demoted to the bottom. These properties enable us to form better LP constraints. We use continuous LP techniques to prove that the above ratio can be improved to 0.501512 if continuous random sample space $[0,1]^V$ is used (due to space constraints, the complete proof is deferred to the full version). Interestingly via these structural properties, we also improve the analysis of (unweighted) Ranking for the unweighted ObMP over the previous best ratio of 0.523166 in the SODA 2014 paper [5].

▶ **Theorem 2** (Ranking for Unweighted ObMP). *The* Ranking *algorithm for unweighted* ObMP *has performance ratio at least* 0.526823.

For EwObMP with a bounded number of distinct edge weights, we show that ratio strictly above 0.5 can be achieved by partitioning edges carefully according to the weights, and running the (unweighted) Ranking algorithm on each part.

▶ **Theorem 3** (EwObMP with Bounded Number of Distinct Weights). *Suppose there is an algorithm on unweighted* ObMP *with performance ratio $\frac{1}{2} + \xi_1$. Then, for each positive integer*

$k > 1$, there exists $\xi_k = \Omega(\xi_1)^{O(k^2)}$ such that the following holds. There exists an algorithm for EwObMP *such that on instances with k distinct edge weights, the performance ratio is at least $\frac{1}{2} + \xi_k$.*

Our analysis is based on a new primal-dual framework of the standard matching LP known as *matching coverage*, in which dual feasibility is bypassed. Instead, only dual constraints corresponding to edges in an optimal matching are satisfied. Indeed the framework of matching coverage introduced for weighted oblivious matching has applications for other well-known problems. In particular using this framework we also design and analyze an algorithm for the *edge-weighted online bipartite matching problem with free disposal.* We prove that for the case of bounded online degrees, the ratio is strictly above 0.5.

EwOnBiMP with free disposal. An adversary fixes an edge-weighted bipartite graph $G(U \cup V, E)$ between a set U of *online* nodes and a set V of *offline* nodes, and determines the arrival order of the online nodes. When an online node u arrives, all the weights w_{uv}'s of edges between u and the offline nodes v in V are revealed to the (randomized) algorithm. The algorithm matches u to one of the offline nodes v. Even if an offline node v is already matched to a previous online node u', the algorithm is allowed to dispose of the edge $\{u', v\}$ and include the edge $\{u, v\}$ in the matching. The goal is to maximize the performance ratio, which is the (expected) sum of weights of edges in the final matching to that of a maximum weight matching in hindsight.

Feldman et al. [8] proved that a greedy algorithm can achieve ratio 0.5. We proposed a randomized algorithm that achieves ratio strictly greater than 0.5 for the case in which each online node has bounded degree.

▶ **Theorem 4** (EwOnBiMP with Bounded Online Degree)**.** *There exists an algorithm for edge-weighted online bipartite matching with free disposal such that on instances in which every online node has degree at most Δ, the performance ratio is $\frac{1}{2} + \Omega(\frac{1}{\Delta^2})$.*

1.2 Related Work

Unweighted ObMP. For the unweighted version, Dyer and Frieze [7] showed that the performance ratio is $0.5 + o(1)$ when the permutation of unordered pairs is chosen uniformly at random. In the mid-1990s, Aronson et al. [2] showed that the Modified Randomized Greedy (MRG) algorithm has ratio $0.5 + \epsilon$ (where $\epsilon = \frac{1}{400000}$). Goel and Tripathi [10] showed a hardness result of 0.7916 for any algorithm and 0.75 for adaptive vertex-iterative algorithms. In a recent SODA 2014 paper, Chan et al. [5] proved that Ranking algorithm has performance ratio at least 0.523166. We improve their analysis and performance ratio in this paper.

A version of the ranking algorithm was first proposed by Karp et al. [12] to solve the online bipartite matching problem (OnBiMP) with ratio $1 - \frac{1}{e}$. Subsequent works by Goel and Mehta [9], and Birnbaum and Mathieu [3] simplified the proof. Since the arrival order of online nodes is arbitrary, the same analysis carries over to obtain the same ratio for ObMP on bipartite graphs.

Since running Ranking on bipartite graphs for ObMP is equivalent to running the ranking algorithm for OnBiMP with *random arrival order*, the result of Karande et al. [11] implies that the ranking algorithm has a ratio at least 0.653 for the ObMP on bipartite graphs. Mahdian and Yan [13] improved the ratio to 0.696 using the technique of strongly factor-revealing LP. Karande et al. [11] also constructed a hard instance in which Ranking performs no better than 0.727.

Weighted Ranking. Aggarwal et al. [1] showed that the ranking algorithm can be applied to OnBiMP when the offline nodes have general weights. They proved that the performance ratio is $1 - \frac{1}{e}$. Devanur et al. [6] gave an alternative proof using randomized primal-dual analysis. We observe that their analysis can be applied to the NwObMP on bipartite graphs. Since their analysis assumes that the online nodes arrive in arbitrary order, by exchanging the roles of online and offline nodes for both partition of nodes, it can be shown that weighted Ranking achieves the same ratio of $1 - \frac{1}{e}$ on bipartite graphs.

EwOnBiMP with Free Disposal. Feldman et al. [8] proposed the free disposal feature for EwOnBiMP. They considered the setting in which each offline node v has capacity $n(v)$, and an online algorithm benefits from the $n(v)$ highest-weighted edges matched to v. They proposed an online algorithm with ratio $1 - \frac{1}{e_k}$, where $e_k = (1 + \frac{1}{k})^k$, and k is a lower bound on capacities. Thus, the proposed algorithm has performance ratio $\frac{1}{2}$ for the classic weighted version, when all capacities are 1.

1.3 Analyzing NwObMP via Linear Programming

A common technique [1, 11, 13, 10, 5] for analyzing Ranking algorithms is to define variables capturing the behavior of the algorithm in question, and derive structural properties that translate into constraints on the variables. A minimization LP with the performance ratio as the objective expressed in terms of the variables gives a lower bound on the ratio of the algorithm.

Let Ω be the sample space of configurations from which the algorithm derives its randomness. An *instance* $(\sigma, u) \in \Omega \times V$ is *good* if node u is matched when the algorithm is run with σ, and *bad* otherwise. We first describe the challenges encountered when previous techniques are applied to the node-weighted version of the problem on general graphs.

- *Why is the problem difficult on general graphs (as opposed to bipartite graphs)?* Bipartite graphs have the following nice property. Suppose in configuration σ, node u is unmatched, while its partner u^* in the optimal matching is matched to some node v. If the rank of u is promoted to form configuration σ', then u^* will be matched to some node v' such that the adjusted weight $w(\sigma', v') \geq w(\sigma, v)$ does not decrease. This naturally gives a way to relate the bad instance (σ, u) to the good instance (σ', v') [12, 11, 13, 1, 6], but unfortunately this property does not hold in general graphs. In fact, u^* might be unmatched in σ' as a result of u's promotion.
- *Why is the problem difficult when nodes have arbitrary weights (as opposed to uniform weight)?* In previous work [5] on the unweighted case, when u^* is matched in σ' in the above scenario, it is argued that the bad instance (σ, u) can be related to the good instance (σ', v), where v is matched in σ' to u^*. However, there is no guarantee that the adjusted weight $w(\sigma', v)$ of the good instance is at least $w(\sigma, u)$, which is needed as in [1, 6] to analyze the ratio for the weighted version.

To overcome the difficulties mentioned above, we have exploited the following structural properties of the Ranking algorithm. We analyze how the resulting matching would change if the rank of one node is changed (in Lemma 14), and give finer classification of good instances. In particular, the following notions are useful for relating bad instances to good instances in order to form LP constraints.

- **Graceful Instance.** A good instance (σ, u) is *graceful* if u is currently matched to a node v such that its optimal partner v^* does not exist or is also matched in σ.

- **Perpetual Instance.** If in a good instance (σ, u), node u has two unmatched neighbors, then (σ, u) is *perpetually good* in the sense that u will still be matched even when its rank is demoted to the bottom.

Breaking 0.5 Ratio for NwObMP. As in [1], we analyze the discrete sample space $\Omega_m := [m]^V$ (with the adjustment function $\varphi(t) := 1 - \frac{e^{17t}-1}{e^{17}-1}$, $\psi(i) := \varphi(\frac{i}{m})$ and adjusted weight $w(\sigma, u) := \psi(\sigma(u)) \cdot w_u$), and show that the performance ratio of weighted Ranking is at least the optimal value of some finite LP_m^ψ with m variables. Since LP_m^ψ does not depend on the size of G, computing the optimal value of LP_m^ψ for some large enough m is sufficient to prove a lower bound on the ratio of weighted Ranking. We show in our full version that a slightly better ratio can be analyzed using continuous LP for the limiting case as m tends to infinity.

We are aware of other adjustment functions that can achieve even slightly better ratios for the weighted Ranking, but we just present here a simple form that crosses the 0.5 barrier. Our result for the node-weighted case achieves the first non-trivial performance ratio that is strictly larger than 0.5.

Improved Ratio for Unweighted ObMP. We also apply our new combinatorial analysis to derive a new finite LP_n^U, that gives a lower bound on the performance ratio of unweighted Ranking running on graphs of size n. For the unweighted version of the problem, the limiting behavior of LP_n^U is analyzed when n tends to infinity and an improved lower bound on the performance ratio of unweighted Ranking is proved using a new class of continuous LP with jump discontinuity. The ideas for formulating the constraints are similar to the node-weighted case and we defer the proof to the full version.

1.4 Analyzing EwObMP and EwOnBiMP via Matching Coverage

Researchers have successfully applied the primal-dual LP framework to design approximation algorithms for matching problems [4, 6]. Consider the following standard maximum weight matching LP relaxation for an undirected graph $G = (V, E)$ with non-negative edge weights. Its dual is known as *vertex cover.*

$$\begin{aligned} \max \quad & w(x) := \sum_{\{u,v\}\in E} w_{uv} x_{uv} && (1) \\ \text{s.t} \quad & \sum_{u:\{u,v\}\in E} x_{uv} \le 1, && \forall v \in V \\ & x_{uv} \ge 0, && \forall \{u,v\} \in E \end{aligned} \qquad \begin{aligned} \min \quad & C(\alpha) := \sum_{u\in V} \alpha_u && (2) \\ \text{s.t} \quad & \alpha_v + \alpha_u \ge w_{uv}, && \forall \{u,v\} \in E \\ & \alpha_v \ge 0, && \forall v \in V \end{aligned}$$

An integral feasible primal solution x indicates whether an edge is selected and corresponds to some matching M, whose weight is denoted by $w(M) := w(x)$. When G is a bipartite graph between U and V, we use α_u for the variables for nodes in U and β_v for those corresponding to V.

Standard Primal-Dual Analysis. Typically, during the execution of an algorithm, both a primal and a dual solution are constructed. To analyze the approximation ratio, the value of the primal solution returned by the algorithm is compared with that of the dual solution. Since the primal is a maximization problem, any feasible dual provides an upper bound on the optimal primal value and can guarantee some approximation ratio. Hence, it is crucial in such a framework to establish the feasibility of the dual solution, for instance by either ensuring feasibility during construction, or scale the dual solution at the end by some appropriate factor. Dual feasibility requires that, for every edge in the graph, the sum of the dual values of its incident nodes is large enough.

New Framework. We observe that this strict requirement of dual feasibility is an artifact of the approximation analysis, and instead explore a new analysis method in which dual feasibility can be bypassed. Specifically, we use this new approach for different variations of edge-weighted maximum matching, and call it *matching coverage*. To emphasize that we do not achieve dual feasibility of any kind, we use a *vector* to mean an assignment of a non-negative value to each node.

▶ **Definition 5** (Matching Coverage)**.** Let M be a matching in graph G. A vector $\alpha \in \mathbb{R}^V$ is a *matching coverage* for matching M if α is non-negative, and the dual constraints of LP (2) corresponding to the edges of M are satisfied. In other words, for each $\{u, v\} \in M$, $\alpha_u + \alpha_v \geq w_{uv}$.

▶ Remark. Since any two distinct edges in a matching do not share any node, it follows that if a vector α is a matching coverage for a matching M, then $C(\alpha) \geq w(M)$.

General Framework of Matching Coverage. In our new analysis framework, the algorithm does not construct any dual solution (not even an infeasible one). This is a major departure from the conventional primal-dual framework in which some dual solution is usually constructed by an algorithm, whereas in our approach, the vector is used only for analysis. In the analysis, we imagine that as an algorithm ALG is executed, a vector α is constructed alongside with the knowledge of an optimal matching M^*. The idea is that the values in α are increased just enough to make sure that α is a matching coverage for M^*.

Why does this help the analysis? Since the vector α is a matching coverage for M^*, by Remark 1.4, we have $w(M^*) \leq C(\alpha)$. As α does not have to be feasible for all edge constraints, it is possible that the resulting value $C(\alpha)$ could be smaller than that of a feasible dual. Therefore, we can hope to get a smaller value of F when we compare $C(\alpha) \leq F \cdot w(M_{\mathsf{ALG}})$ with the weight of the matching M_{ALG} returned by ALG, thereby getting a larger performance ratio $w(M_{\mathsf{ALG}}) \geq \frac{1}{F} \cdot C(\alpha) \geq \frac{1}{F} \cdot w(M^*)$.

We use the framework of matching coverage to design and analyze algorithms for the following problems.

- EwObMP. In Section 4, we present an algorithm that achieves ratio strictly greater than 0.5 when the number of distinct edge weights is bounded. The full analysis is included in our full version.
- EwOnBiMP with Free Disposal. We present and analyze (in our full version) an algorithm that achieves ratio strictly greater than 0.5 when the online nodes have bounded degree. We show that without the free disposal assumption, no randomized algorithm can achieve any non-trivial constant guarantee on the ratio.

2 Defining Variables for Weighted Ranking on NwObMP

An adversary commits to a graph $G = (V, E)$ with $n = |V|$ nodes, where each node u has a non-negative weight w_u. We fix some maximum weight matching OPT in G. When the context is clear, we also use OPT to denote the set of nodes covered by the matching. Observe that in general OPT might be a proper subset of V. Let $w(\mathsf{OPT}) = \sum_{u \in \mathsf{OPT}} w_u$ be the total weight of OPT. For any $u \in V$, if u is matched in OPT, then we denote by u^* the partner of u in OPT, and we call u^* the *optimal partner* of u. If $u \notin \mathsf{OPT}$, then we say that u^* does not exist.

Weighted Ranking. As described in the introduction, the algorithm derives its randomness by sampling from $\Omega_m := [m]^V$ uniformly at random, where m is a sufficiently large integer and $[m] = \{1, 2, \ldots, m\}$. (We omit the subscript for Ω when the context is clear.) This is equivalent to picking $\sigma(u) \in [m]$ uniformly at random and independently for each $u \in V$. As in [1, 6], the algorithm fixes an *adjustment function* $\varphi : [0,1] \to [0,1]$ that is non-increasing. The function $\varphi(t) := 1 - e^{t-1}$ is used in [1, 6]. We shall consider other adjustment functions such that $\varphi(1) = 0$ also holds.

We denote $\psi(i) := \varphi(\frac{i}{m})$. Then, a permutation on V is induced by σ by sorting the nodes in non-increasing order of *adjusted weight* $w(\sigma, u) := \psi(\sigma(u)) \cdot w_u$, where ties are resolved deterministically (for instance by the identities of the nodes). This permutation on V induces a lexicographical order on the node pairs that is used for probing. We denote $(\sigma, u) > (\sigma, v)$ when node u comes before v in the permutation induced by σ, in which case u has higher priority than v.

We denote $\mathcal{U} := \Omega \times V$ as the set of *instances*. Let $M(\sigma)$ be the matching obtained when Ranking is run with configuration σ. If u is matched to some v after running Ranking with configuration σ, then we say that u is matched in σ and v is the (current) partner of u in σ. An instance (σ, u) is *good* if u is matched in σ, and otherwise *bad*. An event is a subset of instances.

Given $\sigma \in \Omega_m$, let σ_u^j be obtained by setting $\sigma_u^j(u) = j$ and $\sigma_u^j(v) = \sigma(v)$ for all $v \neq u$.

▶ **Definition 6** (Events)**.** For each $i \in [m]$, define the following:
- Rank-i good event: $Q_i := \{(\sigma, u) | \sigma(u) = i \textit{ and } u \textit{ is matched in } \sigma\}$
- Rank-i bad event: $R_i := \{(\sigma, u) | \sigma(u) = i, u \textit{ is not matched in } \sigma \text{ and } u \in \mathsf{OPT}\}$

Let $Q := \cup_{i \in [m]} Q_i$ and $R := \cup_{i \in [m]} R_i$.

Notice that Q_i and R_i are disjoint. While Q_i could involve nodes that are not in OPT, R_i only involves nodes in OPT; this idea also appears in [1] for dealing with the case when OPT is a proper subset of V. Define $x_i := \frac{\sum_{(\sigma,u) \in Q_i} w_u}{w(\mathsf{OPT}) \cdot m^{n-1}}$, which can be interpreted as the conditional expected contribution of the nodes given that they are at rank i. We next derive some properties of the x_i's.

- **Monotonicity.** For $i \geq 2$, we have $x_{i-1} \geq x_i \geq 0$, since if $(\sigma, u) \in Q_i$, then $(\sigma_u^{i-1}, u) \in Q_{i-1}$. However, $1 \geq x_1$ does not necessarily hold since there may exist $u \notin \mathsf{OPT}$ and $(\sigma, u) \in Q_1$.
- **Loss due to unmatched nodes.** Similar to x_i associated with Q_i, we consider an analogous quantity associated with R_i:

$$\begin{aligned} \overline{x}_i := & \frac{\sum_{(\sigma,u) \in R_i} w_u}{w(\mathsf{OPT}) \cdot m^{n-1}} = \frac{\sum_{(\sigma,u) \in Q_i \cup R_i} w_u - \sum_{(\sigma,u) \in Q_i} w_u}{w(\mathsf{OPT}) \cdot m^{n-1}} \\ \geq & \frac{w(\mathsf{OPT}) \cdot m^{n-1} - \sum_{(\sigma,u) \in Q_i} w_u}{w(\mathsf{OPT}) \cdot m^{n-1}} = 1 - x_i, \end{aligned} \tag{3}$$

where the inequality $\sum_{(\sigma,u) \in Q_i \cup R_i} w_u \geq w(\mathsf{OPT}) \cdot m^{n-1}$ could be strict because Q_i might involve nodes not in OPT.

- **Performance Ratio.** The performance ratio is the expected sum of weights of matched nodes divided by $w(\mathsf{OPT})$, which is given by $\frac{\sum_{(\sigma,u)\in Q} w_u}{w(\mathsf{OPT})\cdot m^n} = \frac{1}{m}\sum_{i=1}^m x_i$.

▶ **Definition 7** (Marginally Bad Event). For $i \in [m]$, we define rank-i marginally bad event as follows. Let $S_1 := R_1$; for $i \geq 2$, let $S_i := \{(\sigma, u) \in R_i | (\sigma_u^{i-1}, u) \in Q_{i-1}\}$.

Let $S := \cup_{i\in[m]} S_i$ and $\alpha_i := \frac{\sum_{(\sigma,u)\in S_i} w_u}{w(\mathsf{OPT})\cdot m^{n-1}}$ for all $i \in [m]$.

Observe that for an instance (σ, u) such that (σ_u^m, u) is bad, there exists a unique $j \in [m]$ such that $(\sigma_u^j, u) \in S_j$, and we say that j is the *marginal position* of (σ, u).

Relating x_i's and α_i's. From a marginally bad instance $(\sigma, u) \in S_i$, node u will be matched when its rank is promoted to $i - 1$. Hence, for $i \geq 2$, we immediately have

$$\alpha_i \leq \frac{\sum_{(\sigma,u)\in Q_{i-1}} w_u - \sum_{(\sigma,u)\in Q_i} w_u}{w(\mathsf{OPT}) \cdot m^{n-1}} = x_{i-1} - x_i. \tag{4}$$

Moreover, for $i \in [m]$, any bad instance $(\sigma, u) \in R_i$ has a unique marginal position $j \in [i]$ such that $(\sigma_u^j, u) \in S_j$; for each $(\sigma, u) \in S_j$ such that $j \leq i$, we also have $(\sigma_u^i, u) \in R_i$. Hence, there is a one-one correspondence between R_i and $\cup_{j=1}^i S_j$, and so we have:

$$\sum_{j=1}^{i} \alpha_j = \frac{\sum_{j=1}^i \sum_{(\sigma,u)\in S_j} w_u}{w(\mathsf{OPT}) \cdot m^{n-1}} = \frac{\sum_{(\sigma,u)\in R_i} w_u}{w(\mathsf{OPT}) \cdot m^{n-1}} = \overline{x}_i \geq 1 - x_i. \tag{5}$$

▶ Remark. Observe that when all nodes in V are covered by OPT, equality holds for both (4) and (5). In fact, Lemma 9 allow us to remove the α_i's from the LP constraints.

▶ **Fact 8** (Ranking is Greedy). *Suppose* Ranking *is run with configuration σ. If (σ, u) is bad, then each neighbor of u (in G) is matched in σ to some node v such that $(\sigma, v) > (\sigma, u)$.*

3 Analyzing NwObMP Using Graceful and Perpetual Instances

In this section we define some relations from (marginally) bad events to good events to formulate our LP constraints. We describe a general principle which is a weighted version of the argument used in [5].

As mentioned above, the following lemma is used to remove the α_i's from the LP constraints.

▶ **Lemma 9.** *Suppose that $\{b_i\}_{i=1}^{m+1}$ is non-negative and non-increasing such that $b_{m+1} = 0$, and $\{c_i\}_{i=1}^{m+1}$ is non-decreasing such that $c_1 = 0$. Then, we have*
(a) $\sum_{i=1}^m b_i\alpha_i \geq b_1 - \sum_{i=1}^m (b_i - b_{i+1})x_i$.
(b) $\sum_{i=1}^m b_i c_i \alpha_i \geq -\sum_{i=1}^m (b_i c_i - b_{i+1}c_{i+1})x_i$.

Proof. Statement (a) follows because

$$\sum_{i=1}^m b_i\alpha_i = \sum_{i=1}^m (b_i - b_{i+1}) \sum_{j=1}^i \alpha_j \geq \sum_{i=1}^m (b_i - b_{i+1})(1 - x_i) = b_1 - \sum_{i=1}^m (b_i - b_{i+1})x_i\,,$$

where the inequality comes from (5).

For statement (b), observing that $c_1 = 0$, we can assume that $\alpha_1 = x_0 - x_1$, where $x_0 = 1$. Let $C = \max_i c_i$, and define $d_i := C - c_i \geq 0$. Then, we have

$$\sum_{i=1}^{m} b_i c_i \alpha_i = \sum_{i=1}^{m} C b_i \alpha_i - \sum_{i=1}^{m} b_i d_i \alpha_i \geq C b_1 - C \sum_{i=1}^{m} (b_i - b_{i+1}) x_i - \sum_{i=1}^{m} b_i d_i (x_{i-1} - x_i)$$
$$= - \sum_{i=1}^{m} (b_i c_i - b_{i+1} c_{i+1}) x_i \,,$$

where in the inequality we apply statement (a) to the first term (which is still valid because $\alpha_1 \geq 1 - x_1$ holds), and apply $\alpha_1 = x_0 - x_1$ and (4) to the second term. ◀

Weighting Principle. Suppose f is a relation from subset A to subset B of instances, where $f(a)$ is the set of elements in B that are related to $a \in A$, and $f^{-1}(b)$ is the set of elements in A that are related to $b \in B$. Recall that each instance $a = (\sigma, u)$ has adjusted weight $w(a) = w(\sigma, u)$. Suppose further that for all $a \in A$, for all $b \in f(a)$, $w(a) \leq w(b)$. Then, by considering the bipartite graph H induced by f on $A \cup B$, and comparing the weights of end-points for each edge in H, it follows that $\sum_{a \in A} |f(a)| \cdot w(a) \leq \sum_{b \in B} |f^{-1}(b)| \cdot w(b)$.

We shall formulate constraints by considering relations between subsets of instances. The injectivity of a relation f is the minimum integer q such that for all $b \in B$, $|f^{-1}(b)| \leq q$. In this case, we have

$$\textstyle\sum_{a \in A} |f(a)| \cdot w(a) \leq q \sum_{b \in B} w(b). \tag{6}$$

3.1 Demoting Marginally Bad Instances

▶ **Lemma 10.** *We have:* $\frac{1}{m} \sum_{i=1}^{m} [2\psi(i) + (m-i)(\psi(i) - \psi(i+1))] x_i \geq \psi(1)$.

Proof. We define a relation f from the set S of marginally bad instances to the set Q of good instances. Observe that for a (marginally) bad instance (σ, u), u is unmatched in σ and its optimal partner u^* exists. If we further demote u by setting its rank to $j \geq \sigma(u)$, the resulting matching is unchanged. Therefore, by Fact 8, for each $j \geq \sigma(u)$, u^* is matched to the same v such that $w(\sigma, u) \leq w(\sigma, v) = w(\sigma_u^j, v)$. Hence, we can define $f(\sigma, u) := \{(\sigma_u^j, v) | u^* \textit{ is matched to } v \textit{ in } \sigma_u^j, j \geq \sigma(u)\} \subseteq Q$, where $|f(\sigma, u)| = m - \sigma(u) + 1$, and $w(\sigma, u) \leq w(\sigma', v)$ for all $(\sigma', v) \in f(\sigma, u)$.

We next check the injectivity of f. Suppose $(\rho, v) \in f(\sigma, u)$. Then, u^* is the current partner of v in ρ, and this uniquely determines u, which is unmatched in ρ. Hence, $\sigma = \rho_u^j$, where j is uniquely determined as the marginal position of (ρ, u). Therefore, the injectivity is 1.

Hence, our weighting principle (6) gives the following:

$$\sum_{i=1}^{m} \sum_{(\sigma,u) \in S_i} (m - i + 1)\psi(i) w_u = \sum_{a \in S} |f(a)| \cdot w(a) \leq \sum_{b \in Q} w(b) = \sum_{i=1}^{m} \sum_{(\rho,v) \in Q_i} \psi(i) w_v.$$

Dividing both sides by $w(\mathsf{OPT}) \cdot m^n$ gives $\frac{1}{m} \sum_{i=1}^{m} (m - i + 1)\psi(i)\alpha_i \leq \frac{1}{m} \sum_{i=1}^{m} \psi(i) x_i$.

Since we do not wish α_i's to appear in our constraints, we derive a lower bound for the LHS in terms of x_i's by applying Lemma 9 with $b_i := (m - i + 1)\psi(i)$, where $\psi(m+1)$ can be chosen to be any value. Rearranging gives the required inequality. ◀

3.2 Promoting Marginally Bad Instances

▶ **Lemma 11.** *We have:* $\frac{2}{m}\sum_{i=1}^{m}\psi(i)\cdot x_m + \frac{1}{m}\sum_{i=1}^{m}[5\psi(i) - i(\psi(i+1) - \psi(i))]\cdot x_i \geq \frac{3}{m}\sum_{i=1}^{m}\psi(i)$.

To use the weighting principle, we shall define relations from marginally bad instances S to the following subsets of special good instances.

▶ **Definition 12** (*If v is matched, would v^* still be matched?*)**.** For $i \in [m]$, let the graceful instances be $Y_i := \{(\sigma, u) \in Q_i | u$ is matched in σ to some v s.t. v^* does not exist or is also matched in $\sigma\}$. Let $y_i := \frac{\sum_{(\sigma,u)\in Y_i} w_u}{w(\mathsf{OPT})\cdot m^{n-1}}$ and $Y := \cup_{i\in[m]} Y_i$.

▶ **Definition 13** (*You will be matched even at the bottom*)**.** For $i \in [m]$, let the perpetual instances be $Z_i = \{(\sigma, u) \in Q_i | (\sigma_u^m, u) \in Q_m\}$. Let $z_i = \frac{\sum_{(\sigma,u)\in Z_i} w_u}{w(OPT)\cdot m^{n-1}}$ and $Z := \cup_{i\in[m]} Z_i$.

By definition, we know that $Y_i \subseteq Q_i$ and hence $x_i \geq y_i \geq 0$. Moreover, observing that there exists a bijection between Z_i and Q_m that maps each $(\sigma, u) \in Z_i$ to $(\sigma_u^m, u) \in Q_m$, we have $z_i = x_m$.

Suppose (σ, u) is a good instance that has marginal position j. We wish to compare the matchings produced by σ and σ_u^j. Sometimes it is more convenient to consider an unmatched node as being ignored. Specifically, given a configuration σ and a node u, running Ranking with σ_u means that we still use σ to generate the probing order, but any edge involving u is ignored. Observe that if (σ, u) has a marginal position j, then σ_u and σ_u^j will produce the same matching.

▶ **Lemma 14** (Ignoring One Node)**.** *Suppose u is covered by the matching $M(\sigma)$ produced by σ, and $M(\sigma_u)$ is the matching produced by using the same probing list, but any edge involving u is ignored. The symmetric difference $M(\sigma) \oplus M(\sigma_u)$ is an alternating path $P = (u = u_1, u_2, \ldots, u_p)$ such that for all $i \in [p-2]$, $(\sigma, u_i) > (\sigma, u_{i+2})$.*

Proof. We can view probing G with σ_u as using the same list L of unordered node pairs to probe another graph G_u, which is the same as G except that the node u is labelled *unavailable* and will not be matched in any case. After each round of probing, we compare what happens to the partially constructed matchings $M(\sigma)$ in G and $M(\sigma_u)$ in G_u. For the sake of this proof, "unavailable" and "matched" are the same *availability status*, while "unmatched" is a different availability status.

We apply induction on the number of rounds of probing. Observe that the following invariants hold initially. (i) There is exactly one node known as the *crucial* node (which is initially u) that has different availability in G and G_u. (ii) The symmetric difference $M(\sigma) \oplus M(\sigma_u)$ is an alternating path P connecting u to the current crucial node; initially, both $M(\sigma)$ and $M(\sigma_u)$ are empty, and path P is degenerate and contains only u. (iii) If the path $P = (u = u_1, u_2, \ldots, u_l)$ contains $l \geq 3$ nodes, then for all $i \in [l-2]$, then $(\sigma, u_i) > (\sigma, u_{i+2})$.

Consider the inductive step. Suppose currently the alternating path $M(\sigma) \oplus M(\sigma_u)$ contains l nodes, where u_l is crucial. Observe that the crucial node and $M(\sigma) \oplus M(\sigma_u)$ do not change in a round except for the case when the pair being probed is an edge in G (and G_u), involving the crucial node u_l with another currently unmatched node u_{l+1} in G, which is also unmatched in G_u (because the induction hypothesis states that all nodes but u_l have the same availability status in G and G_u).

Since u_l has different availability in G and G_u, but u_{l+1} is unmatched in both G and G_u, it follows that the edge $e := \{u_l, u_{l+1}\}$ is added to exactly one of $M(\sigma)$ and $M(\sigma_u)$.

Hence, the edge e is added to extend the alternating path $M(\sigma) \oplus M(\sigma_u)$, and the node u_{l+1} becomes crucial.

Next, it remains to show that if $l \geq 2$, then $(\sigma, u_{l-1}) > (\sigma, u_{l+1})$. Suppose we go back in time, and consider at the beginning of the round when the edge $\{u_{l-1}, u_l\}$ is about to be probed, and u_{l-1} is crucial. By the induction hypothesis, both u_l and u_{l+1} are unmatched in both G and G_u. It follows that $(\sigma, u_{l-1}) > (\sigma, u_{l+1})$, because otherwise the edge $\{u_{l-1}, u_l\}$ would have lower probing priority than $\{u_{l+1}, u_l\}$. This completes the inductive step. ◀

▶ **Lemma 15** (Two Unmatched Neighbors Implies Perpetual). *Suppose in configuration σ, node u is matched and has two unmatched neighbors in G. Then, $(\sigma, u) \in Z$ is perpetual.*

Proof. If we assume the opposite, then u will be unmatched in σ_u^m. Suppose x and y are two neighbors of u that are unmatched in σ. Then, by Lemma 14, the symmetric difference $M(\sigma) \oplus M(\sigma_u^m)$ is an alternating path starting from u, and hence at most one of x and y will remain unmatched in σ_u^m.

This implies that in σ_u^m, the unmatched node u will have at least one unmatched neighbor; this contradicts the fact that that Ranking will always produce a maximal matching. ◀

Next we derive inequalities involving the graceful instances. Combining the inequalities, we can obtain the crucial constraint involving only x_i's for achieving a ratio that is strictly larger than 0.5.

▶ **Lemma 16** (*You are unmatched because someone is not graceful.*). *We have the following inequality: $\frac{1}{m}\sum_{i=1}^{m} \psi(i) y_i \leq \frac{1}{m}\sum_{i=1}^{m} \psi(i)(2x_i - 1)$.*

Proof. We define a relation from the set R of bad instances to the set $Q \setminus Y$ of good instances that are not graceful.

Given any bad instance $(\sigma, u) \in R$, we know that u^* exists and is matched to some node v such that $w(\sigma, v) \geq w(\sigma, u)$, by Fact 8. Moreover, since v is matched to u^* such that u is unmatched, we know that $(\sigma, v) \in Q \setminus Y$ is good but not graceful. Hence, we define $f(\sigma, u) := \{(\sigma, v)\}$, where v is the current partner of u^*. Observe that each $(\sigma, v) \in Q \setminus Y$ can be related to a unique $(\sigma, u) \in R$, where u is the optimal partner of v's current partner in σ. Hence, the injectivity of f is 1.

Hence, the weighting principle (6) gives: $\sum_{(\sigma,u)\in R} w(\sigma, u) \leq \sum_{(\sigma,v)\in Q\setminus Y} w(\sigma, v)$. Dividing both sides by $w(\mathsf{OPT}) \cdot m^n$ gives: $\frac{1}{m}\sum_{i=1}^{m} \psi(i)\overline{x}_i \leq \frac{1}{m}\sum_{i=1}^{m} \psi(i)(x_i - y_i)$.

Finally, using $\overline{x}_i \geq 1 - x_i$ from (3) and rearranging gives the required inequality. ◀

▶ **Lemma 17** (*If you are marginal, someone else is either graceful or perpetual*). *We have the inequality: $\frac{1}{m}\sum_{i=1}^{m}(i-1)\psi(i)\alpha_i \leq \frac{1}{m}\sum_{i=1}^{m} \psi(i)(3y_i + 2z_i)$.*

Proof. As mentioned earlier, we shall define two relations f and g from marginally bad S to graceful Y and perpetual Z, respectively, such that the following properties hold.

1. For each $a \in S$, for each $b \in f(a) \cup g(a)$, $w(a) \leq w(b)$.
2. For each $a \in S$, $|f(a)| + |g(a)| = \sigma(u) - 1$.
3. The injectivity of f is at most 3 and the injectivity of g is at most 2.

Suppose we have f and g with these properties. Then, our weighting principle (6) gives:

$$\sum_{(\sigma,u)\in S} (\sigma(u) - 1) w(\sigma, u) \leq \sum_{(\rho,v)\in Y} 3w(\rho, v) + \sum_{(\rho,v)\in Z} 2w(\rho, v),$$

which by definition is equivalent to

$$\sum_{i=1}^{m}(i-1)\psi(i)\sum_{(\sigma,u)\in S_i} w_u \le \sum_{i=1}^{m}\psi(i)(3\sum_{(\rho,v)\in Y_i} w_u + 2\sum_{(\rho,v)\in Z_i} w_u).$$

Dividing both sides by $w(\mathsf{OPT})\cdot m^n$ gives the required inequality.

Next we show how f and g are constructed such that all required properties hold.

Given marginally bad $(\sigma,u)\in S$, we consider good instance $(\sigma',u)\in Q$, where $\sigma' = \sigma_u^j, j<\sigma(u)$ is obtained by "promoting" u's rank in σ. Note that by Fact 8, u^* must be matched in σ to some node v_0 such that $(\sigma,v_0) > (\sigma,u)$. Let the partner of u in σ' be p. The next claim is crucial for the construction of f and g.

▶ **Claim 18.** *If $w(\sigma',p) < w(\sigma,u)$, then u^* is matched in σ' to some node v such that $w(\sigma',v)\ge w(\sigma,v_0)\ge w(\sigma,u)$.*

Proof. By Lemma 14, we know that the symmetric difference $M(\sigma')\oplus M(\sigma)$ is an alternating path $(u=u_1, p=u_2, u_3, u_4\ldots)$ that starts with u. Moreover, we have $w(\sigma',u)\ge w(\sigma',u_3)\ge w(\sigma',u_5)\ge\ldots$ and $w(\sigma',p)\ge w(\sigma',u_4)\ge w(\sigma',u_6)\ge\ldots$. If u^* is not contained in the alternating path, then directly we have $v=v_0$ and hence the claim holds.

Assume that u^* is contained in the alternating path. Then, v_0 must also appear in the alternating path. Let $v_0=u_i$. Since $w(\sigma',v_0)=w(\sigma,v_0)\ge w(\sigma,u) > w(\sigma',p)$, we conclude that i must be odd. By Lemma 14, we know that u^* must be u_{i-1} since u_i is matched to u_{i-1} in σ. Moreover, we know that $u^*=u_{i-1}$ is matched to u_{i-2} in σ' such that $w(\sigma',u_{i-2})\ge w(\sigma',u_i)=w(\sigma,v_0)$. ◀

Next we include instances in Y into $f(\sigma,u)$ and instances in Z into $g(\sigma,u)$ on a case by case basis. Recall that for each $1\le j<\sigma(u)$, we consider $\sigma'=\sigma_u^j$; moreover, after promoting u to rank j, u is matched in σ' to p.

Case-1(a). u^* is matched in σ' and $w(\sigma',p)=w(\sigma,p)\ge w(\sigma,u)$. In this case, (σ',p) is graceful, because p is matched in σ' to u, whose optimal partner u^* is also matched. Hence, we include $(\sigma',p)\in Y$ in $f(\sigma,u)$.

Case-1(b). u^* is matched in σ' and $w(\sigma',p)=w(\sigma,p)<w(\sigma,u)$. By Claim 18, u^* is matched in σ' to some node v such that $w(\sigma',v)\ge w(\sigma,u)$. Observe that (σ',v) is graceful, and we include $(\sigma',v)\in Y$ in $f(\sigma,u)$.

Case-2(a). u^* is unmatched in σ', and p^* (if it exists) is also matched in σ'. Note that after promoting u, we have $w(\sigma',u)\ge w(\sigma,u)$. Moreover, (σ',u) is graceful, because the optimal partner p^* either does not exist or is matched in σ'. We include $(\sigma',u)\in Y$ in $f(\sigma,u)$.

Case-2(b). u^* is unmatched in σ', p^* exists and is the only unmatched neighbor of p in σ'. By Claim 18, since u^* is unmatched in σ', we have $w(\sigma,p)=w(\sigma',p)\ge w(\sigma,u)$; also, since p is matched in σ', $p\ne u^*$. Moreover, by Lemma 14, the symmetric difference $M(\sigma)\oplus M(\sigma')$ is an alternating path, and only two nodes (u and u^*) can have different matching status in σ and σ'.

Hence, in σ, p must remain matched and p^* must remain unmatched; this means that p has exactly two unmatched neighbors, namely u and p^*, in σ. By Lemma 15, we conclude that (σ,p) is perpetual, and include $(\sigma,p)\in Z$ in $g(\sigma,u)$.

Case-2(c). u^* is unmatched in σ', p^* exists and is not the only unmatched neighbor of p in σ'. Similar to Case-2(b), in this case, $w(\sigma', p) = w(\sigma, p) \geq w(\sigma, u)$ and p has two different unmatched neighbors in σ', so (σ', p) is perpetual by Lemma 15. We include $(\sigma', p) \in Z$ in $g(\sigma, u)$.

By construction, property 1 holds. Moreover, for each $1 \leq j < \sigma(u)$ and $\sigma' = \sigma_u^j$, exactly one of the above 5 cases happens. Hence, we also have property 2: $|f(\sigma, u)| + |g(\sigma, u)| = \sigma(u) - 1$. Next, we prove the injectivity.

Injectivity Analysis. Observe that in our construction, if $(\rho, v) \in f(\sigma, u) \cup g(\sigma, u)$, then $\sigma = \rho_u^t$, where t is the marginal position of (ρ, u). Hence, in the injectivity analysis, once (ρ, v) and u are identified, σ can be uniquely determined.

For relation f, suppose $(\rho, v) \in Y$ is included in some $f(\sigma, u)$ in the following cases.

- **Case-1(a).** Node u is uniquely identified as the current partner of v in ρ.
- **Case-1(b).** Node u is uniquely identified as the optimal partner of v's current partner.
- **Case-2(a).** Node u is the same as v.

Hence, each $(\rho, v) \in Y$ is related to at most 3 instances in S, which means that f has injectivity at most 3.

For relation g, suppose $(\rho, v) \in Z$ is included in some $g(\sigma, u)$ in the following cases.

- **Case-2(b).** By construction $\rho = \sigma$, and v has exactly two neighbors that are unmatched in ρ, one of which is v^*. Node u is uniquely identified as the other unmatched neighbor.
- **Case-2(c).** Node u is uniquely identified as the current partner of v in ρ.

Hence, each $(\rho, v) \in Z$ is related to at most 2 instances in S, which means that g has injectivity at most 2. This completes the proof of Lemma 17. ◀

We can now derive the main constraint of this subsection.

Proof of Lemma 11. We start from the inequality in Lemma 16. Observing that $z_i = x_m$, and using the upper bound for $\frac{1}{m}\sum_{i=1}^m \psi(i) y_i$ in Lemma 17, we have $\frac{1}{m}\sum_{i=1}^m (i-1)\psi(i)\alpha_i \leq \frac{1}{m}\sum_{i=1}^m \psi(i)(6x_i + 2x_m - 3)$.

We next use Lemma 9 by setting $b_i := \psi(i)$ and $c_i := i - 1$; observe that $c_1 = 0$, and we set $\psi(m+1) := 0$, which is consistent with $\psi(m) \geq 0 = \psi(m+1)$. Hence, we have the following lower bound for the LHS: $\frac{1}{m}\sum_{i=1}^m (i-1)\psi(i)\alpha_i \geq \frac{1}{m}\sum_{i=1}^m (\psi(i) + i(\psi(i+1) - \psi(i))) \cdot x_i$.

Rearranging gives the required inequality. ◀

3.3 Using LP to Bound Performance Ratio

Putting all achieved constraints on x_i's together, we obtain the following linear program LP_m^ψ, which is a lower bound on the performance ratio when weighted Ranking is run with weight adjustment function ψ and sample space $\Omega_m = [m]^V$.

$$
\begin{array}{llll}
\mathsf{LP}_m^\psi & \min & \frac{1}{m}\sum_{i=1}^m x_i & \\
& \text{s.t.} & x_i - x_{i+1} \geq 0, & i \in [m-1] \\
\multicolumn{3}{r}{\frac{2}{m}\sum_{i=1}^m \psi(i) \cdot x_m + \frac{1}{m}\sum_{i=1}^m [5\psi(i) - i(\psi(i+1) - \psi(i))] \cdot x_i \geq \frac{3}{m}\sum_{i=1}^m \psi(i)} & (7) \\
\multicolumn{3}{r}{\frac{1}{m}\sum_{i=1}^m [2\psi(i) + (m-i)(\psi(i) - \psi(i+1))] x_i \geq \psi(1)} & (8) \\
& & x_i \geq 0, & i \in [m].
\end{array}
$$

Achieving ratio strictly larger than 0.5. Observe that LP_m^ψ is independent of the size of G. Hence, to obtain a lower bound on the ratio, we can use an LP solver to solve LP_m^ψ for some large enough m and some appropriate non-negative non-increasing sequence $\{\psi(i)\}_{i=1}^m$. In particular, there exists a weighted Ranking algorithm with ratio strictly above 0.5.

▶ **Theorem 19.** *Using $m = 10000$ and $\psi(i) := 1 - \frac{e^{\frac{17i}{m}}-1}{e^{17}-1}$, the weighted* Ranking *algorithm has performance ratio at least the value given by LP_m^ψ: 0.501505.*

Although the function $\varphi(t) := 1 - e^{t-1}$ (that is used in [1, 6]) cannot give a ratio better 0.5 from our LP, it is still possible that the function could have good performance ratio. More experimental results and our source code can be downloaded at:
`http://i.cs.hku.hk/~algth/project/online_matching/weighted.html`.

Limiting case when m tends to infinity. Experiments show that LP_m^ψ is increasing in m. This suggests that a (slightly) better analysis may be achieved if Ranking samples σ from the continuous space $\Omega_\infty = [0,1]^V$, and uses adjusted weight $w(\sigma, u) := \varphi(\sigma(u)) \cdot w_u$ for each node u.

The variables x_i's are replaced by the function $z(t) := \frac{\sum_{u \in V} \Pr_\sigma[(\sigma,u) \text{ is good} | \sigma(u)=t] \cdot w_u}{w(\mathsf{OPT})}$. Our combinatorial counting argument can be replaced by measure analysis. For instance, $\Omega_\infty = [0,1]^V$ is equipped with the uniform n-dimensional measure, while $z(t)$ has measure of dimension $n-1$. Since we assume that $\psi(m+1) = 0$ in the finite analysis, this corresponds to $\varphi(1) = 0$ in continuous case.

Observe that it is possible to describe a continuous version of the weighting principle using measure theory to derive all the corresponding constraints involving z. However, the formal rigorous proof is out of the scope of this paper, and one can intuitively see that each constraint involving the x_i's translates naturally to a constraint involving z in the limiting case. Hence, the following continuous $\mathsf{LP}_\infty^\varphi$ gives a lower bound on the ratio when Ranking samples continuously, and we analyze it in our full version as a case study.

$$\begin{array}{lll}
\mathsf{LP}_\infty^\varphi & \min & \int_0^1 z(t)dt \\
& \text{s.t.} & z'(t) \le 0 \qquad \forall t \in [0,1] \\
\multicolumn{3}{l}{2\Phi \cdot z(1) + \int_0^1 [5\varphi(t) - t\varphi'(t)]\, z(t)dt \ge 3\Phi} \\
\multicolumn{3}{l}{\int_0^1 [2\varphi(t) - (1-t)\varphi'(t)]\, z(t)dt \ge \varphi(0)} \\
& & z(t) \ge 0 \qquad \forall t \in [0,1] \\
& & \Phi = \int_0^1 \varphi(t)dt.
\end{array}$$

▶ **Theorem 20** (Weighted Ranking with Continuous Sampling)**.** *Using continuous sample space Ω_∞ (with adjustment function $\varphi(t) := 1 - \frac{e^{17t}-1}{e^{17}-1}$), weighted* Ranking *has performance ratio at least* 0.501512.

4 Beating Ratio 0.5 for EwObMP

We consider EwObMP where the number of distinct weights is k. We give an algorithm whose performance ratio is $\frac{1}{2} + \xi_k$, where ξ_k only depends on k. As a subroutine, we use an algorithm $\mathcal{A}^{un}$ for the unweighted version of the problem with performance ratio $\frac{1}{2} + \xi_1$, where $\xi_1 > 0$. For instance, Theorem 2 implies that $\xi_1 \ge 0.0268$. When we run $\mathcal{A}^{un}$ on a subset $H \subseteq \binom{V}{2}$, $\mathcal{A}^{un}$ is first run to produce a random order L of node pairs. Only pairs in H are kept in L, while pairs not in H are removed. Then, the list L is used for probing as before. We partition the pairs in $\binom{V}{2}$ into batches $\{H_i\}_{i\geq 1}$, where the weights of pairs in each batch are similar. Then, starting from the batch with largest weights, we run $\mathcal{A}^{un}$ on each batch H_i to produce a list L_i, and return the concatenated list used for probing.

The following lemma, whose proof can be found in the full version, describes the properties of the intervals picked by the algorithm. Recall that $\mathcal{A}^{un}$ has performance ratio $\frac{1}{2}+\xi_1$ on unweighted ObMP. Given two real numbers $a \leq b$, we denote $\mathsf{dist}(a,b) := 1-\frac{a}{b}$.

▶ **Lemma 21** (Partitioning Weights into Batches). *Given a set W of k distinct weights, there exists an integer $r = O(k^2)$ and $\epsilon = \frac{\xi_1}{2}$ such that the algorithm can return disjoint intervals $\{I_i := [a_i,b_i]\}_{i\geq 1}$, whose union contains W, and for each $i \geq 1$, $b_{i+1} < a_i$, $\mathsf{dist}(a_i,b_i) \leq \epsilon^r$ and $\mathsf{dist}(b_{i+1},b_i) \geq \epsilon^{r-1}$.*

Algorithm 1 Algorithm for Edge-Weighted ObMP

```
1: $W \leftarrow \{w_e : e \in \binom{V}{2}\}$                    ▷ Set of weights of pairs in $\binom{V}{2}$.
2: $\{I_i := [a_i,b_i]\}_{i=1}^K \leftarrow$ Disjoint intervals as given in Lemma 21 to partition $W$, where $I_1$ is
   the interval with the largest weights.
3: for $i$ from 1 to $K$ do
4:     $H_i \leftarrow$ Pairs in $\binom{V}{2}$ with weights in $I_i$
5:     $L_i \leftarrow$ List produced by running unweighted $\mathcal{A}^{un}$ on $H_i$ using independent randomness
6: return concatenated list $L := L_1 \oplus L_2 \oplus \cdots \oplus L_K$
```

Assuming the knowledge of an optimal matching OPT, we construct a matching coverage $\alpha \in \mathbb{R}^V$ for OPT during an execution of the algorithm. For a matching M, we use $|M|$ to denote its cardinality and $w(M)$ to denote the sum of weights of its edges. We say an edge e in OPT is *destroyed* by a matching M if edge e is not in M but at least one end-point of e is matched in M. Moreover, two edges *intersect* if they share at least one end-point. We define the following edge sets for $i \geq 1$.

- ALG_i is the set of edges the algorithm includes in the matching when list L_i is probed.
- OPT_i is the set of edges in OPT that intersect with edges in ALG_i, but do not intersect with edges in ALG_j, for all $j < i$.
- $\mathsf{OPT}_i^H := \mathsf{OPT}_i \cap H_i$, each of which has weight in $[a_i,b_i]$.

The matching resulting from the probing list L returned by the algorithm is $\mathsf{ALG} := \cup_i \mathsf{ALG}_i$. Since ALG is a maximal matching in G, it follows that every edge in OPT appears in exactly one OPT_i.

Suppose V_i is the set of nodes matched in ALG_i. Let $C(\alpha_{V_i}) := \sum_{v\in V_i} \alpha_v$, where α_{V_i} is the vector α restricted to coordinates corresponding to V_i.

We defer the proof of the following lemma to our full version.

▶ **Lemma 22** (Local Performance Ratio). *Suppose the weights of H_i are in $[a_i,b_i]$, where $\eta := \mathsf{dist}(a_i,b_i)$; moreover, let $\lambda := \mathsf{dist}(b_{i+1},b_i)$. Then, $\mathbf{E}[w(\mathsf{ALG}_i)] \geq (1-\eta)\cdot(\frac{1}{2}+\frac{\xi_1\lambda}{1+2\xi_1})\cdot \mathbf{E}[C(\alpha_{V_i})]$.*

Finally, we are ready to prove the performance ratio of the algorithm.

Proof of Theorem 3. From Lemma 21, it follows that the parameters in Lemma 22 satisfy $\lambda \geq \epsilon^{r-1}$ and $\eta \leq \epsilon^r$, where $r = O(k^2)$. Observing that $\epsilon = \frac{\xi_1}{2} \leq \frac{1}{4}$, it follows that the local performance ratio is

$$\begin{aligned}\frac{\mathbf{E}[w(\mathsf{ALG}_i)]}{\mathbf{E}[C(\alpha_{V_i})]} &\geq (1-\eta)\left(\frac{1}{2}+\frac{\xi_1\lambda}{1+2\xi_1}\right) \geq (1-\epsilon^r)\left(\frac{1}{2}+\frac{\xi_1\epsilon^{r-1}}{1+2\xi_1}\right)\\ &=(1-\epsilon^r)\left(\frac{1}{2}+\frac{2\epsilon^r}{1+2\xi_1}\right)\\ &=\frac{1}{2}+\frac{\epsilon^r}{2+4\xi_1}\cdot(3-4(\epsilon+\epsilon^r)) \geq \frac{1}{2}+\frac{1}{2+4\xi_1}(\frac{\xi_1}{2})^r,\end{aligned}$$

where the last inequality follows because $r \geq 1$ and $\epsilon \leq \frac{1}{4}$.

Hence, we have $\mathbf{E}[\mathsf{ALG}] = \sum_i \mathbf{E}[\mathsf{ALG}_i] \geq (\frac{1}{2}+\frac{1}{2+4\xi_1}(\frac{\xi_1}{2})^r)\sum_i \mathbf{E}[C(\alpha_{V_i})]$. Finally, observe that $\sum_i \mathbf{E}[C(\alpha_{V_i})] = E[C(\alpha)] \geq w(\mathsf{OPT})$, as α is a matching coverage for OPT. Therefore, we conclude that the performance ratio for the whole algorithm is at least $\frac{1}{2}+\xi_k$, where $\xi_k = \frac{1}{2+4\xi_1}(\frac{\xi_1}{2})^r = \Omega(\xi_1)^{O(k^2)}$, as required. ◀

References

1 Gagan Aggarwal, Gagan Goel, Chinmay Karande, and Aranyak Mehta. Online vertex-weighted bipartite matching and single-bid budgeted allocations. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 1253–1264, 2011.

2 Jonathan Aronson, Martin Dyer, Alan Frieze, and Stephen Suen. Randomized greedy matching. ii. *Random Struct. Algorithms*, 6(1):55–73, January 1995. doi:10.1002/rsa.3240060107.

3 Benjamin Birnbaum and Claire Mathieu. On-line bipartite matching made simple. *SIGACT News*, 39(1):80–87, March 2008. doi:10.1145/1360443.1360462.

4 Niv Buchbinder, Kamal Jain, and Joseph Seffi Naor. Online primal-dual algorithms for maximizing ad-auctions revenue. In *ESA*, pages 253–264, 2007.

5 T.-H. Hubert Chan, Fei Chen, Xiaowei Wu, and Zhichao Zhao. Ranking on arbitrary graphs: Rematch via continuous lp with monotone and boundary condition constraints. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1112–1122, 2014. doi:10.1137/1.9781611973402.82.

6 Nikhil R. Devanur, Kamal Jain, and Robert D. Kleinberg. Randomized primal-dual analysis of ranking for online bipartite matching. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 101–107, 2013. doi:10.1137/1.9781611973105.7.

7 Martin E. Dyer and Alan M. Frieze. Randomized greedy matching. *Random Struct. Algorithms*, 2(1):29–46, 1991. doi:10.1002/rsa.3240020104.

8 Jon Feldman, Nitish Korula, Vahab S. Mirrokni, S. Muthukrishnan, and Martin Pál. Online ad assignment with free disposal. In *Internet and Network Economics, 5th International Workshop, WINE 2009, Rome, Italy, December 14-18, 2009. Proceedings.*, pages 374–385, 2009. doi:10.1007/978-3-642-10841-9_34.

9 Gagan Goel and Aranyak Mehta. Online budgeted matching in random input models with applications to adwords. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pages 982–991, 2008.

10 Gagan Goel and Pushkar Tripathi. Matching with our eyes closed. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 718–727, 2012. doi:10.1109/FOCS.2012.19.

11 Chinmay Karande, Aranyak Mehta, and Pushkar Tripathi. Online bipartite matching with unknown distributions. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 587–596, 2011. doi:10.1145/1993636.1993715.

12 Richard M. Karp, Umesh V. Vazirani, and Vijay V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 352–358, 1990. doi:10.1145/100216.100262.

13 Mohammad Mahdian and Qiqi Yan. Online bipartite matching with random arrivals: an approach based on strongly factor-revealing LPs. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 597–606, 2011. doi:10.1145/1993636.1993716.

14 Silvio Micali and Vijay V. Vazirani. An $O(\sqrt{V}E)$ algorithm for finding maximum matching in general graphs. In *FOCS'80*, pages 17–27. IEEE Computer Society, 1980. doi:10.1109/SFCS.1980.12.

15 Alvin E Roth, Tayfun Sönmez, and M Utku Ünver. Pairwise kidney exchange. *Journal of Economic theory*, 125(2):151–188, 2005.

Outer Common Tangents and Nesting of Convex Hulls in Linear Time and Constant Workspace

Mikkel Abrahamsen*[1] and Bartosz Walczak[2]

1 Department of Computer Science, University of Copenhagen, Copenhagen, Denmark
miab@di.ku.dk

2 Theoretical Computer Science Department, Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland
walczak@tcs.uj.edu.pl

Abstract

We describe the first algorithm to compute the outer common tangents of two disjoint simple polygons using linear time and only constant workspace. A tangent of a polygon is a line touching the polygon such that all of the polygon lies on the same side of the line. An outer common tangent of two polygons is a tangent of both polygons such that the polygons lie on the same side of the tangent. Each polygon is given as a read-only array of its corners in cyclic order. The algorithm detects if an outer common tangent does not exist, which is the case if and only if the convex hull of one of the polygons is contained in the convex hull of the other. Otherwise, two corners defining an outer common tangent are returned.

1998 ACM Subject Classification I.3.5 Computational Geometry and Object Modeling

Keywords and phrases simple polygon, common tangent, optimal algorithm, constant workspace

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.4

1 Introduction

A tangent of a polygon is a line touching the polygon such that all of the polygon lies on the same side of the line. An outer common tangent of two polygons is a tangent of both polygons such that the polygons lie on the same side of the tangent. Two disjoint polygons have exactly two outer common tangents unless their convex hulls are nested. If they are properly nested, there is no outer common tangent. In this paper, we study the problem of computing the outer common tangents of two disjoint simple polygons, each given as a read-only array of its corners in cyclic order. We give an algorithm computing the outer common tangents in linear time using only a constant number of variables each storing a boolean value or an index of a corner in the array. We are therefore working in the *constant workspace model* of computation.

The constant workspace model is a restricted version of the RAM model in which the input is read-only, the output is write-only, and only $O(\log n)$ additional bits of *workspace* (with both read and write access) are available, where n is the size of the input. Clearly, $\Omega(\log n)$ bits in the workspace are necessary to solve any interesting computational problem, because that many bits are required to store an index of or a pointer to an entry in the input. Since blocks of $\Theta(\log n)$ bits are considered to form *words* in the memory, algorithms in the constant workspace model use $O(1)$ words of memory, which explains the name of the model.

* Research partially supported by Mikkel Thorup's Advanced Grant from the Danish Council for Independent Research under the Sapere Aude research career programme.

The practical relevance of studying problems in the constant workspace model is increasing, as there are many current and emerging memory technologies where writing can be much more expensive than reading in terms of time and energy [8].

The constant workspace model was first studied explicitly for geometric problems by Asano et al. [4]. Recently, there has been growing interest in algorithms for geometric problems using constant or restricted workspace, see for instance [1, 3, 5, 6, 9, 11, 14].

The problem of computing common tangents of two polygons has received most attention in the case that the polygons are convex. For instance, computing the outer common tangents of disjoint convex polygons is used as a subroutine in the classical divide-and-conquer algorithm for the convex hull of a set of n points in the plane due to Preparata and Hong [17]. They give a naive linear-time algorithm for outer common tangents, as it suffices for an $O(n \log n)$-time convex hull algorithm. The problem is also considered in various dynamic convex hull algorithms [7, 12, 16]. Overmars and van Leeuwen [16] give an $O(\log n)$-time algorithm for computing an outer common tangent of two disjoint convex polygons when a separating line is known, where each polygon has at most n corners. Kirkpatrick and Snoeyink [13] give an $O(\log n)$-time algorithm for the same problem but without using a separating line. Guibas et al. [10] give a lower bound of $\Omega(\log^2 n)$ on the time required to compute an outer common tangent of two intersecting convex polygons, even if they are known to intersect in at most two points. They also describe an algorithm achieving that bound. Toussaint [18] considers the problem of computing separating common tangents of convex polygons and notes that the problem occurs in problems related to visibility, collision avoidance, range fitting, etc. He gives a linear-time algorithm. Guibas et al. [10] give an $O(\log n)$-time algorithm for the same problem. All the above-mentioned algorithms with sublinear running times make essential use of the convexity of the polygons. If the polygons are not convex, a linear-time algorithm can be used to compute the convex hulls before computing the tangents [15]. However, if the polygons are given in read-only memory, $\Omega(n)$ extra bits are required to store the convex hulls, so this approach does not work in the constant workspace model.

Abrahamsen [2] gives a linear-time constant-workspace algorithm to compute the outer common tangents of two simple polygons the convex hulls of which are disjoint. In this paper, we show that the same is possible as long as the polygons (but not necessarily their convex hulls) are disjoint. The algorithm is only slightly different from the one in [2], but its proof of correctness requires much more effort. In particular, the proof relies on an intricate continuous analysis of the algorithm. Before, it was not even clear whether to expect existence of a linear-time constant-workspace algorithm that does not require the convex hulls to be disjoint, because it happens quite often that a computational problem exhibits different behavior for disjoint polygons and for polygons that are not disjoint. For instance, as it has been mentioned above, the outer common tangents of two disjoint convex polygons can be computed in time $O(\log n)$, while doing the same for two convex polygons that intersect in two points requires time $\Omega(\log^2 n)$.

A separating common tangent of two polygons is a tangent of both polygons such that the polygons lie on the opposite sides of the tangent. Two disjoint polygons have exactly two separating common tangents provided that their convex hulls are disjoint. If they intersect properly, there is no separating common tangent. Abrahamsen [2] describes a linear-time constant-workspace algorithm that computes the separating common tangents of two simple polygons. In particular, it detects whether the convex hulls of two simple polygons are disjoint. Our current algorithm can decide whether the convex hulls two simple polygons are nested, which happens when it is unable to find an outer common tangent. To the best of our knowledge, this was not known to be possible in linear time and constant workspace prior to this work. Our algorithm and the algorithm from [2] together enable us to determine, for two disjoint simple polygons in general position, the full relation between their convex hulls (whether they are nested, overlapping, or disjoint) in linear time and constant workspace.

It remains open whether an outer common tangent of two polygons that are not disjoint can be found in linear time using constant workspace.

2 Terminology and Notation

For any two points a and b in the plane, the closed line segment with endpoints a and b is denoted by ab. When $a \neq b$, the straight line containing a and b that is infinite in both directions is denoted by $\mathcal{L}(a, b)$, and the ray starting at a and going through b is denoted by $\mathcal{R}(a, b)$. For three points a, b, and c, consider the line $\mathcal{L}(a, b)$ as oriented from a towards b, and define $\mathcal{T}(a, b, c)$ to be 1 if c lies to the left of $\mathcal{L}(a, b)$, 0 if a, b, c are collinear, and -1 if c lies to the right of $\mathcal{L}(a, b)$. Let $\mathrm{LHP}(a, b)$ denote the closed half-plane lying to the left of $\mathcal{L}(a, b)$ and $\mathrm{RHP}(a, b)$ denote the closed half-plane lying to the right of $\mathcal{L}(a, b)$.

A *simple polygon*, or just a *polygon*, with *corners* $x_0, \dots, x_{n-1}$ is a closed polygonal curve in the plane composed of n *edges* $x_0x_1, \dots, x_{n-2}x_{n-1}, x_{n-1}x_0$ such that the segments have no common points other than the common endpoints of pairs of consecutive edges. The region of the plane bounded by a polygon P (including P itself) is a *polygonal region*.

Assume for the rest of this paper that P_0 and P_1 are two disjoint simple polygons with n_0 and n_1 corners, respectively. (We allow one of P_0 and P_1 to be contained in the "interior region" of the other – in that case our algorithm will report that the convex hulls are nested and no outer common tangent exists.) Assume that P_k is defined by a read-only array of its corners $p_k[0], p_k[1], \dots, p_k[n_k - 1]$ for $k \in \{0, 1\}$. Assume further, without loss of generality, that the corners of P_0 are given in counterclockwise order and the corners of P_1 are given in clockwise order. (The orientation of a polygon can be easily tested in linear time using constant workspace, and the algorithm can choose to traverse the polygon forwards or backwards, accordingly.) Finally, assume that the corners are in general position in the sense that P_0 and P_1 have no corners in common and the combined set of corners $\{p_0[0], \dots, p_0[n_0 - 1], p_1[0], \dots, p_1[n_1 - 1]\}$ contains no triple of collinear points.

Indices of the corners of P_k are considered modulo n_k, so that $p_k[i]$ and $p_k[j]$ denote the same corner when $i \equiv j \pmod{n_k}$. For $a, b \in P_k$, the *chain* $P_k[a, b]$ is the portion of P_k from a to b in the order assigned to P_k (counterclockwise for P_0, clockwise for P_1). If i and j are indices of corners on P_k, we write $P_k[i, j]$ to denote $P_k[p_k[i], p_k[j]]$.

A *tangent* of P_k is a line ℓ such that ℓ and P_k are not disjoint and P_k is contained in one of the closed half-planes determined by ℓ. The line ℓ is a *common tangent* of P_0 and P_1 if it is a tangent of both P_0 and P_1. A common tangent is an *outer common tangent* if P_0 and P_1 are on the same side of the tangent, otherwise the common tangent is *separating*.

For a simple polygon P, let $\mathcal{H}(P)$ denote the convex hull of P. The following lemma asserts well-known properties of common tangents of polygons. See Figures 1–3.

▶ **Lemma 1.** *A line is a tangent of a polygon P if and only if it is a tangent of $\mathcal{H}(P)$. Under our general position assumptions, the following holds. If one of $\mathcal{H}(P_0)$ and $\mathcal{H}(P_1)$ is completely contained in the other, there are no outer common tangents of P_0 and P_1. Otherwise, there are two or more, and there are exactly two if P_0 and P_1 are disjoint. If $\mathcal{H}(P_0)$ and $\mathcal{H}(P_1)$ are not disjoint, there are no separating common tangents of P_0 and P_1. Otherwise, there are exactly two.*

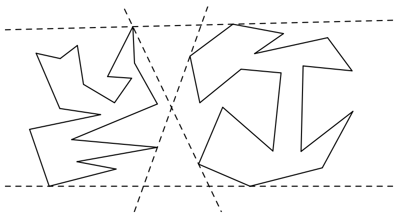


Figure 1 The convex hulls are disjoint – separating and outer common tangents exist.

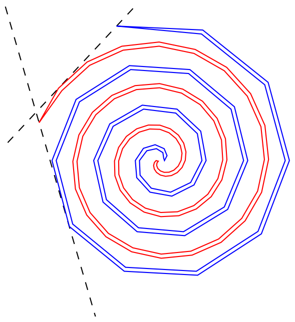


Figure 2 The convex hulls overlap – only outer common tangents exist.

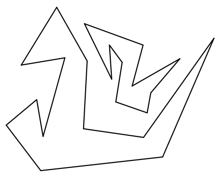


Figure 3 The convex hulls are nested – no common tangents exist.

Algorithm 1: `OuterCommonTangent`(P_0, P_1)

```
1  $s_0 \leftarrow 0$;  $v_0 \leftarrow 0$;  $b_0 \leftarrow$ false;  $s_1 \leftarrow 0$;  $v_1 \leftarrow 0$;  $b_1 \leftarrow$ false;  $u \leftarrow 0$
2  while $s_0 < 2n_0$ and $s_1 < 2n_1$ and $(v_0 < s_0 + n_0$ or $v_1 < s_1 + n_1)$
3      $v_u \leftarrow v_u + 1$
4      if $\mathcal{T}(p_0[s_0], p_1[s_1], p_u[v_u]) = 1$
5          if $p_{1-u}[s_{1-u}] \in \Delta(p_u[s_u], p_u[v_u - 1], p_u[v_u])$
6              $b_u \leftarrow$ true
7          if not $b_u$
8              $s_u \leftarrow v_u$;  $v_{1-u} \leftarrow s_{1-u}$;  $b_{1-u} \leftarrow$ false
9      $u \leftarrow 1 - u$
10 if $s_0 \geq 2n_0$ or $s_1 \geq 2n_1$ or $b_0$ or $b_1$
11     return nested
12 return $(s_0, s_1)$
```

3 Algorithm

Let the outer common tangents of P_0 and P_1 be defined by pairs of corners (ℓ_0, ℓ_1) and (r_0, r_1) so that $\ell_0, r_0 \in P_0$, $\ell_1, r_1 \in P_1$, and $P_0, P_1 \subset \mathrm{LHP}(\ell_0, \ell_1) \cap \mathrm{RHP}(r_0, r_1)$. Algorithm 1 returns a pair of indices (s_0, s_1) such that $(r_0, r_1) = (p_0[s_0], p_1[s_1])$ or, if the convex hulls of P_0 and P_1 are nested so that the tangents do not exist, the algorithm reports that by returning `nested`. Finding (ℓ_0, ℓ_1) requires running Algorithm 1 with the roles of P_0 and P_1 interchanged and with the orders of the corners of P_0 and P_1 reversed – each array reference $p_k[i]$ is translated to $p_{1-k}[-i]$ for $k \in \{0, 1\}$, and the returned result is (s_1, s_0) such that $(\ell_0, \ell_1) = (p_0[s_0], p_1[s_1])$.

The algorithm maintains a pair of indices (s_0, s_1) which determines the *tangent candidate* $\mathcal{L}(p_0[s_0], p_1[s_1])$. Starting from $(s_0, s_1) = (0, 0)$ and advancing the indices s_0, s_1 appropriately, the algorithm attempts to reach a situation that $(p_0[s_0], p_1[s_1]) = (r_0, r_1)$, that is, $P_0, P_1 \subset \mathrm{RHP}(p_0[s_0], p_1[s_1])$. At the start and after each update to (s_0, s_1), the algorithm traverses P_0 and P_1 in parallel with indices (v_0, v_1), starting from $(v_0, v_1) = (s_0, s_1)$ and advancing v_0 and v_1 alternately. The variable $u \in \{0, 1\}$ determines the polygon P_u in which we advance the traversal in a given iteration. If the test in line 4 happens to be positive, then the corner $p_u[v_u]$ lies on the "wrong side" of the tangent candidate, witnessing $P_u \not\subset \mathrm{RHP}(p_0[s_0], p_1[s_1])$.

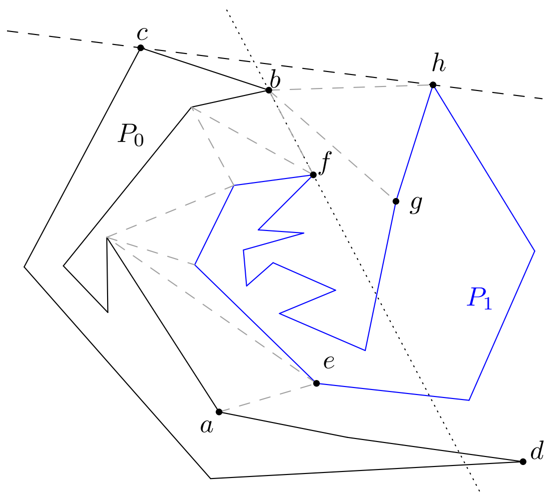


Figure 4 An example of how Algorithm 1 finds the outer common tangent $\mathcal{L}(c, h)$ of P_0 and P_1. The start points are $(p_0[0], p_1[0]) = (a, e)$. The gray dashed line segments are the segments $p_0[s_0]p_1[s_1]$ on the various tangent candidates. In the 11th iteration, an update makes $(p_0[s_0], p_1[s_1]) = (b, f)$, so the tangent candidate becomes the dotted line $\mathcal{L}(b, f)$. In the 19th iteration, $u = 0$ and $p_0[v_0] = d$, so b_0 is set to `true`. In the 28th iteration, $u = 1$ and $p_1[v_1] = g$, and therefore b_0 is cleared. In the 31st iteration, an update makes $(p_0[s_0], p_1[s_1]) = (c, h)$ and the outer common tangent has been found.

In that case, the algorithm updates the tangent candidate by setting $s_u \leftarrow v_u$ and reverts v_{1-u} back to s_{1-u} in line 8, unless a special boolean variable b_u is set, which we will comment on shortly. The reason for reverting v_{1-u} back to s_{1-u} in line 8 is that a corner of P_{1-u} which was on the correct side of the tangent candidate before the update to s_u can be on the wrong side of the tangent candidate after the update to s_u, and then it needs to be traversed again in order to be detected. The algorithm returns (s_0, s_1) in line 12 when it has traversed both polygons entirely with indices v_0 and v_1 after last updates to s_0 and s_1 without detecting any corner on the wrong side of the tangent candidate. That can happen only when $P_0, P_1 \subset \mathrm{RHP}(p_0[s_0], p_1[s_1])$. See Figure 4 for an example of how the algorithm proceeds.

In the test in line 5, $\Delta(a, b, c)$ denotes the filled triangle with corners a, b, c. If that test is positive, then $p_{1-u}[s_{1-u}]$ belongs to the convex hull of P_u, so $p_{1-u}[s_{1-u}] \neq r_{1-u}$. In that case, the boolean variable b_u is set, and then it prevents any updates to s_u in line 8 until it is cleared after a later update to s_{1-u} in line 8. It will be shown in the proof of Lemma 3 that such an update to s_{1-u} must occur if the convex hulls of P_0 and P_1 are not nested.

The main effort in proving correctness of Algorithm 1 lies in the following lemma, which is proved in Section 4.

▶ **Lemma 2.** *If the outer common tangents of P_0 and P_1 exist, then the loop in line 2 of Algorithm 1 ends with $s_0 < 2n_0$ and $s_1 < 2n_1$.*

The above implies that the algorithm ends up returning (s_0, s_1) in line 12 provided that $b_0 = b_1 =$ `false` when the loop in line 2 ends (this will be proved in Lemma 3).

To explain the role of the special variables b_0 and b_1, suppose temporarily that the conditions $s_0 < 2n_0$ and $s_1 < 2n_1$ are omitted from the test in line 2. If we were making the updates in line 8 regardless of the current values of b_0 and b_1, the algorithm could never end making updates to s_0 and s_1 even if the outer common tangents exist (see [2] for an example of such a behavior). In particular, Lemma 2 would no longer be true. On the other hand, if the convex hulls of P_0 and P_1 are nested, then one of the following happens:

- the algorithm never ends making updates to s_0 and s_1,
- one of b_0, b_1, say b_k, is `true` and the algorithm has traversed P_{1-k} entirely with the index v_{1-k} after last update to s_{1-k} without detecting any corner on the wrong side of the tangent candidate.

In both cases, taking the conditions $s_0 < 2n_0$ and $s_1 < 2n_1$ in line 2 back into account, the algorithm reports that the convex hulls of P_0 and P_1 are nested in line 11.

▶ **Lemma 3.** *If the outer common tangents of P_0 and P_1 exist, then the loop in line 2 of Algorithm 1 ends with $b_0 = b_1 =$ `false`.*

Proof. We prove a slightly stronger statement, namely, that at most one of b_0 and b_1 can be `true` at a time, and if one of b_0 and b_1 is `true`, then it will be cleared subsequently. Hence, the algorithm cannot terminate with $b_0 =$ `true` or $b_1 =$ `true`.

Consider an iteration i of the loop in line 2 which leads to changing the value of b_0 from `false` to `true` in line 6. By induction, we can assume that $b_1 =$ `false`. Since the test in line 5 is positive, the edge $P_0[v_0 - 1, v_0]$ intersects $\mathcal{L}(p_0[s_0], p_1[s_1])$ at a point x such that $p_1[s_1]$ lies on the segment $p_0[s_0]x$. Moreover, $P_0[p_0[s_0], x] \subset \mathrm{RHP}(p_0[s_0], p_1[s_1])$, otherwise b_0 would be set before. Let y be the first corner of P_1 after $p_1[s_1]$ such that $y \notin \mathrm{RHP}(p_0[s_0], p_1[s_1])$. Such a corner exists, otherwise P_1 would be contained in the convex hull of P_0. It follows that the test in line 4 will be positive in the first iteration j after i in which $u = 1$ and $p_1[v_1] = y$. The edge $P_1[v_1 - 1, v_1]$ intersects $\mathcal{L}(p_0[s_0], p_1[s_1])$ at a point on the segment $p_0[s_0]x$, and hence the test in line 5 is negative in iteration j. Therefore, b_0 is cleared and we again have $b_0 = b_1 =$ `false`. The same argument shows that b_1 will be cleared after being set. ◀

▶ **Theorem 4.** *Algorithm 1 is correct, runs in linear time, and uses constant workspace. Specifically, if the outer common tangents exist, then Algorithm 1 returns a pair of indices (s_0, s_1) such that $(r_0, r_1) = (p_0[s_0], p_1[s_1])$, that is, $P_0, P_1 \subset \mathrm{RHP}(p_0[s_0], p_1[s_1])$. Otherwise, the algorithm returns* `nested`.

Proof. First, suppose the algorithm returns (s_0, s_1) in line 12. Consider the final values of s_0, s_1, b_0 and b_1. Due to the test in line 10, we have $s_0 < 2n_0$, $s_1 < 2n_1$, and $b_0 = b_1 =$ `false`, so the loop in line 2 has ended because $v_0 \geq s_0 + n_0$ and $v_1 \geq s_1 + n_1$. After the last update to (s_0, s_1), the test in line 4 has been performed for every $v_0 \in \{s_0 + 1, \ldots, s_0 + n_0\}$ and every $v_1 \in \{s_1 + 1, \ldots, s_1 + n_1\}$ and was negative – otherwise a further update would have been performed in line 8, as $b_0 = b_1 =$ `false`. This shows that $P_0, P_1 \subset \mathrm{RHP}(p_0[s_0], p_1[s_1])$.

Now, suppose that the outer common tangents exist. By Lemma 2 and Lemma 3, the loop in line 2 ends with $s_0 < 2n_0$, $s_1 < 2n_1$, and $b_0 = b_1 =$ `false`. Hence (s_0, s_1) is returned in line 12. In view of the discussion above, this proves correctness of the algorithm.

It is clear that Algorithm 1 uses constant workspace. For the running time, note that if an update to (s_0, s_1) happens in iteration i, the sum $s_0 + s_1$ is increased by at least $\frac{i-j}{2}$, where j is the number of the previous iteration in which an update to (s_0, s_1) happened or $j = 0$ if there has been no update before. By induction, we see that there have been at most $2(s_0 + s_1)$ iterations until an update to (s_0, s_1). Suppose first that $s_0 < 2n_0$ and $s_1 < 2n_1$ when the loop in line 2 terminates. There have been at most $4(n_0 + n_1)$ iterations until the final update to (s_0, s_1). Thereafter, at most $2\max\{n_0, n_1\} \leq 2(n_0 + n_1)$ iterations follow until $v_0 \geq s_0 + n_0$ and $v_1 \geq s_1 + n_1$, when the loop in line 2 terminates. Hence, there are at most $6(n_0 + n_1)$ iterations in total. Now, suppose that $s_0 \geq 2n_0$ or $s_1 \geq 2n_1$ when the loop terminates. By the same argument, the second to last update to (s_0, s_1) happens after at most $4(n_0 + n_1)$ iterations, after which at most $2(n_0 + n_1)$ iterations follow until the last update to (s_0, s_1). The loop is terminated immediately after the last update. Hence, we get the same bound of $6(n_0+n_1)$ iterations. Clearly, each iteration takes constant time, so the total running time of the algorithm is linear. ◀

4 Proof of Lemma 2

For our analysis, it will be convenient to imagine the execution of Algorithm 1 in continuous time. By considering various discrete events happening during the continuous execution of the algorithm, we are able to prove the invariant stated in Lemma 2.

4.1 Additional Terminology and Notation

For $U \subseteq \mathbb{R}^2$, let $\mathcal{F}(U)$ denote the set of compact subsets of U. By an *interval*, we mean a bounded interval of real numbers. We allow an interval to be closed or open at each endpoint independently. We shall consider functions defined on an interval I with the following sets (or their subsets) as codomains: $\mathbb{R}$ with the standard metric, $\mathbb{R}^2$ with the Euclidean metric, and $\mathcal{F}(\mathbb{R}^2)$ with the Hausdorff metric, a set $\mathcal{S}$ of functions with the discrete metric, and the power set $2^{\mathcal{S}}$ of a set $\mathcal{S}$ of functions, again with the discrete metric. The only purpose of these metrics is to have a suitable notion of convergence. We think of the domain I as *time*. If f is a function with domain I and I' is a subinterval of I, then $f \restriction I'$ denotes the restriction of f to I'. For a function $f\colon I \to X$, where X is (a subset of) one of the codomains above, a point in time $t \in I$ is a *discontinuity* of f if f is not continuous at t. We write

- $f(\nearrow t^\star)$ to denote the limit of $f(t)$ as $t \to t^\star$ from below, where $t^\star \in I \setminus \{\inf I\}$,
- $f(\searrow t^\star)$ to denote the limit of $f(t)$ as $t \to t^\star$ from above, where $t^\star \in I \setminus \{\sup I\}$.

If the limits $f(\nearrow t^\star)$ exist for all $t^\star \in I \setminus \{\inf I\}$ and the limits $f(\searrow t^\star)$ exist for all $t^\star \in I \setminus \{\sup I\}$, then we say that f has *one-sided limits*. Each of the functions f that we consider has one-sided limits and finitely many discontinuities. Note that f has a discontinuity at a point in time $t \in I$ if and only if $f(\nearrow t) \neq f(t)$ or $f(\searrow t) \neq f(t)$. A function $f\colon I \to \mathcal{F}(U)$, where $U \subseteq \mathbb{R}^2$, is *monotonically decreasing* if $f(t) \supseteq f(t')$ for any $t, t' \in I$ such that $t < t'$.

▶ **Lemma 5.** *Let I be an interval and $f\colon I \to \mathcal{F}(U)$ be a function with one-sided limits and finitely many discontinuities, where $U \subseteq \mathbb{R}^2$. Suppose $f \restriction I'$ is monotonically decreasing for every subinterval $I' \subseteq I$ such that $f \restriction I'$ is continuous on I'. Furthermore, suppose that*

- *$f(\nearrow t) \supseteq f(t)$ for any $t \in I \setminus \{\inf I\}$ such that $f(\nearrow t) \neq f(t)$,*
- *$f(t) \supseteq f(\searrow t)$ for any $t \in I \setminus \{\sup I\}$ such that $f(t) \neq f(\searrow t)$.*

Then f is monotonically decreasing in the entire domain I.

Proof. Let $t_1 < \cdots < t_n$ be the discontinuities of f. Let $t, t' \in I$ and $t < t'$. If there is no i with $t \le t_i \le t'$, then $f \restriction [t, t']$ is continuous, so it follows from the assumption that $f(t) \supseteq f(t')$. Otherwise, let i be minimum and j be maximum such that $t \le t_i \le t_j \le t'$. If $t < t_i$, then the assumptions yield $f(t) \supseteq f(\nearrow t_i) \supseteq f(t_i)$, Similarly, the assumptions yield $f(t_k) \supseteq f(\searrow t_k) \supseteq f(\nearrow t_{k+1}) \supseteq f(t_{k+1})$ for $k \in \{i, \ldots, j-1\}$, and $f(t_j) \supseteq f(\searrow t_j) \supseteq f(t')$ if $t_j < t'$. Thus $f(t) \supseteq f(t')$. ◀

4.2 Continuous Interpretation of the Algorithm

Let m denote the number of iterations of the loop in line 2 performed by Algorithm 1. For $i \in \{0, \ldots, m\}$ and $k \in \{0, 1\}$, let $v_k(i)$ and $s_k(i)$ denote the values of v_k and s_k, respectively, after i iterations of the loop. In particular, $v_k(0) = s_k(0) = 0$. For $x \in \mathbb{R} \setminus \mathbb{Z}$, let $p_k[x]$ denote the interpolated point $(\lceil x \rceil - x)p_k[\lfloor x \rfloor] + (x - \lfloor x \rfloor)p_k[\lceil x \rceil]$ on the edge $P_k[\lfloor x \rfloor, \lceil x \rceil]$.

We extend the functions s_0 and s_1 to the real interval $[0, m]$ as follows. We imagine that the ith iteration of the loop in line 2 starts at time $i-1$ and ends at time i, and during that iteration v_u grows continuously from $v_u(i-1) = v_u(i) - 1$ to $v_u(i)$. Thus we define $v_u(t) = v_u(i) - i + t$ for $t \in (i-1, i)$. Suppose that the update in line 8 is to be performed in the ith iteration. If $s_u(i-1) = v_u(i-1)$, then all of the edge $P_u[v_u(i-1), v_u(i)]$ is in $\mathrm{LHP}(p_0[s_0(i-1)], p_1[s_1(i-1)])$. We therefore imagine that the update happens at time $i-1$ and then s_u grows continuously together with v_u up to $v_u(i)$; thus we define $s_u(t) = v_u(t)$ and $v_{1-u}(t) = s_{1-u}(i-1)$ for $t \in (i-1, i)$. If $s_u(i-1) < v_u(i-1)$, then the edge $P_u[v_u(i-1), v_u(i)]$ intersects the tangent candidate at a point $p_u[v_u(t^\star)]$, where $t^\star \in (i-1, i)$. We therefore imagine that the update in line 8 happens at time $t^\star$ and then s_u grows continuously together with v_u up to $v_u(i)$; thus we define

- $s_u(t) = s_u(i-1)$ and $v_{1-u}(t) = v_{1-u}(i-1)$ for $t \in (i-1, t^\star]$,
- $s_u(t) = v_u(t)$ and $v_{1-u}(t) = s_{1-u}(i-1)$ for $t \in (t^\star, i)$,

and we say that s_u *jumps* from $s_u(t^\star)$ to $v_u(t^\star) = s_u(\searrow t^\star)$ at time $t^\star$. Finally, in either case, we define $s_{1-u}(t) = s_{1-u}(i-1)$ for $t \in (i-1, i)$. The functions $s_0, s_1 \colon [0, m] \to \mathbb{R}$ thus defined are nondecreasing, have one-sided limits and finitely many discontinuities, and are left-continuous, that is, $s_0(\nearrow t) = s_0(t)$ and $s_1(\nearrow t) = s_1(t)$ for every $t \in (0, m]$. We have also defined functions $v_0, v_1 \colon [0, m] \to \mathbb{R}$, but we are not going to use them any more.

▶ **Observation 6.** *At any point in time during the execution of the continuous version of Algorithm 1, at most one of s_0, s_1 is changing. The tangent candidate $\mathcal{L}(p_0[s_0], p_1[s_1])$ either is not moving, or is turning continuously counterclockwise around $p_0[s_0]$ (when s_1 is changing), or is turning continuously clockwise around $p_1[s_1]$ (when s_0 is changing).*

The following is trivial if $s_k(t) = s_k(\searrow t)$ and otherwise is a direct consequence of the test in line 5 and of the fact that the update in line 8 is only performed when $b_u = \texttt{false}$.

▶ **Observation 7.** *If $t \in [0, m)$ and $k \in \{0, 1\}$, then $p_k[s_k(\searrow t)] \in \mathcal{R}(p_{1-k}[s_{1-k}(t)], p_k[s_k(t)])$ and $P_k[s_k(t), s_k(\searrow t)] \subset \mathrm{RHP}(p_0[s_0(t)], p_1[s_1(t)])$.*

4.3 Auxiliary Structure on the Polygons

In this subsection, we introduce some auxiliary concepts used in the proof of Lemma 2. They are defined in terms of the polygons P_0, P_1 only and are independent of the algorithm.

Assume for this entire subsection that the convex hulls of P_0 and P_1 are not nested. Thus there are two outer common tangents – let them be given by points $\ell_0, r_0 \in P_0$ and $\ell_1, r_1 \in P_1$ such that $P_0, P_1 \subset \mathrm{LHP}(\ell_0, \ell_1) \cap \mathrm{RHP}(r_0, r_1)$. Let $L = \ell_0\ell_1$ and $R = r_0r_1$. Let E be the polygonal region bounded by the chains $P_0[\ell_0, r_0]$, $P_1[\ell_1, r_1]$ and by the segments L, R. Since P_0 is oriented counterclockwise and P_1 clockwise, the interiors of P_0 and P_1 lie outside E.

▶ **Lemma 8.** *Every segment xy such that $xy \cap P_0 = \{x\}$ and $xy \cap P_1 = \{y\}$ is contained in E.*

Proof. The set $E \smallsetminus (P_0[\ell_0, r_0] \cup P_1[\ell_1, r_1])$ separates P_0 and P_1 in $\mathrm{LHP}(\ell_0, \ell_1) \cap \mathrm{RHP}(r_0, r_1)$, so it contains a point z in common with the segment xy. If $z \in L$ or $z \in R$, then $xy = \ell_0\ell_1$ or $xy = r_0r_1$, respectively, so xy lies in E. So suppose z is in the interior of E. The segment zx cannot cross the boundary of E at any point other than x, and zy at any point other than y. This shows that xy lies in E. ◀

See Figure 5. Let $q_0 \in P_0$ and $q_1 \in P_1$ be fixed points such that at least one of q_0, q_1 is a corner of the respective polygon P_0 or P_1. Let $S = q_0q_1$. We consider the segment S as oriented from q_0 to q_1, so that we can speak of the *left side* of S, $\mathrm{LHP}(q_0, q_1)$, and the

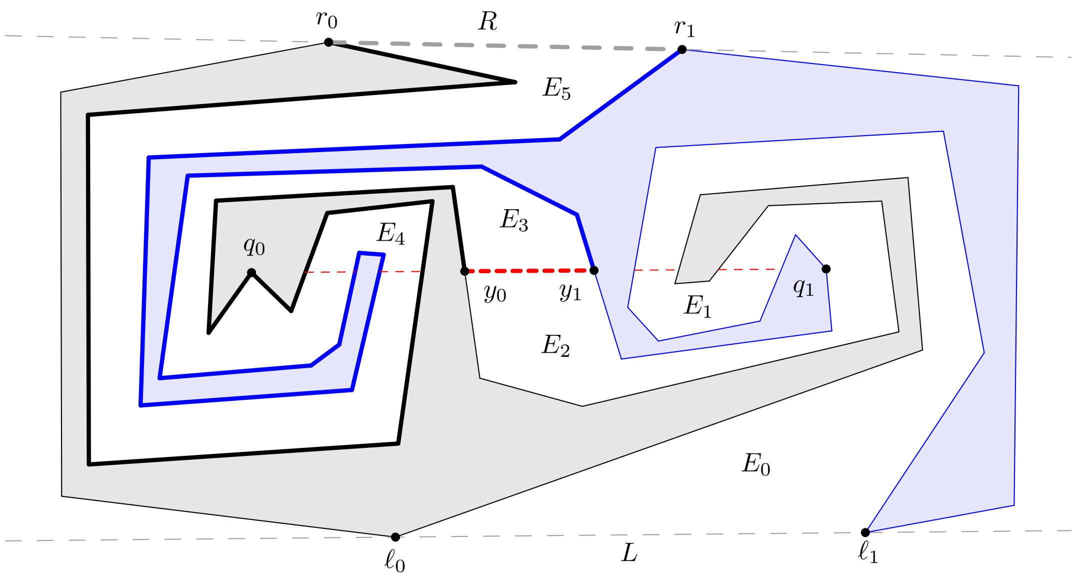


Figure 5 The doors are the five dashed segments on $S = q_0q_1$: D_4, D_5, D_3, D_1, D_2 in the order from q_0 to q_1. The weights of the doors are 2, 1, 1, -1, 0, respectively. $D_3 = y_0y_1$ is the primary door. The boundary of the primary region $E' = E_3 \cup E_4 \cup E_5$ is drawn with thick lines.

right side of S, $\mathrm{RHP}(q_0, q_1)$. A *door* is a subsegment xy of S such that $xy \cap P_k = \{x\}$ and $xy \cap P_{1-k} = \{y\}$ for some $k \in \{0, 1\}$. By Lemma 8, every door is contained in E. A *fence* is a subsegment xy of S such that $xy \subset E$, $xy \cap P_k = \{x, y\}$, and $xy \cap P_{1-k} = \emptyset$ for some $k \in \{0, 1\}$. Exceptionally, when S contains an edge xy of P_k, we call the whole edge xy a fence. Since at least one of q_0, q_1 is a corner, the latter is possible only when $x = q_k$ or $y = q_k$. Let $\mathcal{D}$ be the set of all doors defined by the fixed points q_0 and q_1. Figure 5 also illustrates the following lemma.

▶ **Lemma 9.** *The doors in $\mathcal{D}$ can be ordered as $D_1, \ldots, D_d$ so that if $D_i \cap P_0 = \{x_i\}$ and $D_i \cap P_1 = \{y_i\}$ for $i \in \{1, \ldots, d\}$, then*

- *the order of points along $P_0[\ell_0, r_0]$ is $\ell_0, x_1, \ldots, x_d, r_0$ (with possible coincidences),*
- *the order of points along $P_1[\ell_1, r_1]$ is $\ell_1, y_1, \ldots, y_d, r_1$ (with possible coincidences).*

The doors partition E into polygonal regions $E_0, \ldots, E_d$ such that

- *E_0 is bounded by L, $P_0[\ell_0, x_1]$, D_1 and $P_1[\ell_1, y_1]$ (it is degenerate when $D_1 = L$),*
- *E_i is bounded by D_i, $P_0[x_i, x_{i+1}]$, D_{i+1} and $P_1[y_i, y_{i+1}]$, for $i \in \{1, \ldots, d-1\}$,*
- *E_d is bounded by D_d, $P_0[x_d, r_0]$, R and $P_1[y_d, r_1]$ (it is degenerate when $D_d = R$).*

Proof. Suppose there are doors $xy, x'y' \in \mathcal{D}$ such that x is strictly before x' on $P_0[\ell_0, r_0]$ while y' is strictly before y on $P_1[\ell_1, r_1]$. It follows that the clockwise order of the four points along the boundary of E is x, x', y, y' and no two of these points coincide. By Lemma 8, both xy and $x'y'$ lie in E, so they must cross at a point different from their endpoints, which is a contradiction. This shows that the order of endpoints of the doors along $P_0[\ell_0, r_0]$ agrees with that along $P_1[\ell_1, r_1]$, which proves the first statement. The second statement is a straightforward corollary to the first. ◀

From now on, we use $D_1, \ldots, D_d$ to denote the doors in their order according to Lemma 9, and we use $E_0, \ldots, E_d$ to denote the regions defined in Lemma 9.

Recall that we consider S as a segment oriented from q_0 to q_1. Every door inherits that orientation, so that we can speak of the left side and the right side of the door. Taking into account that the regions E_{i-1} and E_i lie on opposite sides of D_i, we classify each door D_i as

- a *right-door* if E_{i-1} lies to the right and E_i lies to the left of D_i (in particular, if $D_i = L$),
- a *left-door* if E_{i-1} lies to the left and E_i lies to the right of D_i (in particular, if $D_i = R$).

▶ **Lemma 10.** *Consider a chain $P_k[a,b]$, where $k \in \{0,1\}$. If $P_k[a,b] \cap S = \{a,b\}$ and $P_k[a,b] \subset \mathrm{RHP}(q_0,q_1)$, then all doors contained in the segment ab occur in pairs of a left-door followed by a right-door, consecutive in the order on $\mathcal{D}$.*

Proof. Consider the polygonal region F bounded by the chain $P_k[a,b]$ and by the segment ab. It follows that $F \subset \mathrm{RHP}(q_0,q_1)$. Each of the regions $E_0,\dots,E_d$ lies either inside or outside F, where E_0 and E_d lie outside F. Each region E_i lying inside F connects the door D_i, which is therefore a left-door, and the door D_{i+1}, which is therefore a right-door. ◀

So far we were considering q_0 and q_1 as fixed points. Now, we allow them to change in time. Specifically, let I be a real interval that can be open or closed at each endpoint independently, and consider q_0 and q_1 as continuous functions $q_0\colon I \to P_0$ and $q_1\colon I \to P_1$. This way S becomes a continuous function $S\colon I \to \mathcal{F}(\mathbb{R}^2)$. Furthermore, suppose at least one of $q_0(t)$, $q_1(t)$ is a corner of the respective polygon for every $t \in I$, so that $S(t)$ can contain at most one other corner (by the general position assumption). Let $X(t)$ denote the set of intersection points of $S(t)$ with $P_0 \cup P_1$. In the exceptional case that $S(t)$ contains an edge of P_0 or P_1, we only include the endpoints of the edge in $X(t)$. The points in $X(t)$ are changing continuously except that an intersection point appears or disappears at a point in time $t \in I$ when $S(t)$ sweeps over a corner whose both incident edges lie on the same side of $S(t)$. Note that since the corners of P_0 and P_1 are assumed to be in general position and one of q_0 and q_1 is a corner, at most one point can appear in or disappear from $X(t)$ at any point in time. The doors are changing continuously except when one of the following *door events* happens as a point appears in or disappears from $X(t)$:

1. a fence splits into two doors,
2. two doors merge into a fence,
3. a door splits into a smaller door and a fence,
4. a door and a fence merge into a larger door.

Specifically, every door D can be represented as a continuous function $D\colon I_D \to \mathcal{F}(\mathbb{R}^2)$, where I_D is a subinterval of I (open or closed at each endpoint independently) such that

1. if $t = \inf I_D \in I_D$, then an endpoint of $D(t)$ is in $X(t)$ but not in $X(\nearrow t)$,
2. if $t = \sup I_D \in I_D$, then an endpoint of $D(t)$ is in $X(t)$ but not in $X(\searrow t)$,
3. if $t = \sup I_D \notin I_D$, then an interior point of $D(\nearrow t)$ is in $X(t)$ but not in $X(\nearrow t)$,
4. if $t = \inf I_D \notin I_D$, then an interior point of $D(\searrow t)$ is in $X(t)$ but not in $X(\searrow t)$.

At any point in time $t \in I$, the set of doors $\mathcal{D}(t)$ consists of the doors D such that $t \in I_D$ ordered according to Lemma 9. The following observation, a straightforward consequence of Lemma 9, summarizes how $\mathcal{D}(t)$ and the order on $\mathcal{D}(t)$ are changing in time.

▶ **Observation 11.** *The set $\mathcal{D}(t)$ and the order on $\mathcal{D}(t)$ are constant in time intervals where no door event happens. A door event at time t makes the following change to $\mathcal{D}(t)$:*

1. *if a fence splits into two doors D and D', then D and D' are added to $\mathcal{D}(\nearrow t)$ as consecutive doors to form $\mathcal{D}(t)$,*
2. *if two doors D and D' merge into a fence, then D and D' are consecutive in $\mathcal{D}(t)$ and they are removed from $\mathcal{D}(t)$ to form $\mathcal{D}(\searrow t)$,*

3. *if a door D splits into a smaller door D' and a fence, then D is replaced by D' in $\mathcal{D}(\nearrow t)$ to form $\mathcal{D}(t)$,*
4. *if a door D and a fence merge into a larger door D', then D is replaced by D' in $\mathcal{D}(t)$ to form $\mathcal{D}(\searrow t)$.*

In case of door events 1 and 2, the two doors D and D' are, in their order in $\mathcal{D}(t)$,

- *a right-door followed by a left-door if the edges incident to w lie to the right of $S(t)$,*
- *a left-door followed by a right-door if the edges incident to w lie to the left of $S(t)$,*

where w denotes the corner that triggers the event (i.e., the corner that appears in or disappears from $X(t)$ at time t). In case of door events 3 and 4, the door D' keeps the left/right-door status of D. The left/right-door status of every door D remains constant over the entire time interval I_D.

Now, consider q_0 and q_1 again as fixed points. Recall that $D_1, \ldots, D_d$ denote the doors in their order according to Lemma 9. We define the *weight* $W(D_i)$ of every door D_i by induction, as follows:

$$W(D_1) = \begin{cases} 1 & \text{if } D_1 \text{ is a right-door,} \\ -1 & \text{if } D_1 \text{ is a left-door,} \end{cases} \qquad W(D_i) = \begin{cases} W(D_{i-1}) + 1 & \text{if } D_i \text{ is a right-door,} \\ W(D_{i-1}) - 1 & \text{if } D_i \text{ is a left-door,} \end{cases}$$

for $i \in \{2, \ldots, d\}$. See Figure 5. The following is a direct consequence of Observation 11.

▶ **Observation 12.** *When $q_0 \colon I \to P_0$, $q_1 \colon I \to P_1$ are continuous functions, every door $D \colon I_D \to \mathcal{F}(\mathbb{R}^2)$ maintains constant weight over the entire time interval I_D. Furthermore, the function $W^\star \colon I \to \mathbb{Z}$ defined so that $W^\star(t)$ is the weight of the last door in the order on $\mathcal{D}(t)$ is constant over the entire time interval I.*

▶ **Lemma 13.** *For any fixed points q_0, q_1, there is at least one door with weight 1.*

Proof. The statement is obvious if $q_0 = \ell_0$ and $q_1 = \ell_1$, because in that case there is just one door L, which is a right-door by definition, so it has weight 1. To prove the lemma in general, let $I = [0, 1]$ and (abusing notation) consider arbitrary continuous functions $q_0 \colon I \to P_0$ and $q_1 \colon I \to P_1$ such that $q_0(0) = \ell_0$, $q_1(0) = \ell_1$, and $q_0(1)$, $q_1(1)$ are the points q_0, q_1 fixed in the statement of the lemma. By Observation 12, the function $W^\star \colon I \to \mathbb{Z}$ is constant over I, so $W^\star(1) = W^\star(0) = 1$ as observed above. This shows that the last door in the order on $\mathcal{D}(1)$ has weight 1. ◀

For any fixed points q_0, q_1, let the *primary door* D' be the first door with weight 1 in the order on $\mathcal{D}$. Such a door always exists due to Lemma 13.

▶ **Observation 14.** *The primary door D' is a right-door and is not preceded by a left-door in the order on $\mathcal{D}$.*

Let y_0 and y_1 denote the endpoints of D' so that $y_0 \in P_0$ and $y_1 \in P_1$. Let $Y_0 = P_0[y_0, r_0]$ and $Y_1 = P_1[y_1, r_1]$. Finally, let the *primary region* E' be defined as the polygonal region determined by D', Y_0, R and Y_1. See Figure 5.

▶ **Observation 15.** *If $D' = D_i$, then E' is the union of $E_i, \ldots, E_d$. In particular, E' contains the doors $D_{i+1}, \ldots, D_d$. By Observation 14, the region E' meets D' from the left.*

4.4 Back to the Algorithm

We recall the functions $s_0, s_1\colon [0,m] \to \mathbb{R}$ describing the execution of Algorithm 1 as explained in Section 4.2, and we define functions $q_0\colon [0,m] \to P_0$ and $q_1\colon [0,m] \to P_1$ as follows:

$$q_0(t) = p_0[s_0(t)], \quad q_1(t) = p_1[s_1(t)] \quad \text{for } t \in [0,m].$$

They have the property that at least one of $q_0(t)$, $q_1(t)$ is a corner at any point in time $t \in [0,m]$. Some other objects that have been defined in Section 4.3 based on fixed points q_0, q_1 now become functions of time $t \in [0,m]$: the segment S, the primary door D', the points y_0, y_1, the chains Y_0, Y_1, and the primary region E'.

The functions q_0 and q_1 have finitely many discontinuities – the points of time $t \in [0,m)$ when the respective s_k jumps from $s_k(t)$ to $s_k(\searrow t)$. It is also clear that they have one-sided limits, since the functions s_0 and s_1 are bounded and piecewise monotone. It follows that the functions $D'\colon [0,m] \to \mathcal{F}(\mathbb{R}^2)$, $y_0\colon [0,m] \to P_0$, $y_1\colon [0,m] \to P_1$, $Y_0\colon [0,m] \to \mathcal{F}(P_0)$, $Y_1\colon [0,m] \to \mathcal{F}(P_1)$, and $E'\colon [0,m] \to \mathcal{F}(\mathbb{R}^2)$ also have one-sided limits and finitely many discontinuities, which arise from discontinuities of q_0, q_1 and from door events in between.

The following lemma is the heart of the proof of correctness of the algorithm. Informally speaking, it asserts that the primary region E' can only shrink in time, since the primary door D' always sweeps continuously into or jumps into E'.

▶ **Lemma 16.** *The functions Y_0 and Y_1 are monotonically decreasing.*

Proof. First, we let I be an arbitrary subinterval of $[0,m]$ in which q_0 and q_1 are continuous, and we prove the lemma for functions restricted to I: $y_0 \upharpoonright I$, $y_1 \upharpoonright I$, $Y_0 \upharpoonright I$ and $Y_1 \upharpoonright I$. Following the convention from Section 4.3, we consider doors as continuous functions $D\colon I_D \to \mathcal{F}(\mathbb{R}^2)$ with $I_D \subseteq I$ and, for $t \in I$, we let $\mathcal{D}(t)$ denote the set of doors D such that $t \in I_D$. Accordingly, we redefine $D'(t)$ to denote the function $D\colon I_D \to \mathcal{F}(\mathbb{R}^2)$ that is chosen as the primary door at time $t \in I$.

By Observation 12, every door $D\colon I_D \to \mathcal{F}(\mathbb{R}^2)$ maintains constant weight over the entire time interval I_D. By Observation 11, the only possible changes to $\mathcal{D}$ and to the order on $\mathcal{D}$ over time interval I are that doors are being added to or removed from $\mathcal{D}$. Therefore, any change to the choice of the primary door can only occur at a point in time $t \in I$ when a door event happens; moreover, the primary door $D'(t)$ must participate in that event, that is, if $D'(t) = D$, then $t = \inf I_D$ or $t = \sup I_D$.

Consider an interval $I' \subseteq I$ over which the choice of the primary door remains constant, that is, there is a door $D\colon I_D \to \mathcal{F}(\mathbb{R}^2)$ such that $I' \subseteq I_D$ and $D'(t) = D$ for every $t \in I'$. Since D is a continuous function, so are the functions $y_0 \upharpoonright I'$, $y_1 \upharpoonright I'$, $Y_0 \upharpoonright I'$ and $Y_1 \upharpoonright I'$. Furthermore, it follows from Observation 6 that the segment S is constant or is sweeping continuously to the left at any point in time $t \in I$. By Observation 15, D can only be moving towards the interior of E' in time interval I'. This shows that $E' \upharpoonright I'$ and hence $Y_0 \upharpoonright I'$ and $Y_1 \upharpoonright I'$ are monotonically decreasing functions.

In view of Lemma 5, to complete the proof that $Y_0 \upharpoonright I$ and $Y_1 \upharpoonright I$ are monotonically decreasing, it remains to prove that

- $Y_0(\nearrow t) \supseteq Y_0(t)$ and $Y_1(\nearrow t) \supseteq Y_1(t)$ whenever $D'(\nearrow t) \neq D'(t)$, for $t \in I \setminus \{\inf I\}$,
- $Y_0(\searrow t) \subseteq Y_0(t)$ and $Y_1(\searrow t) \subseteq Y_1(t)$ whenever $D'(\searrow t) \neq D'(t)$, for $t \in I \setminus \{\sup I\}$.

We consider the kinds of door events as identified in Section 4.3, looking for events happening at time $t \in I$ that result in a primary door being added to or removed from $\mathcal{D}$.

1. A fence splits into two doors. Since $S(t)$ can only be sweeping to the left, both polygon edges incident to the corner triggering that event lie to the left of $S(t)$. Therefore, by

Observation 11, the two doors are a left-door followed by a right-door in the order on $\mathcal{D}(t)$. Consequently, by Observation 14, neither of the two doors can be primary.

2. Two doors merge into a fence. If $D'(t)$ is one of the two doors, then the choice of the primary door changes to some door $D \in \mathcal{D}(\searrow t) \subset \mathcal{D}(t)$ that is after $D'(t)$ in the order on $\mathcal{D}(t)$. By Lemma 9, the endpoints $y_0(\searrow t)$ and $y_1(\searrow t)$ of $D(t)$ lie on $Y_0(t)$ and $Y_1(t)$, respectively, so $Y_0(\searrow t) \subseteq Y_0(t)$ and $Y_1(\searrow t) \subseteq Y_1(t)$ as required.
3. A door splits into a smaller door and a fence. It follows from Observation 11 that the door added to $\mathcal{D}(t)$ maintains the weight of the door removed from $\mathcal{D}(\nearrow t)$. Therefore, assuming $D'(t) \neq D'(\nearrow t)$, $D'(t)$ is the door added to $\mathcal{D}(t)$ and $D'(\nearrow t)$ is the one removed from $\mathcal{D}(\nearrow t)$. Let $w \in P_k$ denote the corner that triggers the event, where $k \in \{0,1\}$. It follows that $y_{1-k}(t) = y_{1-k}(\nearrow t)$, so $Y_{1-k}(t) = Y_{1-k}(\nearrow t)$. Since $w = y_k(t)$, we need to prove that $w \in Y_k(\nearrow t)$. Let $D = D'(\nearrow t)$ and let t_0 be a value in $I_D \cap I$ such that $t_0 < t$ and no door event happens in time interval $[t_0, t)$. For every $t' \in [t_0, t)$, let $\varphi(t')$ be the point on $D(t')$ closest to w. Since D moves continuously, φ is a continuous curve. Since $\varphi(t') \in E'(t')$ and $E'(t') \subset E'(t_0)$ as shown before for every $t' \in [t_0, t)$, φ must be contained in $E'(t_0)$. Since $w \in D(\nearrow t)$, we have $\varphi(\nearrow t) = w$. Furthermore, $E'(t_0)$ is a closed set, and hence $w \in E'(t_0)$. The interior of $E'(t_0)$ is disjoint from P_0 and P_1, so w must be a corner on the chain $Y_k(t_0) = P_k[y_k(t_0), r_k]$. Since $w = y_k(t)$, it follows that $Y_k(t) \subseteq Y_k(t_0)$. By letting t_0 approach t from below, we get $Y_k(t) \subseteq Y_k(\nearrow t)$.
4. A door and a fence merge into a larger door. Again, it follows from Observation 11 that the door added to $\mathcal{D}(\searrow t)$ maintains the weight of the door removed from $\mathcal{D}(t)$. Therefore, assuming $D'(t) \neq D'(\searrow t)$, $D'(t)$ is the door removed from $\mathcal{D}(t)$ and $D'(\searrow t)$ is the one added to $\mathcal{D}(\searrow t)$. Let $w \in P_k$ denote the corner that triggers the event, where $k \in \{0,1\}$. It follows that $y_{1-k}(t) = y_{1-k}(\searrow t)$, so $Y_{1-k}(t) = Y_{1-k}(\searrow t)$. We make an argument similar to the one in the above case to show that $Y_k(\searrow t) \subseteq Y_k(t)$, but using reversed time. Let $D = D'(\searrow t)$ and let t_0 be a value in $I_D \cap I$ such that $t_0 > t$ and no door event happens in time interval $(t, t_0]$. For every $t' \in (t, t_0]$, let $\varphi(t')$ be the point on $D(t')$ closest to w. Since D moves continuously, φ is a continuous curve. For every $t' \in [0, m]$, let $F(t')$ be the polygonal region bounded by $P_0[\ell_0, y_0(t')]$, the primary door at time t', $P_1[\ell_1, y_1(t')]$, and the segment $L = \ell_0\ell_1$. Thus $F(t')$ is a sort of complementary region to $E'(t')$ in the region E. Since E' is monotonically decreasing on $(t, t_0]$ as shown before, F' is monotonically increasing on $(t, t_0]$. Therefore, since $\varphi(t') \in F(t')$, φ must be contained in $F(t_0)$. Since $w \in D(\searrow t)$, we have $\varphi(\searrow t) = w$. Furthermore, $F(t_0)$ is a closed set, and hence $w \in F(t_0)$. The interior of $F(t_0)$ is disjoint from P_0 and P_1, so w must be a corner on the chain $P_k[\ell_k, y_k(t_0)]$. Since $w = y_k(t)$, it follows that $P_k[\ell_k, y_k(t)] \subseteq P_k[\ell_k, y_k(t_0)]$. By letting t_0 approach t from above, we get $P_k[\ell_k, y_k(t)] \subseteq P_k[\ell_k, y_k(\searrow t)]$. Hence, $y_k(\searrow t)$ is on the chain $Y_k(t) = P_k[y_k(t), r_k]$ and therefore $Y_k(\searrow t) \subseteq Y_k(t)$.

Now, we return to the general case of functions y_0, y_1, Y_0 and Y_1 defined on the entire interval $[0, m]$. Consider a point in time $t \in [0, m)$ that is a discontinuity of q_k, where $k \in \{0,1\}$. That is, s_k jumps from $s_k(t)$ to $s_k(\searrow t)$ at time t. We shall see that the jump of s_k has no effect on the choice of the primary door. By Observation 7, the point $p_k[s_k(\searrow t)] = q_k(\searrow t)$ lies on the ray $\mathcal{R}(q_{1-k}(t), q_k(t))$ and the chain $P_k[q_k(t), q_k(\searrow t)]$ belongs to $\mathrm{RHP}(q_0(t), q_1(t))$. Let $\mathcal{D}(t)$ and $\mathcal{D}(\searrow t)$ denote the sets of doors as defined for the segments $S(t)$ and $S(\searrow t)$, respectively. We shall prove that the primary door with respect to $\mathcal{D}(t)$ (i.e., defined for $S(t)$) is the same as with respect to $\mathcal{D}(\searrow t)$ (i.e., defined for $S(\searrow t)$).

Suppose $q_k(\searrow t)$ is on the segment $S(t)$. It follows that $S(\searrow t) \subset S(t)$, $\mathcal{D}(\searrow t) \subseteq \mathcal{D}(t)$, and $\mathcal{D}(t) \setminus \mathcal{D}(\searrow t)$ is the set of doors on $S(t) \setminus S(\searrow t)$. Since $P_k[q_k(t), q_k(\searrow t)] \subset \mathrm{RHP}(q_0(t), q_1(t))$, it follows from Lemma 10 that the doors in $\mathcal{D}(t) \setminus \mathcal{D}(\searrow t)$ occur in pairs of a left-door followed by a right-door, consecutive in the order on $\mathcal{D}(t)$. Therefore, the weights of every door $D \in \mathcal{D}(\searrow t)$ with respect to the sets of doors $\mathcal{D}(\searrow t)$ and $\mathcal{D}(t)$ are equal. By Observation 14, none of the doors in $\mathcal{D}(t) \setminus \mathcal{D}(\searrow t)$ can be primary with respect to $\mathcal{D}(t)$, so the primary door is the same with respect to $\mathcal{D}(t)$ as with respect to $\mathcal{D}(\searrow t)$.

Now, suppose $q_k(\searrow t)$ is not on the segment $S(t)$. It follows that $S(t) \subset S(\searrow t)$, $\mathcal{D}(t) \subseteq \mathcal{D}(\searrow t)$, and $\mathcal{D}(\searrow t) \setminus \mathcal{D}(t)$ is the set of doors on $S(\searrow t) \setminus S(t)$. An argument analogous to that for $q_k(\searrow t) \in S(t)$ above shows that the primary door is the same with respect to $\mathcal{D}(\searrow t)$ as with respect to $\mathcal{D}(t)$.

To conclude, let $t_0 = 0$, $t_1, \dots, t_{n-1}$ be the discontinuities of q_0 or q_1 ordered so that $t_1 < \dots < t_{n-1}$, and $t_n = m$, and consider the closed intervals $I_i = [t_{i-1}, t_i]$ for $i \in \{1, \dots, n\}$. Fix an index i and consider the restrictions $q_0 \restriction I_i$ and $q_1 \restriction I_i$. Only one of them, say $q_k \restriction I_i$, is not continuous, and the only discontinuity of $q_k \restriction I_i$ is t_{i-1}. As we have proved above, if we redefine $q_k(t_{i-1})$ by letting $q_k(t_{i-1}) = q_k(\searrow t_{i-1})$, the primary door $D'(t_{i-1})$ does not change, but then $q_k \restriction I_i$ becomes continuous. Therefore, what we have proved for restrictions of Y_0 and Y_1 to subintervals $I \subseteq [0, m]$ such that $q_0 \restriction I$ and $q_1 \restriction I$ are continuous implies that $Y_0 \restriction I_i$ and $Y_1 \restriction I_i$ are monotonically decreasing, for every $i \in \{1, \dots, n\}$. The assumptions of Lemma 5 are satisfied for Y_0 and Y_1, so Y_0 and Y_1 are monotonically decreasing in the entire domain $[0, m]$. ◀

We are now ready to prove Lemma 2. Using the continuous interpretation of the algorithm, it can be rephrased as follows.

▶ **Lemma 17.** *For any $t \in [0, m]$, we have $0 \le s_0(t) < 2n_0$ and $0 \le s_1(t) < 2n_1$.*

Proof. We only present the proof of the bound on $s_0(t)$. That for $s_1(t)$ is analogous. Let $c_0(0)$ be the unique real in the interval $[0, n_0)$ such that $y_0(0) = p_0[c_0(0)]$. Let $\hat{c}_0$ be the unique real in the interval $[c_0(0), c_0(0) + n_0)$ such that $r_0 = p_0[\hat{c}_0]$. By Lemma 16, for $t \in (0, m]$, there is a unique real $c_0(t) \in [c_0(0), \hat{c}_0]$ such that $y_0(t) = p_0[c_0(t)]$, and this defines a nondecreasing function $c_0 \colon [0, m] \to \mathbb{R}$ with one-sided limits and finitely many discontinuities. Obviously, $0 \le s_0(t)$ and $c_0(t) \le \hat{c}_0 < c_0(0) + n_0 < 2n_0$. It remains to prove $s_0(t) \le c_0(t)$ for $t \in [0, m]$.

We first prove that for every $t \in [0, m)$ with $s_0(t) \le c_0(t)$, there is $\varepsilon > 0$ such that $s_0(t') \le c_0(t')$ for all $t' \in [t, t + \varepsilon)$. Let $t \in [0, m)$ be such that $s_0(t) \le c_0(t)$. First, suppose s_0 is continuous and either constant or strictly increasing on some interval $[t, t + \varepsilon)$ with $\varepsilon > 0$. If $s_0(t) < c_0(t)$, then the statement is clear, so suppose $s_0(t) = c_0(t)$. If s_0 is constant on $[t, t + \varepsilon)$, then the statement is clear, as c_0 is nondecreasing. If s_0 is strictly increasing on $[t, t + \varepsilon)$, we either have $c_0(t') = s_0(t')$ for $t' \in [t, t + \varepsilon')$, for some $\varepsilon' \in (0, \varepsilon]$, or c_0 jumps at time t to a higher value, that is, $c_0(t) < c_0(\searrow t)$. In both cases, the statement holds.

Now, suppose s_0 jumps at time t, that is, $s_0(t) < s_0(\searrow t)$. By choosing $\varepsilon > 0$ small enough, we can assume that s_0 is continuous on the interval $(t, t + \varepsilon)$ and that the points $\{p_0[s_0(t')] \colon t' \in (t, t + \varepsilon)\}$ are a part of one edge e of P_0, which also contains the point $p_0[s_0(\searrow t)]$. By Observation 7, we have $P_0[s_0(t), s_0(\searrow t)] \subset \mathrm{RHP}(p_0[s_0(t)], p_1[s_1])$. This and the facts that $p_0[c_0(t')] \in S(t')$ and $S(t') \cap \mathrm{RHP}(p_0[s_0(t)], p_1[s_1]) = \{p_1[s_1]\}$ imply $c_0(t') > s_0(\searrow t)$, for every $t' \in (t, t + \varepsilon)$. We conclude that for every $t' \in (t, t + \varepsilon)$, either $c_0(t') = s_0(t')$ or $p_0[c_0(t')]$ is on an edge of P_0 other than e, in which case $c_0(t') > s_0(t')$.

We now return to proving that $s_0(t) \le c_0(t)$ for every $t \in [0, m]$. Suppose the contrary, and let $t^\star = \inf\{t \in [0, m] \colon s_0(t) > c_0(t)\}$. In view of the discussion above, we must have $s_0(t^\star) > c_0(t^\star)$. Then $t^\star > 0$, because $c_0(0) \ge 0 = s_0(0)$. By the definition of s_0, we have $s_0(\nearrow t^\star) = s_0(t^\star) > c_0(t^\star) \ge c_0(\nearrow t^\star)$. This contradicts the definition of $t^\star$. ◀

References

1 M. Abrahamsen. An optimal algorithm computing edge-to-edge visibility in a simple polygon. In *25th Canadian Conference on Computational Geometry (CCCG 2013)*, pages 157–162, 2013.

2 M. Abrahamsen. An optimal algorithm for the separating common tangents of two polygons. In *31st International Symposium on Computational Geometry (SoCG 2015)*, volume 34 of *LIPIcs*, pages 198–208, 2015. arXiv:1511.04036 (corrected version).

3 T. Asano, K. Buchin, M. Buchin, M. Korman, W. Mulzer, G. Rote, and A. Schulz. Memory-constrained algorithms for simple polygons. *Comput. Geom.*, 46(8):959–969, 2013.

4 T. Asano, W. Mulzer, G. Rote, and Y. Wang. Constant-work-space algorithms for geometric problems. *J. Comput. Geom.*, 2(1):46–68, 2011.

5 L. Barba, M. Korman, S. Langerman, K. Sadakane, and R.I. Silveira. Space–time trade-offs for stack-based algorithms. *Algorithmica*, 72(4):1097–1129, 2015.

6 L. Barba, M. Korman, S. Langerman, and R.I. Silveira. Computing the visibility polygon using few variables. *Comput. Geom.*, 47(9):918–926, 2014.

7 G.S. Brodal and R. Jacob. Dynamic planar convex hull. In *43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2002)*, pages 617–626, 2002.

8 E. Carson, J. Demmel, L. Grigori, N. Knight, P. Koanantakool, O. Schwartz, and H.V. Simhadri. Write-avoiding algorithms. Technical Report UCB/EECS-2015-163, University of California, Berkeley, 2015.

9 O. Darwish and A. Elmasry. Optimal time-space tradeoff for the 2D convex-hull problem. In *European Symposium on Algorithms (ESA 2014)*, volume 8737 of *LNCS*, pages 284–295. Springer, 2014.

10 L. Guibas, J. Hershberger, and J. Snoeyink. Compact interval trees: a data structure for convex hulls. *Int. J. Comput. Geom. Appl.*, 1(1):1–22, 1991.

11 S. Har-Peled. Shortest path in a polygon using sublinear space. In *31st International Symposium on Computational Geometry (SoCG 2015)*, volume 34 of *LIPIcs*, pages 111–125, 2015.

12 J. Hershberger and S. Suri. Applications of a semi-dynamic convex hull algorithm. *BIT Numer. Math.*, 32(2):249–267, 1992.

13 D. Kirkpatrick and J. Snoeyink. Computing common tangents without a separating line. In *4th International Workshop on Algorithms and Data Structures (WADS 1995)*, volume 955 of *LNCS*, pages 183–193. Springer, 1995.

14 M. Korman, W. Mulzer, A. van Renssen, M. Roeloffzen, P. Seiferth, and Y. Stein. Time-space trade-offs for triangulations and voronoi diagrams. In *Workshop on Algorithms and Data Structures (WADS 2015)*, volume 9214 of *LNCS*, pages 482–494. Springer, 2015.

15 A.A. Melkman. On-line construction of the convex hull of a simple polyline. *Inform. Process. Lett.*, 25(1):11–12, 1987.

16 M.H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *J. Comput. System Sci.*, 23(2):166–204, 1981.

17 F.P. Preparata and S.J. Hong. Convex hulls of finite sets of points in two and three dimensions. *Commun. ACM*, 20(2):87–93, 1977.

18 G.T. Toussaint. Solving geometric problems with the rotating calipers. In *IEEE Mediterranean Electrotechnical Conference (MELECON 1983)*, pages A10.02/1–4, 1983.

Sublinear Distance Labeling

Stephen Alstrup[*1], **Søren Dahlgaard**[†2], **Mathias Bæk Tejs Knudsen**[‡3], **and Ely Porat**[4]

1 **University of Copenhagen, Copenhagen, Denmark**
`s.alstrup@di.ku.dk`

2 **University of Copenhagen, Copenhagen, Denmark**
`soerend@di.ku.dk`

3 **University of Copenhagen, Copenhagen, Denmark**
`knudsen@di.ku.dk`

4 **Bar-Ilan University, Ramat Gan, Israel**
`poratelY@cs.biu.ac.il`

Abstract

A distance labeling scheme labels the n nodes of a graph with binary strings such that, given the labels of any two nodes, one can determine the distance in the graph between the two nodes by looking only at the labels. A D-preserving distance labeling scheme only returns precise distances between pairs of nodes that are at distance at least D from each other. In this paper we consider distance labeling schemes for the classical case of unweighted and undirected graphs.

We present a $O(\frac{n}{D}\log^2 D)$ bit D-preserving distance labeling scheme, improving the previous bound by Bollobás et al. [SIAM J. Discrete Math. 2005]. We also give an almost matching lower bound of $\Omega(\frac{n}{D})$. With our D-preserving distance labeling scheme as a building block, we additionally achieve the following results:

1. We present the first distance labeling scheme of size $o(n)$ for sparse graphs (and hence bounded degree graphs). This addresses an open problem by Gavoille et al. [J. Algo. 2004], hereby separating the complexity from distance labeling in general graphs which require $\Omega(n)$ bits, Moon [Proc. of Glasgow Math. Association 1965].[1]
2. For approximate r-additive labeling schemes, that return distances within an additive error of r we show a scheme of size $O\left(\frac{n}{r}\cdot\frac{\operatorname{polylog}(r\log n)}{\log n}\right)$ for $r \geq 2$. This improves on the current best bound of $O\left(\frac{n}{r}\right)$ by Alstrup et. al. [SODA 2016] for sub-polynomial r, and is a generalization of a result by Gawrychowski et al. [arXiv preprint 2015] who showed this for $r = 2$.

1998 ACM Subject Classification G.2.2 Graph Theory, E.1 Data Structures, G.2.1 Combinatorics

Keywords and phrases Graph labeling schemes, Distance labeling, Graph theory, Sparse graphs

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.5

[*] Research partly supported by the FNU project AlgoDisc – Discrete Mathematics, Algorithms, and Data Structures.
[†] Research partly supported by Mikkel Thorup's Advanced Grant from the Danish Council for Independent Research under the Sapere Aude research career programme.
[‡] Research partly supported by the FNU project AlgoDisc – Discrete Mathematics, Algorithms, and Data Structures. Research partly supported by Mikkel Thorup's Advanced Grant from the Danish Council for Independent Research under the Sapere Aude research career programme.
[1] This result for sparse graphs was made available online in a preliminary version of this paper [6]. The label size was subsequently slightly improved by an $O(\log\log n)$ factor by Gawrychowski et al. [25].

1 Introduction

The concept of *informative labeling schemes* dates back to Breuer and Folkman [12, 13] and was formally introduced by Kannan et al. [30, 34]. A labeling scheme is a way to represent a graph in a distributed setting by assigning bit strings (called *labels*) to each node of the graph. In a distance labeling scheme we assign labels to a graph G from a family $\mathcal{G}$ such that, given *only* the labels of a pair of nodes, we can compute the distance between them without the need for a centralized data structure. When designing a labeling scheme the main goal is to minimize the *maximum label size* over all nodes of all graphs G in the family $\mathcal{G}$. We call this the size of the labeling scheme. As a secondary goal some papers consider the *ecoding* and *decoding* time of the labeling scheme in various computational models. In this paper we study the classical case of *undirected* and *unweighted* graphs.

Exact distances. The problem of exact distance labeling in general graphs is a classic problem that was studied thoroughly in the 1970/80's. Graham and Pollak [26] and Winkler [39] showed that labels of size $\lceil (n-1) \cdot \log_2 3 \rceil$ suffice in this case. Combining [30] and [33] gives a lower bound of $\lceil n/2 \rceil$ bits (see also [24]). Recently, Alstrup et al. [7] improved the label size to $\frac{\log_2 3}{2} n + O\left(\log^2 n\right)$ bits.

Distance labeling schemes have also been investigated for various families of graphs, providing both upper and lower bounds. For trees, Peleg [35] showed that labels of size $O(\log^2 n)$ suffice with a matching lower bound by Gavoille et. al [24]. Gavoille et. al [24] also showed a $\Omega(n^{1/3})$ lower bound for planar graphs and $\Omega(\sqrt{n})$ bound for bounded degree (and thus sparse) graphs. They also provided an $O(\sqrt{n}\log n)$ labeling scheme for planar graphs, however nothing better than the $O(n)$ scheme for general graphs is known for bounded-degree graphs. It remains a major open problem in the field of labeling schemes whether a scheme of size $O(\sqrt{n})$ or even $o(n)$ exists for bounded-degree graphs as stated in e.g. [24].

Other families of graphs studied include distance-hereditary [22], bounded clique-width [16], some non-positively curved plane [15], as well as interval [23] and permutation graphs [10].

Approximate distances. For some applications, the $\Omega(poly(n))$ requirement on the label size for several graph classes is prohibitive. Therefore a large body of work is dedicated to labeling schemes for approximating distances in various families of graphs [2, 14, 19, 24, 27, 28, 32, 35, 36, 37, 38]. Such labeling schemes often provide efficient implementations of other data structures like distance oracles [37] and dynamic graph algorithms [2].

In [35] a labeling scheme of size $O(\log^2 n \cdot \kappa \cdot n^{1/\kappa})$ was presented for approximating distances up to a factor[2] of $\sqrt{8\kappa}$. In [37] a scheme of poly-logarithmic size was given for planar graphs when distances need only be reported within a factor of $(1+\varepsilon)$. Labeling schemes of additive error have also been investigated. For general graphs Alstrup et. al [7] gave a scheme of size $O(n/r)$ for r-additive distance labeling with $r \geq 2$ and a lower bound of $\Omega(\sqrt{n/r})$ was given by Gavoille et al. [21]. For $r = 1$ a lower bound of $\Omega(n)$ can be established by observing that such a scheme can answer adjacency queries in bipartite graphs.

Distance preserving. An alternative to approximating all distances is to only report exact distances above some certain threshold D. A labeling scheme, which reports exact distances

[2] This does not break the Girth Conjecture, as the labeling scheme may under-estimate the distance as well.

for nodes u, v where $dist(u,v) \geq D$ is called a *D-preserving distance labeling scheme*[3]. Bollobás et al. [11] introduced this notion and gave a labeling scheme of size $O(\frac{n}{D}\log^2 n)$ for both directed and undirected graphs. They also provided an $\Omega(\frac{n}{D}\log D)$ lower bound for directed graphs.

1.1 Related work

A problem closely related to distance labeling is adjacency labeling. For some classes such as general graphs the best-known lower bounds for distance is actually that of adjacency. Adjacency labeling has been studied for various classes of graphs. In [8] the label size for adjacency in general undirected graphs was improved from $n/2 + O(\log n)$ [30, 33] to optimal size $n/2+O(1)$, and in [5] adjacency labeling for trees was improved from $\log_2 n+O(\log^* n)$ [9] to optimal size $\log_2 n + O(1)$.

Distance labeling schemes and related 2-hop labeling are used in SIGMOD and is central for some real-world applications [4, 17, 29]. Approximate distance labeling schemes have found applications in several fields such as reachability and distance oracles [37] and communication networks [35]. An overview of distance labeling schemes can be found in [7].

1.2 Our results

We address open problems of [7, 11, 24] improving the label sizes for *exact distances in sparse graphs*, *r-additive distance in general graphs*, and *D-preserving distance labeling.* We do this by showing a strong relationship between D-preserving distance labeling and several other labeling problems using D-preserving distance labels as a black box. Thus, by improving the result of [11] we are able to obtain the first sublinear labeling schemes for several problems studied at SODA over the past decades. Our results are summarized below.

Sparse graphs. We present the first sublinear distance labeling scheme for sparse graphs giving the following theorem:

▶ **Theorem 1.** *Let $\mathcal{S}_n$ denote the family of undirected and unweighted graphs on n nodes with at most $n^{1+o(1)}$ edges. Then there exists a distance labeling scheme for $\mathcal{S}_n$ with maximum label size $o(n)$.*

As noted, prior to this work the best-known bound for this family was the $O(n)$ scheme of [7] for general graphs. Thus, Theorem 1 separates the family of sparse graphs from the family of general graphs requiring $\Omega(n)$ label size. Our result uses a black-box reduction from sparse graphs to the D-preserving distance scheme of Theorem 3 below. The result of Theorem 1 was made available online in a preliminary version of this paper [6] and was subsequently slightly improved by Gawrychowski et al. [25] by noting, that one of the steps in the construction of our D-preserving distance scheme can be skipped when only considering sparse graphs[4].

Approximate labeling schemes. For r-additive distance labeling Gawrychowski et al. [25] showed that a sublinear labeling scheme for sparse graphs implies a sublinear labeling scheme

[3] In this paper we adopt the convention that the labeling scheme returns an upper-bound if $dist(u,v) < D$.

[4] The scheme presented in this paper has labels of length $O\left(\frac{n \operatorname{polylog} \Delta}{\Delta}\right)$, where $\Delta = \frac{\log n}{1+\log\frac{m+n}{n}}$. In [25] they improve the exponent of the polylog Δ term from 2 to 1.

for $r = 2$ in general graphs. We generalise this result to $r \geq 2$ by a reduction to the D-preserving scheme. We note that a reduction to sparse graphs does not suffice in this case, and the scheme of [25] thus only works for $r = 2$. More precisely, we show the following:

▶ **Theorem 2.** *For any $r \geq 2$, there exists an approximate r-additive labeling schemes for the family $\mathcal{G}_n$ of undirected and unweighted graphs on n nodes with maximum label size*

$$O\left(\frac{n}{r} \cdot \frac{\operatorname{polylog}(r \log n)}{\log n}\right) .$$

Theorem 2 improves on the previous best bound of $O\left(\frac{n}{r}\right)$ by [7] whenever $r = 2^{o\left(\sqrt{\log n}\right)}$, e.g. when $r = \operatorname{polylog} n$.

D-preserving labeling schemes. For D-preserving labeling schemes we show that:

▶ **Theorem 3.** *For any integer $D \in [1, n]$, there exists a D-preserving distance labeling scheme for the family $\mathcal{G}_n$ of undirected and unweighted graphs on n nodes with maximum label size*

$$O\left(\frac{n}{D} \max\left\{\log^2 D, 1\right\}\right) .$$

Theorem 3 improves the result of [11] by a factor of $O(\log^2 n / \log^2 D)$ giving the first sublinear size labels for this problem for any $D = \omega(1)$. This sublinearity is the main ingredient in showing the results of Theorems 1 and 2. Our scheme uses sampling similar to that of [11]. By sampling fewer nodes we show that not “too many” nodes end up being problematic and handle these separately by using a tree structure similar to [7][5].

Finally, we give an almost matching lower bound showing:

▶ **Theorem 4.** *A D-preserving distance labeling scheme for the family $\mathcal{G}_n$ of undirected and unweighted graphs on n nodes require label size $\Omega(\frac{n}{D})$, when D is an integer in $[1, n-1]$.*

This bound is a slight modification of the $\Omega(\frac{n}{D} \log D)$ lower bound for directed graphs given in [11].

2 Preliminaries

Throughout the paper we adopt the convention that $\lg x = \max(\log_2 x, 1)$ and $\log x = \ln x$. When $x \leq 0$ we define $\lg x = 1$. In this paper we assume the word-RAM model, with word size $w = \Theta(\log n)$. If s is a bitstring we denote its length by $|s|$ and will also use s to denote the integer value of s when this is clear from context. We use $s \circ s'$ to denote concatenation of bit strings. Finally, we use the Elias γ code [18] to encode a bitstring s of unknown length using $2|s|$ bits such that we may concatenate several such bitstrings and decode them again.

Labeling schemes. A *distance labeling scheme* for a family of graphs $\mathcal{G}$ consists of an encoder e and a decoder d. Given a graph $G \in \mathcal{G}$ the encoder computes a *label assignment* $e_G : V(G) \to \{0,1\}^*$, which assigns a *label* to each node of G. The decoder is a function such that given any graph $G \in \mathcal{G}$ and any pair of nodes $u, v \in V(G)$ we have $d(e_G(u), e_G(v)) =$

[5] We note that after making this result available online in a preliminary version [6], the bound of Theorem 3 was slightly improved by Gawrychowski et al. [25] to $O(\frac{n}{D} \log D)$.

$dist_G(u, v)$. Note that the decoder is oblivious to the actual graph G and is only given the two labels $e_G(u)$ and $e_G(v)$.

The *size* of a labeling scheme is defined as the maximum label size $|e_G(u)|$ over all graphs $G \in \mathcal{G}$ and all nodes $u \in V(G)$. If for all graphs $G \in \mathcal{G}$ the mapping e_G is injective we say that the labeling scheme assigns *unique labels* (note that two different graphs $G, G' \in \mathcal{G}$ may share a label).

If the encoder and graph is clear from the context, we will sometimes denote the label of a node u by $\ell(u) = e_G(u)$.

Various computability requirements are sometimes imposed on labeling schemes [1, 30, 31].

3 D-preserving distance labeling schemes

In this section we will prove Theorem 3. Observe first that for $D = 1$ Theorem 3 is exactly the classic problem of distance labeling and we may use the result of [7]. We will therefore assume that $D \geq 2$ for the remainder of this paper. Let us first formalize the definition of a D-preserving distance labeling scheme.

▶ **Definition 5.** Let D be a positive integer let $\mathcal{G}$ be a family of graphs. For each graph $G \in \mathcal{G}$ let $e_G : V(G) \to \{0, 1\}^*$ be a mapping of nodes to labels. Let $d : \{0, 1\}^* \times \{0, 1\}^* \to \mathbb{Z}$ be a decoder. If e and d satisfy the following two properties, we say that the pair (e, d) is a *D-distance preserving labeling scheme* for the graph family $\mathcal{G}$.

1. $d(e_G(u), e_G(v)) \geq dist_G(u, v)$ for all $u, v \in G$ for any $G \in \mathcal{G}$.
2. $d(e_G(u), e_G(v)) = dist_G(u, v)$ for all $u, v \in G$ with $dist_G(u, v) \geq D$ for any $G \in \mathcal{G}$.

The idea of the labeling scheme presented in this section is to first make a labeling scheme for distances in the range $[D, 2D]$ and use this scheme for increasingly bigger distances until all distances of at least D are covered. Loosely speaking, the scheme is obtained by sampling a set of nodes R, such that *most* shortest paths of length at least D contain a node from R. Then all nodes are partitioned into *sick* and *healthy* nodes adding the sick nodes to the set R. All nodes then store their distance to each node of R and healthy nodes will store the distance to all nodes, for which the shortest path is not *covered* by some node in R.

3.1 A sample-based approach

As a warm-up, we first present the $O\left(\frac{n}{D} \log^2 n\right)$ scheme of Bollobás et al. in [11] with a slight modification.

Given a graph $G = (V, E) \in \mathcal{G}$ we pick a random multiset $R \subseteq V$ consisting of $\left\lceil c \cdot \frac{n}{D} \log n \right\rceil$ nodes for a constant c to be decided. Each element of R is picked uniformly and independently at random from V (i.e. the same node might be picked several times)[6]. We order R arbitrarily as $(w_1, \ldots, w_{|R|})$ and assign the label of a node $u \in V$ as

$$\ell(u) = dist_G(u, w_1) \circ dist_G(u, w_2) \circ \ldots \circ dist_G(u, w_{|R|})$$

▶ **Lemma 6.** *Let u and v be two nodes of some graph $G \in \mathcal{G}$. Set*

$$d = \min_{w \in R} dist_G(u, w) + dist_G(v, w) \ . \tag{1}$$

Then $d \geq dist_G(u, v)$ and $d = dist_G(u, v)$ if R contains a node from a shortest path between u and v.

[6] In [11] they instead picked R by including each node of G with probability $\frac{c \log n}{D}$.

Proof. Let $z \in R$ be the node corresponding to the minimum value of (1). We then have $d = dist_G(u,z) + dist_G(z,v)$. By the triangle inequality this implies $d \geq dist(u,v)$.

Now let p be some shortest path between u and v in G and assume that $z \in p$. Then $dist_G(u,v) = dist_G(u,z) + dist_G(z,v)$, implying that $d \leq dist_G(u,v)$, and thus $d = dist(u,v)$. ◀

By Lemma 6 it only remains to show that the set R is likely to contain a node on a shortest path between any pair of nodes $u, v \in V$ with $dist_G(u,v) \geq D$.

▶ **Lemma 7.** *Let R be defined as above. Then the probability that there exists a pair of nodes $u, v \in V$ such that $dist_G(u,v) \geq D$ and no node on the shortest path between u and v is sampled is at most n^{2-c}.*

Proof. Consider a pair of nodes $u, v \in V$ with $dist_G(u,v) \geq D$. Let p be a shortest path between u and v, then $|p| \geq D$. Each element of R has probability at least D/n of belonging to p (independently), so the probability that no element of R belonging to p is at most

$$\left(1 - \frac{D}{n}\right)^{|R|} \leq \exp\left(-\frac{D}{n} \cdot |R|\right) \leq \exp(-c \log n) = n^{-c} \ . \tag{2}$$

Since there are at most n^2 such pairs, by a union bound the probability that there exists a pair u, v with $dist_G(u,v) \geq D$, such that no element on a shortest path between u and v is sampled in R is thus at most $n^2 \cdot n^{-c} = n^{2-c}$ ◀

By setting $c > 2$ we can ensure that the expected number of times we have to re-sample the set R until the condition of Lemma 7 is satisfied is $O(1)$. The labels can be assigned using $O(|R| \log n) = O(\frac{n}{D} \log^2 n)$ bits as each distance can be stored using $O(\log n)$ bits.

3.2 A scheme for medium distances

We now present a scheme, which preserves distances in the range $[D, 2D]$ using $O\big(\frac{n}{D} \log^2 D\big)$ bits. More formally, we present a labeling scheme such that given a family of unweighted undirected graphs $\mathcal{G}$ the encoder and the decoder satisfies the following constraints for any $G \in \mathcal{G}$:

1. $d(e_G(u), e_G(v)) \geq dist_G(u,v)$ for any $u, v \in G$.
2. $d(e_G(u), e_G(v)) = dist_G(u,v)$ for any $u, v \in G$ with $dist_G(u,v) \in [D, 2D]$.

Let such a labeling scheme be called a *$[D, 2D]$-preserving distance labeling scheme.*

The labeling scheme is based on a sampling procedure similar to that presented in Section 3.1, but improves the label size by introducing the notion of *sick* and *healthy* nodes.

Let $G = (V, E) \in \mathcal{G}$. We sample a multiset R of size $2 \cdot \frac{n}{D} \log D$. Similar to Section 3.1, each element of R is picked uniformly at random from V.

▶ **Definition 8.** Let R be as defined above and fix some node u. We say that a node v is *uncovered* for u if $dist_G(u,v) \geq D$ and no node in R is contained in a shortest path between u and v. A node u with more than $\frac{n}{D}$ uncovered nodes is called *sick* and all other nodes are called *healthy.*

Let S denote the set of sick nodes and let $\mathrm{uc}(u)$ denote the set of uncovered nodes for u. The main outline of the scheme is as follows:

1. Each node u stores the distance from itself to each node of $R \cup S$ using a tree structure to be described.

2. If u is healthy, u stores the distance from itself to every $v \in \mathrm{uc}(u)$ for which $dist_G(u, v) \in [D, 2D]$.

We start by showing that the set of sick nodes has size $O(n/D)$ with probability at least $1/2$. This is captured by the following lemma.

▶ **Lemma 9.** *Let R be defined as above and let S be the set of sick nodes. Then*

$$\Pr\left[|S| \geq 2\frac{n}{D}\right] \leq 1/2 \ .$$

Proof. Fix some node $u \in V$ and let $v \in V$ be a node such that $dist_G(u, v) \geq D$. Using the same argument as in (2) of Lemma 7 we see that the probability that v is uncovered for u is at most D^{-2}. Therefore $\mathbf{E}[|\mathrm{uc}(u)|] \leq \frac{n}{D^2}$. By Markov's inequality we have

$$\Pr[u \text{ is sick}] = \Pr\left[|\mathrm{uc}(u)| \geq D \cdot \frac{n}{D^2}\right] \leq \frac{1}{D} \ ,$$

and thus $\mathbf{E}[|S|] \leq n/D$. We again use Markov's inequality to conclude that

$$\Pr\left[|S| \geq 2\frac{n}{D}\right] \leq 1/2 \, .$$ ◀

The goal is now to store the distances to the nodes of $R \cup S$ as well as $\mathrm{uc}(u)$ using few bits. First consider the distances to $R \cup S$. We will store these distances using a tree structure similar to that of [7]. To do this we will use the following algorithm:

1. Let $r \in V$ be an arbitrary node.
2. Let T' be the BFS-tree of r in G rooted in r.
3. For $i \in \{0, \dots, D-1\}$, let $A_i = \{u \in V \mid dist_G(r, u) \equiv i \mod D\}$.
4. Let $j = \arg\min_{i \in \{0,\dots,D-1\}} |A_i|$.
5. Let T be a graph with $V(T) = A_j \cup \{r\} \cup R \cup S$ and $E(T) = \emptyset$.
6. For each $u \in V(T) \setminus \{r\}$ let v be the nearest ancestor of u in $T' \setminus \{u\}$ such that $v \in V(T)$. Add the edge (v, u) to T with weight $dist_G(v, u) = dist_{T'}(v, u)$.

This process is illustrated in Figure 1.

▶ **Lemma 10.** *Let T be the tree created by the algorithm above. Then T contains $O(\frac{n}{D} \log D)$ nodes with probability at least $1/2$ and each edge of T has weight at most D.*

Proof. Let T', r and A_j be as defined in the algorithm above.

The size of T is at most $|S| + |R| + |A_j| + 1$. By our choice of A_j and R this is bounded by

$$|S| + 2 \cdot \frac{n}{D} \log D + \left\lfloor \frac{n}{D} \right\rfloor + 1 \ .$$

Using Lemma 9 we see that this is $O\big(\frac{n}{D} \log D\big)$ with probability at least $1/2$.

Consider now any edge $(u, p(u)) \in E(T)$ and let $d = dist_G(u, r)$. If $d \leq D$ it follows from the definition of T that $dist_G(u, p(u)) \leq D$, as r is an ancestor of all nodes, and thus also u, in T'. If $d > D$ consider the unique path from u to r in T' and denote the nodes on this path as $(u, v_1, v_2, \dots, v_k, r)$. It follows that $dist_G(v_1, r) = d - 1$, $dist_G(v_2, r) = d - 2$, etc. Since $d > D$ we have $k \geq D$ and thus one of $v_1, \dots, v_D$ is contained in the set A_j and has distance at most D to u. It now follows that $dist_G(u, p(u)) \leq D$ and thus $dist_T(u, p(u)) \leq D$. ◀

Using Lemma 10 we are able to store the distance from any node u to all nodes of T by storing the differences between the distance from u to adjacent nodes in T. This is captured in the following lemma:

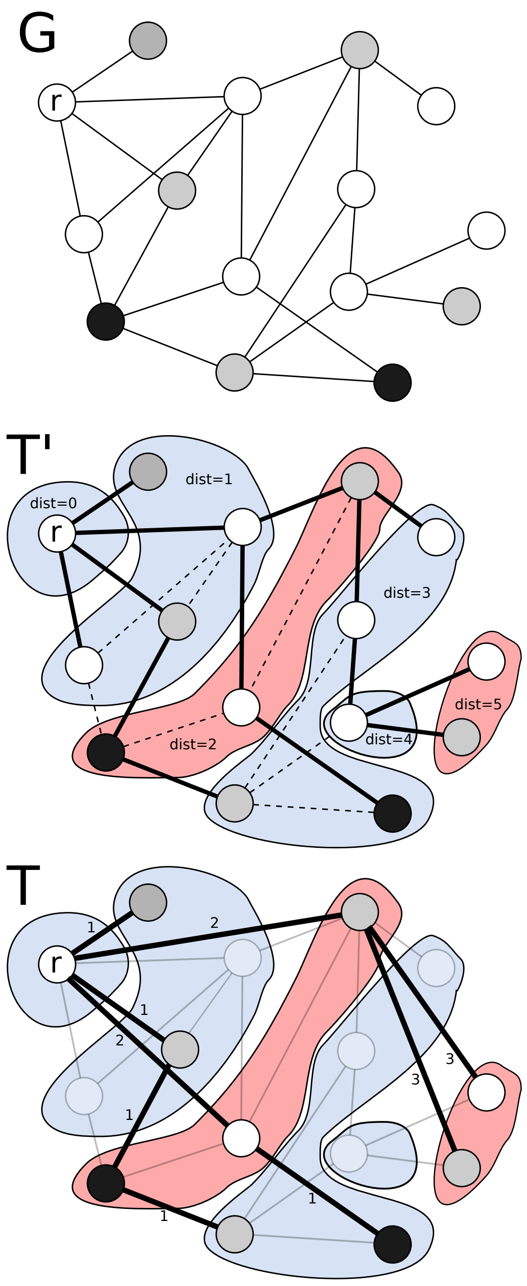


Figure 1 The process of creating T as described above. Gray nodes are the sampled nodes, R, and black nodes are the sick nodes, S. We assume $D = 3$ and pick A_2 as the smallest set (marked in red). *Note that the black nodes are only for illustration and might not actually be sick by our definition.*

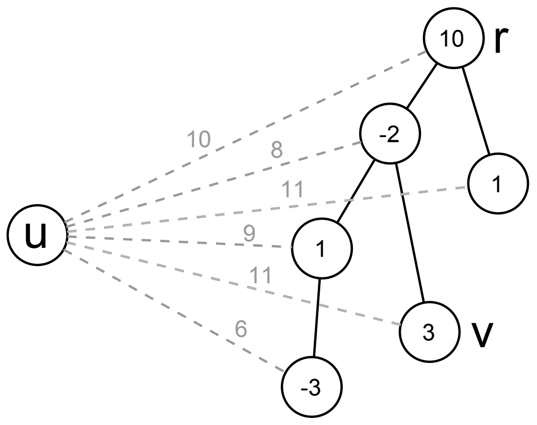


Figure 2 Storing the tree $T = (V', E')$ using few bits. For each node $u \in V'$, we store $dist(u, v) - dist(u, p(v))$. Shortest path distances from u in G are denoted in gray. The distance from u to v is calculated as $10 + (-2) + 1 + (-3) - (-3) - 1 + 3 = 11$.

▶ **Lemma 11.** *Let u be some node in $G = (V, E)$ and let $T = (V', E')$ be the tree resulting from the algorithm above rooted in r. Then we can store the distance from u to every node in T using $O\left(\frac{n}{D}\log^2 D\right)$ bits.*

Proof. Consider the following encoding: We fix some canonical DFS ordering of T and describe it using $2|T|$ bits. This will be the same for all nodes $u \in V$. Next, we store $dist_G(u, r)$ using $\lceil \lg n \rceil$ bits. For each node $v \in V' \setminus \{r\}$ taken in the DFS ordering of T we store $dist_G(u, v) - dist_G(u, p(v))$. Using this description, we can calculate $dist_G(u, v)$ for any $v \in V'$ by summing up the differences on the path from v to r and adding the distance from u to r.

We now argue that $dist_G(u, v) - dist_G(u, p(v))$ can be stored using $\lceil \lg(2D + 1) \rceil$ bits for any node $v \in V' \setminus \{r\}$. Set $t = dist_G(u, p(v))$. By Lemma 10 and the triangle inequality it holds that

$$dist_G(u, v) \le dist_G(u, p(v)) + dist_G(p(v), v) \le t + D\ .$$

Similarly,

$$dist_G(u, v) \ge dist_G(u, p(v)) - dist_G(v, p(v)) \ge t - D\ .$$

Thus, it follows that

$$dist_G(u, v) - dist_G(u, p(v)) \in \{-D, \dots, 0, \dots, D\}\ ,$$

which can be stored using $\lceil \lg(2D + 1) \rceil$ bits. We can thus store all the information using

$$O(2|T| + \log n + |T| \log D) = O\left(\frac{n}{D}\log^2 D\right)$$

bits. ◀

The values $dist_G(u, v) - dist_G(u, p(v))$ are illustrated in Figure 2.

We may now assign the label $\ell(u)$ of a node u to be $id(u)$ concatenated with the bitstring resulting for Lemma 11 and if u is healthy this is concatenated with the id of the nodes in $uc(u)$ whose distance from u is in the interval $[D, 2D]$ along with these distances. The decoder works by simply checking if one nodes stores the others distance or by taking the minimum of going via any node in T.

Label size. In order to bound the size of the label we only need to bound the size of storing id's and distances to the nodes of $uc(u)$ whose distance is in $[D, 2D]$. Since we only store this for healthy nodes this set has size at most n/D and can be described using at most $O(\frac{n}{D}\log D)$ bits. Since each distance can be stored using $O(\log D)$ bits we conclude that the total label size is bounded by $O\big(\frac{n}{D}\log^2 D\big)$.

▶ **Theorem 12.** *There exists a $[D, 2D]$-preserving distance labeling scheme for the family $\mathcal{G}_n$ of undirected and unweighted graphs on n nodes with maximum label size*

$$O\Big(\frac{n}{D}\log^2 D\Big) .$$

Proof. This is a direct corollary of the discussion above. ◀

3.3 Bootstrapping the scheme

In order to show Theorem 3 we will concatenate several instances of the label from Theorem 12. First define $\ell_D(u)$ to be the $[D, 2D]$-preserving distance label for the node u assigned by the scheme of Theorem 12. Now assign the following label to each node u:

$$\ell(u) = \ell_D(u) \circ \ell_{2D}(u) \circ \ell_{4D}(u) \circ \ldots \circ \ell_{2^k D}(u) , \tag{3}$$

where $k = \lfloor \lg(n/D) \rfloor$. Let d_D be the distance returned by running the decoder of Theorem 12 on the corresponding component of the label $\ell(u)$. Then we let the decoder of the full labeling scheme return

$$\hat{d} = \min(d_D, d_{2D}, \ldots, d_{2^k D}) , \tag{4}$$

with k defined as above. We are now ready to prove Theorem 3.

Proof of Theorem 3. Consider any pair of nodes u, v in some graph $G \in \mathcal{G}_n$ and let $d = dist_G(u, v)$. Also, let $\hat{d}$ be the value returned by the decoder for $\ell(u)$ and $\ell(v)$. If $d \le D$ we have $\hat{d} \ge d$. Now assume that $d \in [2^i \cdot D, 2^{i+1} \cdot D]$ for some non-negative integer i. Then, by Theorem 12 and (4) we have $\hat{d} = d$.

The size of the label assigned by (3) is bounded by

$$\begin{aligned}\sum_{i=0}^{\lfloor \lg_2(n/D) \rfloor} O\Big(\frac{n}{2^i \cdot D}\log^2(2^i \cdot D)\Big) &\le \sum_{i=0}^{\infty} O\Big(\frac{n}{2^i \cdot D}\log^2(2^i \cdot D)\Big) \\ &\le O\left(\frac{n}{D}\log^2(D)\sum_{i=1}^{\infty}\frac{i^2+1}{2^i}\right) \\ &= O\Big(\frac{n}{D}\log^2(D)\Big) .\end{aligned}$$

◀

3.4 Lower bound

Proof of Theorem 4. Let $k = \left\lfloor \frac{n}{D+1} \right\rfloor$ and let L and R be sets of k nodes which make up the left and right side of a bipartite graph respectively. Furthermore, let each node of R be the first node on a path of D nodes.

Consider now the family of all such bipartite graphs (L, R) with the attached paths. There are exactly 2^{k^2} such graphs.

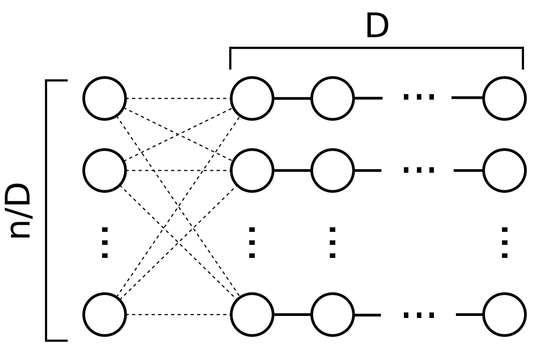


Figure 3 Illustration of the graph family used in the proof of Section 3.4.

Now observe, that a node $u \in L$ is adjacent to a node $v \in R$ if and only if $dist(u, w) = D$, where w is the last node on the path starting in v. By querying all such pairs (u, w) we obtain k^2 bits of information using only $2k$ labels, thus at least one label of size

$$\frac{k^2}{2k} = \frac{\left\lfloor \frac{n}{D+1} \right\rfloor}{2} \geq \frac{n}{8D}$$

is needed. Since the graph has $\leq n$ nodes this implies the result. ◀

This is illustrated in Figure 3.

4 Sparse and bounded degree graphs

We are now ready to prove Theorem 1. In fact we will show the following more general lemma:

▶ **Lemma 13.** *Let $\mathcal{H}_{n,m}$ denote the family of undirected and unweighted graphs on n nodes with at most m edges. Then there exists a distance labeling scheme for $\mathcal{H}_{n,m}$ with maximum label size*

$$O\left(\frac{n}{D} \cdot \log^2 D\right), \text{ where } D = \frac{\log n}{1 + \log \frac{m+n}{n}}$$

Since $\frac{\log n}{1+\log \frac{m+n}{n}} = \omega(1)$ when $m = n^{1+o(1)}$ it will suffice to prove Lemma 13. In order to do so we first show the following lemma for bounded-degree graphs:

▶ **Lemma 14.** *Let $\mathcal{B}_n(\Delta)$ be the family of graphs on n nodes with maximum degree Δ. There exists a distance labeling scheme for $\mathcal{B}_n(\Delta)$ with maximum label size*

$$O\left(\frac{n}{D} \log^2 D\right), \text{ where } D = \frac{\log n}{1 + \log \Delta}$$

Proof. Suppose we are labeling some graph $G \in \mathcal{B}_n(\Delta)$ and let $u \in G$. Let $D = \left\lceil \frac{\log n}{1+2\log \Delta} \right\rceil$ and let $\ell_D(u)$ be the D-distance preserving label assigned by using Theorem 3 with parameter D. Using this label we can deduce the distance to all nodes of distance at least D to u.

Since $G \in \mathcal{B}_n(\Delta)$ there are at most $\Delta^D = O(\sqrt{n})$ nodes closer than distance D to u. Thus, we may describe the IDs and distances of these nodes using at most $O(\sqrt{n} \log n)$ bits. This gives the desired total label size of

$$|\ell(u)| = O\left(\sqrt{n} \log n + \frac{n}{D} \log^2 D\right) = O\left(\frac{n}{D} \log^2 D\right) .$$ ◀

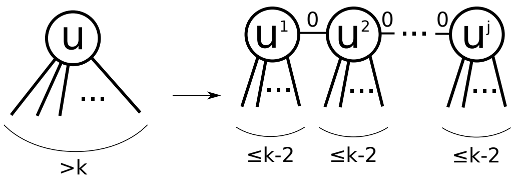


Figure 4 Illustration of the transformation from sparse graph to bounded degree graph.

Using this result we may now prove Lemma 13 by reducing to the bounded degree case in Lemma 14. This has been done before e.g. in distance oracles [20, 3].

Proof of Lemma 13. Let $G \in \mathcal{H}_{n,m}$ be some graph and let $k = \max\left\{\left\lceil \frac{m}{n} \right\rceil, 3\right\}$. Let $u \in G$ be some node with more than k incident edges. If no such node exists, we may apply Lemma 14 directly and we are done. Otherwise we split u into $\lceil \deg(u)/(k-2) \rceil$ nodes and connect these nodes with a path of 0-weight edges. Denote these nodes $u^1, \ldots, u^{\lceil \deg(u)/(k-2) \rceil}$. For each edge (u, v) in G we assign the end-point at u to a node u^i with $\deg(u^i) < k$. This process is illustrated in Figure 4.

Let the graph resulting from performing this process for every node $u \in G$ be denoted by G'. We then have $\Delta(G') \leq k$. Furthermore it holds that for every pair of nodes $u, v \in G$ we have $dist_G(u, v) = dist_{G'}(u^1, v^1)$. Consider now using the labeling scheme of Lemma 14 on G' and setting $\ell(u) = \ell(u^1)$ for each node $u \in G$. By observing that the labeling scheme of Theorem 3 preserves distances for nodes who have at least D *edges* on the shortest path we see that this is actually a distance labeling scheme for G. The number of nodes in G' is bounded by

$$\sum_{u \in G} \left\lceil \frac{\deg(u)}{k-2} \right\rceil \leq \sum_{u \in G} \left(\frac{\deg(u)}{k-2} + 1 \right) = \frac{2m}{k-2} + n = O(n) ,$$

which means that Lemma 14 gives the desired label size. ◀

5 Additive error

We will now show how we can use our D-preserving labeling scheme of Theorem 3 to generalize the 2-additive distance labeling scheme of Gawrychowski et al. [25]. We will assume that $r \leq n^{1/10}$ for simplicity.

Let $t = r \log^{10} n$ and let $D = \frac{r \log n}{4 \log t}$. We describe the scheme in three parts:

1. Let G^r be a copy of G, where an edge is added between any pair of nodes whose distance is at most $r/2$ in G. Let $V^r_{\geq t}$ be the set of nodes in G^r with degree at least t and let S be a minimum dominating set of $V^r_{\geq t}$ in G_r. Then $|S| = O(\frac{n \log t}{t})$.

   For all nodes $u \in G$ we store $dist(u, v)$ and $id(v)$ for all $v \in S$.
2. Consider now the subgraph of G induced by $V \setminus V^r_{\geq t}$. For a node $u \notin V^r_{\geq t}$, let $B_u(D)$ be the ball of radius D around u in this induced subgraph Then $|B_u(D)| \leq t^{2D/r} = O(\sqrt{n})$. This follows from the definition of $V^r_{\geq t}$: There are at most t nodes within distance $r/2$ from u and thus at most t^2 nodes within distance r from u, etc.

   For all $u \notin V^r_{\geq t}$ we store $dist(u, v)$ and $id(v)$ for all $v \in B_u(D)$.
3. Finally we store a D-preserving distance label for all $u \in G$.

The total label size is then

$$O\left(n \cdot \frac{\log n \log t}{t} + \sqrt{n}\log n + \frac{n \log t}{r \log n} \cdot (\log(r \log n))^2\right) = O\left(\frac{n}{r \log n} \cdot \text{polylog}(\log n \cdot r)\right) ,$$

as stated in Theorem 2.

Decoding. To see that the distance between two nodes u and v can be calculated within an additive error r we split into several cases:

- If $dist(u,v) \geq D$ we can report the exact distance between u and v using the D-preserving distance scheme.
- If $dist(u,v) \leq D$ and $deg_{G_r}(v) \geq t$ we can find a node $z \in S$ such that $dist(z,v) \leq r/2$ and thus

  $$dist(u,z) + dist(z,v) \leq dist(u,v) + dist(v,z) + dist(z,v) \leq dist(u,v) + r ,$$

  and symmetrically if $deg_{G_r}(u) \geq t$.
- Finally, if $dist(u,v) \leq D$ and $deg_{G_r}(u) < t$ and $deg_{G_r}(v) < t$, then we $v \in B_u(D)$ and we can thus report the exact distance between u and v.

Acknowledgements. We would like to thank Noy Rotbart for helpful discussions and observations.

References

1 S. Abiteboul, H. Kaplan, and T. Milo. Compact labeling schemes for ancestor queries. In *Proc. of the 12th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 547–556, 2001.

2 I. Abraham, S. Chechik, and C. Gavoille. Fully dynamic approximate distance oracles for planar graphs via forbidden-set distance labels. In *Proc. 44th Annual ACM Symp. on Theory of Computing (STOC)*, pages 1199–1218, 2012.

3 R. Agarwal, P. B. Godfrey, and S. Har-Peled. Approximate distance queries and compact routing in sparse graphs. In *INFOCOM 2011. 30th IEEE International Conference on Computer Communications*, pages 1754–1762, 2011.

4 T. Akiba, Y. Iwata, and Y. Yoshida. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In *ACM International Conference on Management of Data (SIGMOD)*, pages 349–360, 2013. `doi:10.1145/2463676.2465315`.

5 S. Alstrup, S. Dahlgaard, and M. B. T. Knudsen. Optimal induced universal graphs and labeling schemes for trees. In *Proc. 56th Annual Symp. on Foundations of Computer Science (FOCS)*, 2015.

6 S. Alstrup, S. Dahlgaard, M. B. T. Knudsen, and E. Porat. Sublinear distance labeling for sparse graphs. *CoRR*, abs/1507.02618, 2015. URL: `http://arxiv.org/abs/1507.02618`.

7 S. Alstrup, C. Gavoille, E. B. Halvorsen, and H. Petersen. Simpler, faster and shorter labels for distances in graphs. In *Proc. 27th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 338–350, 2016.

8 S. Alstrup, H. Kaplan, M. Thorup, and U. Zwick. Adjacency labeling schemes and induced-universal graphs. In *Proc. of the 47th Annual ACM Symp. on Theory of Computing (STOC)*, 2015.

9 S. Alstrup and T. Rauhe. Small induced-universal graphs and compact implicit graph representations. In *Proc. 43rd Annual Symp. on Foundations of Computer Science (FOCS)*, pages 53–62, 2002.

10 F. Bazzaro and C. Gavoille. Localized and compact data-structure for comparability graphs. *Discrete Mathematics*, 309(11):3465–3484, 2009. doi:10.1016/j.disc.2007.12.091.

11 B. Bollobás, D. Coppersmith, and M. Elkin. Sparse distance preservers and additive spanners. *SIAM J. Discrete Math.*, 19(4):1029–1055, 2005. See also SODA'03. doi:10.1137/S0895480103431046.

12 M. A. Breuer. Coding the vertexes of a graph. *IEEE Trans. on Information Theory*, IT–12:148–153, 1966.

13 M. A. Breuer and J. Folkman. An unexpected result on coding vertices of a graph. *J. of Mathemathical analysis and applications*, 20:583–600, 1967.

14 V. D. Chepoi, F. F. Dragan, B. Estellon, M. Habib, and Y. Vaxès. Diameters, centers, and approximating trees of delta-hyperbolic geodesic spaces and graphs. In 24^{th} *Annual ACM Symp. on Computational Geometry (SoCG)*, pages 59–68, 2008. doi:10.1145/1377676.1377687.

15 V. D. Chepoi, F. F. Dragan, and Y. Vaxès. Distance and routing labeling schemes for non-positively curved plane graphs. *J. of Algorithms*, 61(2):60–88, 2006. doi:10.1016/j.jalgor.2004.07.011.

16 B. Courcelle and R. Vanicat. Query efficient implementation of graphs of bounded clique-width. *Discrete Applied Mathematics*, 131:129–150, 2003. doi:10.1016/S0166-218X(02)00421-3.

17 D. Delling, A. V. Goldberg, R. Savchenko, and R. F. Werneck. Hub labels: Theory and practice. In 13^{th} *International Symp. on Experimental Algorithms (SEA)*, pages 259–270, 2014. doi:10.1007/978-3-319-07959-2_22.

18 P. Elias. Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory*, 21(2):194–203, 1975.

19 M. Elkin, A. Filtser, and O. Neiman. Prioritized metric structures and embedding. In *Proc. of the 47th Annual ACM Symp. on Theory of Computing (STOC)*, pages 489–498, 2015.

20 M. Elkin and S. Pettie. A linear-size logarithmic stretch path-reporting distance oracle for general graphs. In *Proc. of the 26th Annual Symp. on Discrete Algorithms (SODA)*, pages 805–821, 2015.

21 C. Gavoille, M. Katz, N. A. Katz, C. Paul, and D. Peleg. Approximate distance labeling schemes. In *Proc. of the 9th annual European Symp. on Algorithms (ESA)*, pages 476–488, 2001.

22 C. Gavoille and C. Paul. Distance labeling scheme and split decomposition. *Discrete Mathematics*, 273(1-3):115–130, 2003.

23 C. Gavoille and C. Paul. Optimal distance labeling for interval graphs and related graphs families. *SIAM J. Discrete Math.*, 22(3):1239–1258, 2008. doi:10.1137/050635006.

24 C. Gavoille, D. Peleg, S. Pérennes, and R. Raz. Distance labeling in graphs. *J. of Algorithms*, 53(1):85–112, 2004. See also SODA'01. doi:10.1016/j.jalgor.2004.05.002.

25 P. Gawrychowski, A. Kosowski, and P. Uznanski. Even simpler distance labeling for (sparse) graphs. *CoRR*, abs/1507.06240, 2015. URL: http://arxiv.org/abs/1507.06240.

26 R. L. Graham and H. O. Pollak. On embedding graphs in squashed cubes. In *Lecture Notes in Mathematics*, volume 303. Springer-Verlag, 1972.

27 A. Gupta, R. Krauthgamer, and J. R. Lee. Bounded geometries, fractals, and low-distortion embeddings. In *44th Annual Symp. on Foundations of Computer Science (FOCS)*, pages 534–543, 2003. doi:10.1109/SFCS.2003.1238226.

28 A. Gupta, A. Kumar, and R. Rastogi. Traveling with a pez dispenser (or, routing issues in mpls). *SIAM J. on Computing*, 34(2):453–474, 2005. See also FOCS'01.

29 R. Jin, N. Ruan, Y. Xiang, and V. Lee. A highway-centric labeling approach for answering distance queries on large sparse graphs. In *ACM International Conference on Management of Data (SIGMOD)*, pages 445–456, May 2012. doi:10.1145/2213836.2213887.

30 S. Kannan, M. Naor, and S. Rudich. Implicit representation of graphs. *SIAM J. Disc. Math.*, pages 596–603, 1992. See also STOC'88.

31 M. Katz, N. A. Katz, A. Korman, and D. Peleg. Labeling schemes for flow and connectivity. *SIAM J. Comput.*, 34(1):23–40, 2004. See also SODA'02. doi:10.1137/S0097539703433912.

32 R. Krauthgamer and J. R. Lee. Algorithms on negatively curved spaces. In *47th Annual Symp. on Foundations of Computer Science (FOCS)*, pages 119–132, 2006. doi:10.1109/FOCS.2006.9.

33 J. W. Moon. On minimal n-universal graphs. *Proc. of the Glasgow Mathematical Association*, 7(1):32–33, 1965.

34 J. H. Müller. *Local structure in graph classes.* PhD thesis, Georgia Institute of Technology, 1988.

35 D. Peleg. Proximity-preserving labeling schemes. *J. Graph Theory*, 33(3):167–176, 2000.

36 K. Talwar. Bypassing the embedding: algorithms for low dimensional metrics. In *Proc. of the 36th Annual ACM Symp. on Theory of Computing (STOC)*, pages 281–290, 2004. doi:10.1145/1007352.1007399.

37 M. Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *J. ACM*, 51(6):993–1024, 2004. See also FOCS'01. doi:10.1145/1039488.1039493.

38 M. Thorup and U. Zwick. Approximate distance oracles. *J. of the ACM*, 52(1):1–24, 2005. See also STOC'01.

39 P. M. Winkler. Proof of the squashed cube conjecture. *Combinatorica*, 3(1):135–139, 1983. doi:10.1007/BF02579350.

Probabilistic Routing for On-Street Parking Search

Tobias Arndt[1], Danijar Hafner[2], Thomas Kellermeier[3], Simon Krogmann[4], Armin Razmjou[5], Martin S. Krejca[6], Ralf Rothenberger[7], and Tobias Friedrich[8]

1 **Hasso Plattner Institute, Potsdam, Germany**
2 **Hasso Plattner Institute, Potsdam, Germany**
3 **Hasso Plattner Institute, Potsdam, Germany**
4 **Hasso Plattner Institute, Potsdam, Germany**
5 **Hasso Plattner Institute, Potsdam, Germany**
6 **Hasso Plattner Institute, Potsdam, Germany**
7 **Hasso Plattner Institute, Potsdam, Germany**
8 **Hasso Plattner Institute, Potsdam, Germany**

Abstract

An estimated 30 % of urban traffic is caused by search for parking spots [8]. Suggesting routes along highly probable parking spots could reduce traffic. In this paper, we formalize parking search as a probabilistic problem on a road graph and show that it is NP-complete. We explore heuristics that optimize for the driving duration and the walking distance to the destination. Routes are constrained to reach a certain probability threshold of finding a spot. Empirically estimated probabilities of successful parking attempts are provided by TomTom on a per-street basis. We release these probabilities as a dataset of about 80,000 roads covering the Berlin area. This allows to evaluate parking search algorithms on a real road network with realistic probabilities for the first time. However, for many other areas, parking probabilities are not openly available. Because they are effortful to collect, we propose an algorithm that relies on conventional road attributes only. Our experiments show that this algorithm comes close to the baseline by a factor of 1.3 in our cost measure. This leads to the conclusion that conventional road attributes may be sufficient to compute reasonably good parking search routes.

1998 ACM Subject Classification I.2.8 Problem Solving, Control Methods, and Search

Keywords and phrases parking search, on-street parking, probabilistic routing, constrained optimization, dataset

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.6

1 Introduction

Searching for a parking spot is expensive, time-consuming, and causes a significant amount of urban traffic: An estimated 30 % of traffic is due to people searching for parking spots [8]. Drivers could be assisted in their search by suggesting a route along streets with a high probability of yielding a vacant on-street parking spot. In this paper, we investigate the problem of generating these routes. While off-street parking options, such as car parks, are valid alternatives and have been explored, e.g., by Cassady and Kobza [2], we focus on *free of charge on-street parking*, which makes up a majority of the parking capacity in most cities [3].

Parking search is probabilistic by nature. Modeling the road network as a graph, each edge has a probability of having a vacant parking spot. Several works on probabilistic graph routing have been conducted as outlaid in section 2. Most relevant, Jossé, Schmid and Schubert [5] recently proposed a probabilistic model for parking search that we build on.

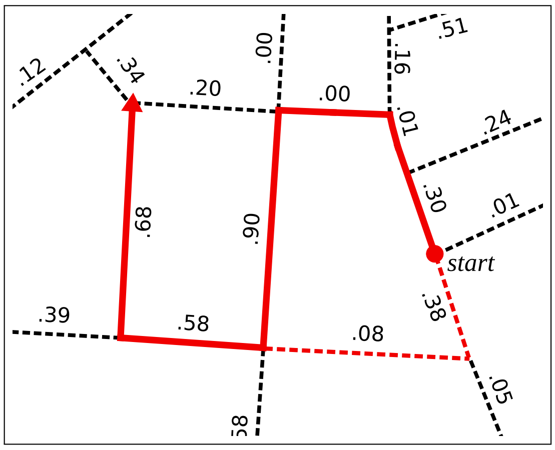


Figure 1 A simple route computed by our **Branch and Bound** algorithm (red line). While a greedy route (dashed red line) would collect more probability mass with the first two segments by leading South and West, the shown route reaches the 90 % edge slightly earlier and can collect the 58 % edge right afterward.

In theory, parking routes can have an infinite length, since a parking spot can never be guaranteed. To evaluate our algorithms, however, we have to restrict routes to a finite length. In our formalization in section 3, we thus introduce a bound on the converse probability mass of routes, similar to previous work [5, 6]. Traversing the graph, at each edge the algorithm collects the probability of having found a parking spot. A route is considered successful if its collected probability mass reaches a certain threshold. See figure 1 for an example of a successful route. We evaluate routes based on the two criteria *driving duration* and *walking distance* to the desired destination.

For generating good parking search routes, we propose two algorithms in Section 4. The first one, **Branch and Bound**, is our baseline. Since it considers routes that reach a probability mass threshold constraint, it requires know ledge of per-street probabilities. For the case where those probabilities are not available, we propose a **Heuristic Search** algorithm that iteratively explores sub-routes scored by a heuristic. This approach comes close to the baseline by a factor of 1.3 in our cost.

Along with our work, we release a *dataset of empirically estimated probabilities* of parking successes collected by TomTom. Those probabilities are given per-street and for each hour of day and day of week. The data was obtained from several million anonymously collected user records. In Section 5, we describe the collection and preprocessing processes. We then explore characteristics of the dataset such as the difference of the probability distributions at day- and nighttime.

Based on the dataset, we evaluate our algorithms in Section 6. First, we compare our **Branch and Bound** and **Heuristic Search** algorithms and a greedy algorithm proposed by Jossé et al. [5] at different *times of day.* Second, we compare different choices for *weighting the cost.* Since our cost is composed of two objectives, we can simulate different preferences. We show that finding a parking spot quickly tends to be easier than finding one that is close to the desired destination. Third, we assess the *impact of inaccurate probabilities* by considering our dataset as ground-truth and showing a disturbed version of it to the algorithms. We show that **Branch and Bound** still works well under a high level of noise but is outperformed by **Heuristic Search** later.

2 Related Work

Routing on conventional road networks has been subject to extensive research. Especially the approach of route queries on probabilistic graphs has recently gained increasing attention. More specifically, the problem of finding a parking spot in urban areas, which can be distinguished into on-street and off-street parking, has been examined.

Kanza et al. [6] calculate routes to a given destination on a probabilistic graph, maximizing the certainty of visiting relevant points of interest. Hua and Pei [4] study routing under uncertain travel time. They either bound probability or duration and optimize for the other. In our scenario, after bounding the probability, we still have to optimize a multi-objective problem with *driving duration* and *walking distance*. Moreover, their algorithms assume a specified destination which does not exist in parking search, thus we can not apply their algorithms to our problem.

Probabilistic routing is usually modeled as a graph of resources that each can either be available or not [7, 6, 5]. Kanza et al. [6] and Jossé et al. [5] both abstract from the road network and span their graphs over resources only. Since we have a probability of parking success for each road, we must span our graph over the complete road network. While this is conceptually the same, it results in large graphs where back-tracking, as used by Jossé et al. [5], is not suitable anymore.

Kanza et al. [6] refer to uncertainty as the probability of a particular resource being relevant and available, comparable to our per-street probabilities. As an answer to a *route-search query*, they suggest two complementary length-bounded and probability-bounded scenarios. We use the latter approach with our probability mass threshold. For on-street parking search, a bounded-probability scenario makes sense since a parking spot can never be guaranteed completely and routes can potentially be infinite.

As mentioned, Jossé et al. [5] propose a resource graph model that they use to answer parking search queries. Their main focus lies on resource reappearance. In their model, the observed state of a resource decays over time, allowing consumed resources to reappear with a certain probability. They differentiate between long-term and short-term observations. Long-term observations correspond to our static probability model, while we do not model short-term observations because real reappearance data is not available.

3 Problem

3.1 Formalization

We now give a formal definition of our parking search scenario and prove that it is NP-complete. We model the road network as a directed graph whose edges are augmented with information about the distance of an edge to the destination, the time to traverse the edge, and the probability of finding a parking spot.

The search process starts at a crossing in the network, given by a specified node v_0. For simplicity, this node is also the desired destination, since we assume that the driver has already reached the destination and now starts looking for a parking spot from there. Our proposed algorithms can work with other destinations without any modifications. The parking route is represented as a path P on the graph and is considered successful if the probability of not finding a parking spot is at most ε.

The overall cost of a successful path is a convex combination with parameter λ of, on the one hand, the time spent not finding a parking place and, on the other hand, the distance to the destination v_0. Note that probabilities do not reappear, that is, an edge can contribute a positive probability of finding a parking spot at most once.

Since a driver usually scans opposing lanes of a single street at once, we further introduce a function D that maps edges of the graph to sets of edges that share a single probability. If we traverse an edge (u, v), the probabilities of all edges in $D\big((u, v)\big)$ disappear as well. Normally, for an edge (u, v), $D\big((u, v)\big)$ would consist of at most (u, v) and (v, u). If, however, opposing lanes were separated, only choosing $D\big((u, v)\big) = \{(u, v)\}$ would make sense.

▶ **Definition 1** (Minimal Parking Spot Search (MPSS)).

Instance: Directed graph $G = (V, E)$, time function $t\colon E \to \mathbb{R}_{\geq 0}$, distance function $d\colon E \to \mathbb{R}_{\geq 0}$, probability function $p\colon E \to [0, 1]$, specified vertex $v_0 \in V$, threshold $\varepsilon \in [0, 1]$, and value $\lambda \in [0, 1]$. Let $D\colon E \to \mathcal{P}(E)$ such that, for all $e \in E$, $e \in D(e)$.

Solution: Edge sequence $P = (e_1, e_2, \dots, e_l) \in E^l$ with $l \leq |E|^2$ such that $e_1 = (v_0, v)$ for some $v \in V$ and $\forall\ i \in \{1, 2, \dots, l-1\}\ \exists\ u, v, w \in V\colon e_i = (u, v) \wedge e_{i+1} = (v, w)$ and $\prod_{\substack{i=1:\\ \forall j<i\colon e_i \notin D(e_j)}}^{l} \big(1 - p(e_i)\big) \leq \varepsilon$.

Measure: Minimize the cost c(P) defined as

$$c(P) = \lambda \sum_{i=1}^{l} t(e_i) \cdot \left(\prod_{\substack{j=1:\\ \forall k<j\colon e_j \notin D(e_k)}}^{i-1} \big(1 - p(e_j)\big) \right) + (1-\lambda) \sum_{\substack{i=1:\\ \forall k<i\colon e_i \notin D(e_k)}}^{l} p(e_i) \cdot d(e_i) \cdot \left(\prod_{\substack{j=1:\\ \forall k<j\colon e_j \notin D(e_k)}}^{i-1} \big(1 - p(e_j)\big) \right) .$$

Note that our measure is equivalent to

$$\sum_{\substack{i=1:\\ \forall k<i\colon e_i \notin D(e_k)}}^{l} p(e_i) \left(\prod_{\substack{j=1:\\ \forall k<j\colon e_j \notin D(e_k)}}^{i-1} \big(1 - p(e_j)\big) \right) \cdot \left(\lambda \sum_{j=1}^{i} t(e_j) + (1-\lambda) \cdot d(e_i) \right) +$$
$$\lambda \sum_{i=1}^{l} t(e_i) \cdot \prod_{\substack{j=1:\\ \forall k<j\colon e_j \notin D(e_k)}}^{l} \big(1 - p(e_j)\big) .$$

The last term models an optimistic extension of the route by a road which contributes zero time, zero distance and a probability of one.

3.2 NP-Completeness

Let k-Minimal Parking Spot Search (k-MPSS) be the decision version of MPSS, i.e., the problem to decide if there is a feasible solution of cost at most k. We prove the NP-completeness of k-MPSS by a polynomial time reduction from Hamiltonian Path, i.e., the problem of finding a simple path in G which visits each node exactly once.

▶ **Theorem 2.** *k-MPSS is NP-complete.*

Proof. First of all, k-MPSS is in NP, since we can guess an edge sequence of size at most $|E|^2$ at random and it can be tested in polynomial time if the edge sequence is a feasible solution and if its cost is at most k.

Now we sketch how to reduce an instance of Hamiltonian Path to an instance of k-MPSS in polynomial time. Suppose we are given a directed graph $G = (V, E)$ as an input for Hamiltonian Path. We construct a new graph $G' = (V', E')$ with $V' = \{v_0\} \cup \big\{v^{\text{in}}, v^{\text{out}} \mid v \in V\big\}$ and $E' = \big\{(v_0, v^{\text{in}}) \mid v \in V\big\} \cup \big\{(v^{\text{in}}, v^{\text{out}}) \mid v \in V\big\} \cup \big\{(u^{\text{out}}, v^{\text{in}}) \mid (u, v) \in E\big\}$. We define

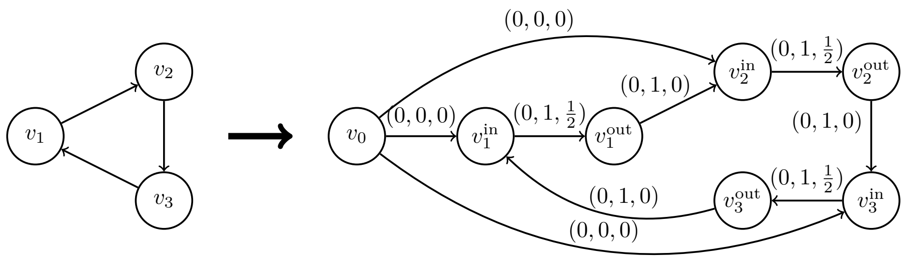


Figure 2 An exemplary reduction from Hamiltonian Path to k-MPSS. The graph on the left is transformed into the one on the right. The triple of an edge e in the right graph has the form $\big(d(e), t(e), p(e)\big)$.

$d(e) = 0$ for all $e \in E'$, $t(e) = 1$ for all $e \in E' \setminus \{(v_0, v^{\text{in}}) \mid v \in V\}$ and $t(e) = 0$ for all $e \in \{(v_0, v^{\text{in}}) \mid v \in V\}$. For all $e \in E'$, we choose $p(e) = \frac{1}{2}$ if $e = (v^{\text{in}}, v^{\text{out}})$ for some $v \in V$ and $p(e) = 0$ otherwise. Further, we define, for all $e \in E'$, $D(e) = \{e\}$, i.e., for two edges e_i and e_j, our proposition of $e_i \notin D(e_j)$ in the problem formalization simplifies to $e_i \neq e_j$. Finally, we choose $\lambda = 1$, $\varepsilon = 2^{-|V|}$ and $k = \sum_{i=1}^{2|V|-1} \left(\frac{1}{2}\right)^{\lfloor i/2 \rfloor}$. An example of such a reduction can be seen in Figure 2.

Suppose the graph G has a Hamiltonian path $(v_{i_1}, v_{i_2}, \dots, v_{i_{|V|}})$. Then the edge sequence

$$\big((v_0, v_{i_1}^{\text{in}}), (v_{i_1}^{\text{in}}, v_{i_1}^{\text{out}}), (v_{i_1}^{\text{out}}, v_{i_2}^{\text{in}}), \dots, (v_{i_{|V|}}^{\text{in}}, v_{i_{|V|}}^{\text{out}})\big)$$

is a feasible solution for k-MPSS. Since $(v_{i_1}, v_{i_2}, \dots, v_{i_{|V|}})$ is a Hamiltonian path, each $(v^{\text{in}}, v^{\text{out}})$-edge is traversed exactly once with exactly one other edge in-between. Also, each edge is traversed at most once. The sequence consists of $2|V| \le |E'|$ edges and amounts to

$$\prod_{\substack{i=0:\\ \forall j<i:\ e_i \neq e_j}}^{l} \big(1 - p(e_i)\big) = 2^{-|V|} = \varepsilon\,.$$

The cost of the sequence is

$$\sum_{i=1}^{2|V|} t(e_i) \cdot \left(\prod_{\substack{j=1:\\ \forall k<j:\ e_j \neq e_k}}^{i-1} \big(1 - p(e_j)\big) \right) = \sum_{i=2}^{2|V|} \left(\prod_{j=1}^{i-1} \big(1 - p(e_j)\big) \right)$$
$$= \sum_{i=2}^{2|V|} \left(\frac{1}{2}\right)^{\lfloor (i-1)/2 \rfloor} = \sum_{i=1}^{2|V|-1} \left(\frac{1}{2}\right)^{\lfloor i/2 \rfloor} = k\ ,$$

since $t(e_1) = 0$, $t(e_i) = 1$ for all $i \ge 2$, $1 - p(e_i) = \frac{1}{2}$ if i even and $1 - p(e_i) = 1$ if i odd.

To transform a solution $(e_1, e_2, \dots, e_l)$ of k-MPSS into a solution of Hamiltonian Path, we simply take the nodes $(v_{i_1}, v_{i_2}, \dots, v_{i_{|V|}})$ according to the order in which their corresponding $(v^{\text{in}}, v^{\text{out}})$-edges are traversed.

Now suppose we have a solution $(e_1, e_2, \dots, e_l)$ of k-MPSS. To achieve

$$\prod_{\substack{i=0:\\ \forall j<i:\ e_i \neq e_j}}^{l} \big(1 - p(e_i)\big) \le 2^{-|V|}\,,$$

all $(v^{\text{in}}, v^{\text{out}})$-edges have to be visited at least once. Due to the construction of G', every edge sequence has to alternate between $(v^{\text{in}}, v^{\text{out}})$-edges and $(u^{\text{out}}, v^{\text{in}})$-edges. The factor in the cost term only changes if we visit a new $(v^{\text{in}}, v^{\text{out}})$-edge. If we visit any other edge or an edge we already visited, the same factor as before is added to the cost. That means, visiting e_i with $i > 1$ adds $(1/2)^{|\{(v^{\text{in}},v^{\text{out}})|v \in V\} \cap \{e_1,e_2,\ldots,e_{i-1}\}|}$ to the cost. Since each $(v^{\text{in}}, v^{\text{out}})$-edge has to be visited at least once and the edge sequence has to be alternating, starting at some edge (v_0, u^{in}) with cost 0, the cost of every feasible edge sequence is at least k. Furthermore, to reach a cost of at most k, each $(v^{\text{in}}, v^{\text{out}})$-edge can be visited at most once, i.e., after transforming the solution, we get a path in G which visits each node exactly once. ◀

4 Algorithms

We now introduce two algorithms for optimizing the problem defined in Section 3. The space of potential solutions for this problem forms a tree where each node holds a route. Each route begins with an outgoing road of the start node, and children in the tree extend their parent route by one possible edge each. Because of the number of outgoing edges at each node, the time of brute-force search in this tree grows exponentially in depth. As we already showed, the problem is NP-complete, so our algorithms only explore this *solution tree* partially, in order to make computations viable on large road networks.

Our **Branch and Bound** algorithm serves as *baseline* for a given probability mass threshold ε. For the case of not having probability data available, we propose a **Heuristic Search** algorithm that explores the solution tree shallowly to compute sub-routes. We iterate this routine to obtain routes of theoretically infinite length. Since we cannot compute the cost of a route without probabilities, we have to make estimations. For this, we mention heuristics varying in their required knowledge of road attributes.

4.1 Branch and Bound

Branch and Bound, as shown in Algorithm 1, has knowledge of per-street probabilities and the evaluation threshold ε. As an outline, we follow the branch and bound paradigm and obtain an initial upper bound B on the cost of a best route from v_0. We then explore the solution tree by pruning intermediate routes that exceed this bound. If a considered route reaches ε before getting pruned, it must be better than the previous bound. Thus, we update B with the cost of this route and proceed exploring the remaining solutions.[1]

For the initial upper bound, we greedily expand a single route from v_0. At each node, we choose an outgoing edge e with the largest term $\frac{p(e)}{c(P+e)-c(P)}$ as the next segment. Eventually, this route reaches ε. The cost of this serves as an initial upper bound $B := c(P)$ on the cost of the best route from v_0 that satisfies the threshold constraint.

We then explore the whole solution tree discarding any intermediate route P' if $c(P') \geq B$. We restrict the number of explored routes with an *expands* parameter. In comparison to restricting the depth of the search, this allows for consistent computation times, as it is independent of the local branching factors of the road network. We traverse the restricted solution tree in a *breadth-first search* manner to look at shorter routes first, with random ordering within each level. We use this algorithm as baseline in the experiments in Section 6.

[1] Jossé et al. [5] also proposed an algorithm based on the branch and bound paradigm. They suggest an expensive forward estimation of the remaining cost of intermediate routes. We found this to be ineffective compared to the increased amount of routes that can be considered otherwise.

Algorithm 1: Branch and Bound($expands$)

```
1  $queue \leftarrow$ queue with empty route;
2  $best \leftarrow$ empty route;
3  while probability mass of $best < 1 - \varepsilon$ do
4      $best \leftarrow best$ concatenated with outgoing edge $e$ where $\frac{p(e)}{c(best+e)-c(best)}$ is largest;
5      if $best.length > 50$ then
6          $c(best) \leftarrow \infty$;
7          break;
8  for $n = 1$ to $expands$ do
9      $route \leftarrow queue.pop()$;
10     foreach outgoing $edge$ from $route$ do
11         if $c(route$ concatenated with $edge) > c(best)$ then
12             continue;
13         if $probability\ mass\ of\ route < 1 - \varepsilon$ then
14             $best \leftarrow route$;
15             continue;
16         $queue.push(route$ concatenated with $edge)$;
17 return $best$;
```

Note that in reality a parking spot can never be guaranteed and routes found by **Branch and Bound** may not lead along a vacant parking spot. The algorithm can be modified to handle this scenario: If the returned route ends before we found a vacant parking spot, we restart the algorithm with the last visited node as new start node. Since it knows per-street probabilities, visited routes stay at zero probability. We use this approach for the experiment on inaccurate probabilities in Section 6.3, where the algorithm sees disturbed probabilities and the returned route might not reach the probability mass threshold.

4.2 Heuristic Search

For some cities, the data used for generating probabilities may not be available and only map data can be used. Therefore, we introduce **Heuristic Search** (Algorithm 2) which computes sub-routes using *breadth-first search*, also limited by the number of *expands*. Since the algorithm does not access probabilities, it cannot evaluate the real cost for a route. In order to choose a good route from the explored solution tree, it uses a heuristic $h(P)$. Whenever the current route is exhausted, the algorithm restarts to compute the next sub-route. However, it memorizes all visited edges.

We design $h(P)$ mainly based on the distances of the edges in P to the desired destination. This makes the heuristic widely applicable. In our formalization, later edges tend to contribute less to the cost since a parking spot likely is already found. Without the correct probabilities, we have to estimate this discount function $s(i)$. Our heuristic is given by

$$h(P) = \sum_{\substack{i=0: \\ \forall j<i:\ e_i \notin D(e_j)}}^{|P|} \frac{s(i)}{d(e_i)} \ .$$

We tried to improve the heuristic using commonly available road attributes, such as

Algorithm 2: Heuristic Search(*expands*)

```
1 $queue \leftarrow$ queue with empty route;
2 for $n = 1$ to $expands$ do
3     $route \leftarrow queue.pop()$;
4     foreach outgoing $edge$ from $route$ do
5         $queue.push(route$ concatenated with $edge)$;
6 return $P \in queue$ where $h(P)$ is largest;
```

information about nearby points of interest and the importance of a road, but none of them yielded significant performance improvements. Although there was some correlation to the probability, nearly all of it could be explained by a shared correlation to the road length. Correcting this by using the probability density as defined in section 5 yielded no significant correlations. Incorporating more information, such as demographic data, could potentially yield better heuristics, but this data may not be as widely available.

5 Dataset

Complementary to the specification of a theoretical model, we evaluate realistic scenarios on the central area of Berlin, Germany. In the following section, we describe this dataset and its collection process conducted by TomTom[2]. While some of the provided information could be extracted from public sources, we did not identify any party that made a comparable amount of on-street parking measures available or used such data for research on simulated parking scenarios. Our dataset will be made available at `https://hpi.de/friedrich/research/parking/`.

The dataset contains a road network reflecting the graph described in Section 3. Each edge contains data about the *nodes it leads from and to*, its *opposite edge*, the *parking probability*, and various other properties like the *length*, *functional road class*, and *average speed*. Unlike the other properties, the probabilities are broken up by hour of day and day of week. Since we determined the probability data empirically, they include a human factor. For example, drivers might have rejected a vacant parking spot because of small space or expensive parking fees. We interpret our experiments in Section 6 accordingly.

The raw data was collected by TomTom as floating-car data: Anonymized GPS positioning-information was analyzed and filtered to detect on-street parking events based on navigational destinations, driving speed, and behavior. The results were aggregated over a multitude of traces to find the number of parking searches and successes for each road segment. We further adjust the probabilities for each road, using the lower bound of the Agresti-Coull confidence interval [1] with a confidence level of 95 %. This results in pessimistic estimates of parking probabilities, based on the number of observations for each edge. Thus, we avoid unrealistic probabilities of 1.0 that might be observed in streets with too few detected parking searches for a meaningful result.

From the probability $p(e)$ and length $l(e)$ of a road e, we compute a *probability density* $1 - \big(1 - p(e)\big)^{\frac{1}{l(e)}}$ that resembles the probability of a unit-length part of the road. The collected probability mass stays the same, no matter whether we observe the whole road at

[2] TomTom Development Germany GmbH, An den Treptowers 1, 12435 Berlin, Germany.

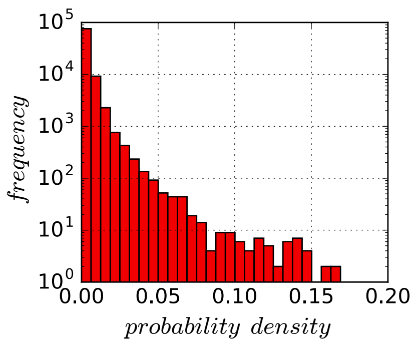
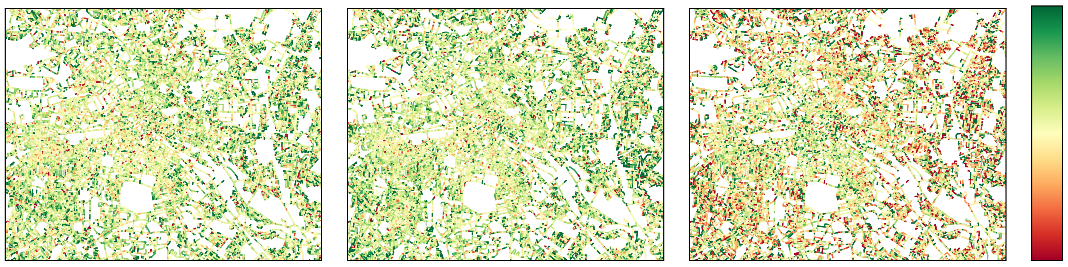


Figure 3 The *probability density* is a measure for the parking probability per meter. For our dataset, we assume an underlying Pareto II distribution. (Mon–Sun, 9 a.m.–4 p.m.)



(a) Morning, 6 a.m. to 2 p.m. **(b)** Evening, 2 p.m. to 9 p.m. **(c)** Night, 9 p.m. to 6 a.m.

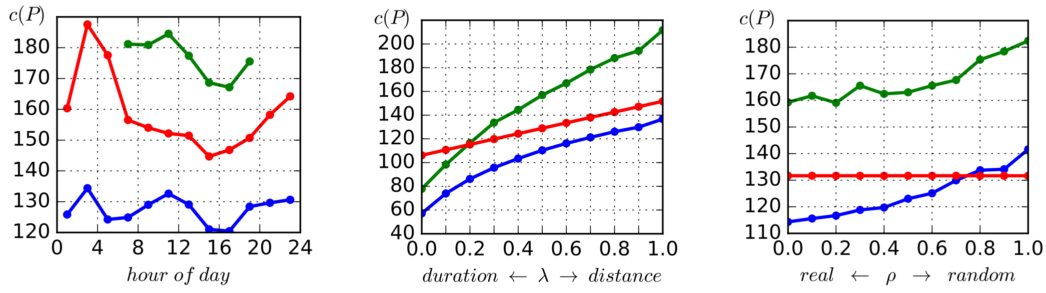
Figure 4 We compute the probability densities of all roads at different times of day. The three heat maps show differences to the per-street averages over the whole day. Containing 90 % of values, we map the interval $(-0.003, +0.003)$ from red (worse than average) to green (better than average). One can see that parking becomes harder at night, especially in residential areas, which can also be observed in our experiments in Section 6. (All data for Mon–Sun; best viewed in color.)

once or each unit length separately. This is necessary, because the probability correlates strongly with the length of the road, because long roads can simply fit more parking spots. Therefore, we can counteract arbitrary splitting of roads into segments in the map by using densities.

We found the probability mass to be Pareto-distributed as shown in Figure 3. The fitted distribution with parameters $\sigma = 1.3445$, $\mu = 0.0000$, and $\xi = 0.0022$ is the basis for our experiments on inaccurate probabilities in Section 6. We further compare the probability density at different time spans with the density averaged across the whole day in Figure 4. Between weekdays and weekends, we did not find a prominent difference. We also tried to reconstruct the probabilities from other attributes of that road, using a statistical model, which did not yield meaningful results. This, however, emphasizes the uniqueness of the released dataset.

6 Experiments

We conduct three experiments that we will describe and interpret in this section. In all of them, we compare our **Branch and Bound**, our **Heuristic Search**, and the **G2** algorithm proposed by Jossé et al. [5], which greedily chooses the next edge that maximizes the probability per cost. We do not consider computation time in our evaluation because all three



(a) Average algorithm costs throughout the day. We only show data points where the algorithm finds a successful route in at least 95 % of all v_0.

(b) Average algorithm costs at different weightings of driving duration and walking distance. Cost is doubled to match the other experiments at $\lambda = 0.5$.

(c) Average algorithm costs when operating on probability data that is disturbed to varying levels. Our heuristic search algorithm is unaffected.

Figure 5 Algorithm performance in the three experiments we conducted. Averaged over 1000 sampled start nodes v_0. From bottom to top: **Branch and Bound**, **Heuristic Search**, and **G2** by Jossé et al. [5].

algorithms take less than one second to compute. A query time in this order of magnitude does not affect applicability in practice.

In contrast to the problem formalization in Section 3, we weight the two cost terms slightly differently to make interpretation easier. While in the formalization, *driving duration* and *walking distance* are weighted with factors λ and $1 - \lambda$, we first divide the walking distance by a typical walking speed of $1.4\,\frac{\text{m}}{\text{s}}$ and then add up both measures. Therefore, $c(P)$ refers to the total duration for finding a parking spot in this section.

In all the following experiments, we choose probability mass threshold $\varepsilon = 0.05$. Thus a route is considered successful if it leads along a vacant parking spot with a probability of at least 95 %. We sample 1000 start nodes v_0, proportional to the amount of parking searches in the time bin as of our dataset. If not stated otherwise the time bin we use for the simulation is Monday to Friday from 9 a.m. to 5 p.m.

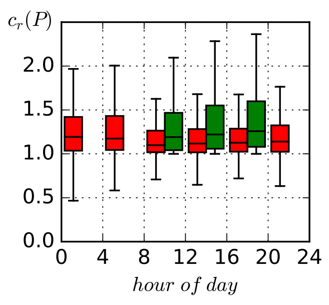
For **Branch and Bound**, the *expands* parameter is 10000 while for **Heuristic Search** it is 1000. For both algorithms, we did not find a significant improvement in cost above those values. In other applications, we suggest to experimentally determine the number of expands as they may vary based on the road network. Further, we use a fall off function of $s(i) = 1 - \frac{i}{20}$ for **Heuristic Search** that we determined through parameter search. **G2** does not have any free parameters.

To prevent infinite loops when algorithms drive through cycles of streets with no probabilities, we limited the route length to 100 edges. We do not include routes that exceed this length. If an algorithm does not reach the probability mass threshold ε in at least 95 % of all sampled starting positions v_0 we declare it as failing in a certain scenario.

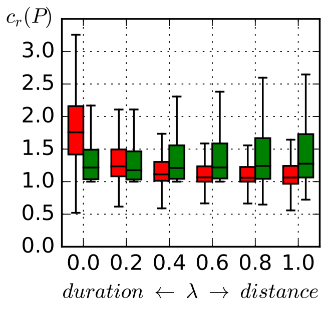
The box plots in Figure 6 provide statistical information for the costs of the algorithms used in the line diagrams in Figure 5. The whiskers limit 95 % of all values while the box represents the first and the third quartiles with the central black line showing the median.

6.1 Time of Day

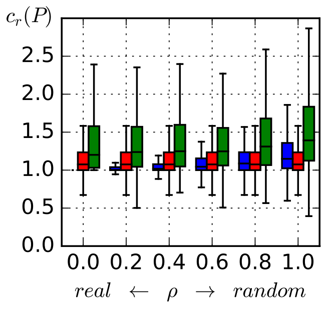
In 5a we compare the three algorithms against each other in cost throughout the day. To avoid the aforementioned infinite loops, we merged the data into two-hour blocks. Nevertheless,



(a) Algorithm costs throughout the day, relative to the baseline. We only show boxes where the algorithm finds a successful route in at least 95 % of all v_0.



(b) Algorithm cost at different weightings of driving duration and walking distance, relative to the baseline.



(c) Algorithm cost using probabilities with different levels of accuracy, relative to the baseline at $\rho = 0.0$.

Figure 6 Relative algorithm performance $c_r(P)$ divided by the **Branch and Bound** baseline. The respective baseline is represented by just a line. Boxes from left to right: **Branch and Bound**, **Heuristic Search**, and **G2** by Jossé et al. [5].

G2 achieved ε only during rush hours in more than 95 % of the runs. **Heuristic Search** and **Branch and Bound** both do experience decreasing costs in the early morning and the afternoon. Besides that, **Branch and Bound** is relatively consistent while **Heuristic Search** has a significant peak between 2 and 4 a.m., being 30 % more expensive than at its optimal time. Over the whole day, the baseline stays ahead of our heuristic approach by a factor of 1.3 on average.

6.2 Cost Weighting

Until now, we weighted all our cost measures equally. Because walking speeds and user preferences may differ, our cost weighting is not universal. To investigate the influence of the weighting parameters on the algorithms, we ran our experiments with a range of different weightings. For each cost weighting $\lambda \in [0, 1]$, we simulated the algorithms with the same starting points. The new weighted cost is defined as $c(P) = 2\big(\lambda \cdot \mathit{distance} + (1 - \lambda)\mathit{duration}\big)$. It is important to note that the algorithms *did* have access to this cost function.

We see that **Heuristic Search** performs well compared to the other algorithms when distance has a larger weight, because its heuristic relies heavily on the distance. All algorithms perform better when duration is weighted high, since it is faster to drive to segments with high probabilities than walking back from them.

6.3 Impact of Inaccurate Probabilities

We can assume that, despite a high number of parking observations, computed success probabilities do not fully correspond to real-life circumstances. Hence, we need to assess the impact of inaccurate probability values. We do this by applying an interpolated noise model. When simulating noise, all algorithms work with gradually mutated probabilities while their results continue to be evaluated on the original setting with ground-truth data provided by TomTom. This means, that an algorithm requesting the costs of its next step would obtain a noisy response and act potentially less accurate. A probability-agnostic algorithm, however, would always perform the same, no matter which noise setting. A noise parameter ρ from the interval $[0, 1]$ determines the level of accuracy, with 0 implying unaltered probabilities and 1 fully applied noise. We conducted simulations on various noise distributions, but here we focus on an interpolated Pareto II distribution. For this model, we take random samples from a Pareto distribution, fitted on the probability densities of all roads on the map.

Figure 5c compares the costs of our algorithms on noise levels within the interval of $[0, 1]$. As expected, the performance of all algorithms decreases with a rising noise level. Notably, Branch and Bound turns out to be particularly stable against noise and outperforms the other algorithms on all levels. A higher accuracy of the collected probabilities results in a lower cost. Furthermore, we see that until an assumed noise of 0.7, knowledge of the probabilities means an advantage over the heuristic approaches.

7 Conclusion

We proposed two algorithmic approaches for solving the NP-complete problem of parking search. Our **Branch and Bound** algorithm finds the best route within all routes up to a certain length as measured by the cost. For the scenario where no probability data is available, we proposed our **Heuristic Search** algorithm that on average comes close to the baseline by a factor of 1.3 in cost. This leads to the conclusion that per-street probabilities of parking successes are beneficial but not strictly necessary to compute good parking routes. Also, it might be possible to reduce the effort of collecting probability data, since less accurate values still yield attractive results.

We evaluated our algorithms on real per-street probability data for the Berlin area. We released this dataset along with the paper to allow the community to evaluate further parking search algorithms on real data. However, it is unclear how our heuristics would perform in *other cities*. Building on the presented work, two directions of further research sound promising to us. First, we assumed that the probability of a parking spot stays 0 after visiting it once. It is natural to assume that the observed information decays over time. However, including recovery adds extra assumptions to the model. A *dataset of per-street recovery functions*, e.g., as modeled in [5], would be very interesting in order to explore the impact of recovery in a realistic model. Second, a limitation of our model is to assume independence between the probabilities of nearby roads. Again, real data on this is not available. We conclude that collecting and releasing additional metrics would open the doors to additional research in the field of parking search.

Acknowledgments. We thank TomTom Development Germany GmbH, An den Treptowers 1, 12435 Berlin and its engineering team including our contact persons Steffen Wiesner and Peter Mieth. During our work, we received valuable feedback and support from them. They provided us with the collected probability data and additional road attributes. We are grateful for the permission to release the dataset used in this paper to the public.

References

1 Alan Agresti and Brent A. Coull. Approximate is better than "exact" for interval estimation of binomial proportions. *The American Statistician*, 52(2):119–126, 1998. URL: `http://www.jstor.org/stable/2685469`.

2 C Richard Cassady and John E Kobza. A probabilistic approach to evaluate strategies for selecting a parking space. *Transportation Science*, 32(1):30–42, 1998.

3 Simon Evenepoel, Jan Van Ooteghem, Sofie Verbrugge, Didier Colle, and Mario Pickavet. On-street smart parking networks at a fraction of their cost: performance analy-

sis of a sampling approach. *Transactions on Emerging Telecommunications Technologies*, 25(1):136–149, 2014.

4 Ming Hua and Jian Pei. Probabilistic path queries in road networks: Traffic uncertainty aware path selection. In *Proceedings of the 13th International Conference on Extending Database Technology*, pages 347–358, 2010. `doi:10.1145/1739041.1739084`.

5 Gregor Jossé, Klaus Arthur Schmid, and Matthias Schubert. Probabilistic resource route queries with reappearance. In *Proceedings of the 18th International Conference on Extending Database Technology*, pages 445–456, 2015. `doi:10.5441/002/edbt.2015.39`.

6 Yaron Kanza, Eliyahu Safra, and Yehoshua Sagiv. Route search over probabilistic geospatial data. In *Proceedings of the 11th International Symposium on Advances in Spatial and Temporal Databases*, pages 153–170, 2009. `doi:10.1007/978-3-642-02982-0_12`.

7 Eliyahu Safra, Yaron Kanza, Nir Dolev, Yehoshua Sagiv, and Yerach Doytsher. Computing a k-route over uncertain geographical data. In *Proceedings of the 10th International Conference on Advances in Spatial and Temporal Databases*, pages 276–293, 2007. URL: `http://dl.acm.org/citation.cfm?id=1784462.1784478`.

8 Donald Shoup. *The High Cost of Free Parking*. APA Planners Press, 2005.

Scalable Exact Visualization of Isocontours in Road Networks via Minimum-Link Paths[*]

Moritz Baum[1], Thomas Bläsius[2], Andreas Gemsa[3], Ignaz Rutter[4], and Franziska Wegner[5]

1 **Karlsruhe Institute of Technology, Karlsruhe, Germany**
`moritz.baum@kit.edu`
2 **Hasso Plattner Institute, Potsdam, Germany**
`thomas.blaesius@hpi.de`
3 **Karlsruhe Institute of Technology, Karlsruhe, Germany**
`andreas.gemsa@kit.edu`
4 **Karlsruhe Institute of Technology, Karlsruhe, Germany**
`ignaz.rutter@kit.edu`
5 **Karlsruhe Institute of Technology, Karlsruhe, Germany**
`franziska.wegner@kit.edu`

Abstract

Isocontours in road networks represent the area that is reachable from a source within a given resource limit. We study the problem of computing accurate isocontours in realistic, large-scale networks. We propose isocontours represented by polygons with minimum number of segments that separate reachable and unreachable components of the network. Since the resulting problem is not known to be solvable in polynomial time, we introduce several heuristics that run in (almost) linear time and are simple enough to be implemented in practice. A key ingredient is a new practical linear-time algorithm for minimum-link paths in simple polygons. Experiments in a challenging realistic setting show excellent performance of our algorithms in practice, computing near-optimal solutions in a few milliseconds on average, even for long ranges.

1998 ACM Subject Classification G.2.2 Graph Theory, G.2.3 Applications, I.3.5 Computational Geometry and Object Modeling

Keywords and phrases isocontours, separating polygons, minimum-link paths

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.7

1 Introduction

How far can I drive my battery electric vehicle (EV), given my position and the current state of charge? – This question can be answered by a map visualizing the reachable region. This region is bounded by *isocontours* representing points that require the same amount of energy to be reached. Isocontours are typically considered in the context of functions $f\colon \mathbb{R}^2 \to \mathbb{R}$, in our case describing the energy necessary to reach a point in the plane. However, f is defined only at certain points, namely vertices of the graph representing the road network. We have to fill the gaps by deciding how an isocontour should pass through regions between roads. The fact that the quality of the resulting visualization heavily depends on these decisions makes

[*] This work was partially supported by the EU FP7/2007-2013 under grant agreement no. 609026 (project MOVESMART) and the Helmholtz Program Storage and Cross-linked Infrastructures, Topic 6 Superconductivity, Networks and System Integration.

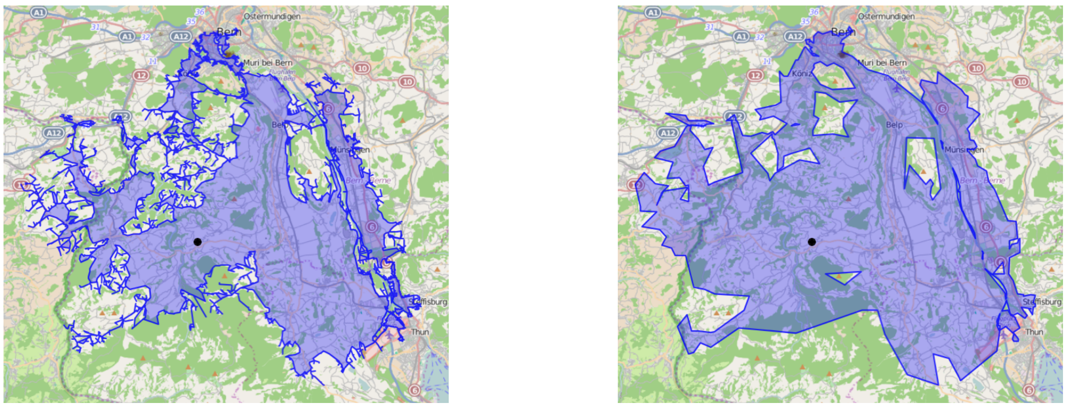


Figure 1 Isocontours in a mountainous area (near Bern, Switzerland), showing the range of an EV positioned at the black disk with a state of charge of 2 kWh. Note that the polygons contain holes, due to unreachable high-ground areas. Left: state-of-the-art approach (over 10 000 segments, computed by `RP-RC` from Section 4); right: our approach (416 segments, computed by `RP-CU`).

computing isocontours in road networks an interesting algorithmic problem. Besides range visualization for EVs, isocontour visualization is relevant in a wide range of applications, including reachability analyses in urban planning [2, 20, 22, 25], geomarketing [10], and environmental and social sciences [20].

Several techniques consider the problem of computing the subnetwork that is reachable within a given timespan (but not the actual isocontour), enabling query times in the order of milliseconds [4, 11, 12]. O'Sullivan et al. [25] introduce basic approaches for isocontour visualization based on merging shapes covering the reachable area. Marciuska and Gamper [22] propose isocontours induced by reachable points in the network, but their approaches are too slow for interactive applications (several seconds for small and medium ranges). In contrast, our work is motivated by more challenging scenarios, e. g., visualizing the range of high-end Tesla models or the area reachable by a truck driver within a day of work. Our algorithms are guided by three major objectives: Isocontours must be *exact*, i. e., correctly separate the reachable subgraph from the remaining unreachable part; they should be polygons of low *complexity*, i. e., consist of few segments (enabling fast rendering and a clear, uncluttered visualization); algorithms should be fast enough for interactive applications, even on large inputs. Figure 1 compares an example resembling state-of-the-art techniques [9, 10, 12, 22] to one of our approaches. The original works also consider inexact variants of the approach we refer to as state-of-the-art (e. g., by omitting holes or degeneracies from the polygon). We resort to an exact variant, in accordance with our objectives.

Contribution and Outline. We propose several new algorithms for computing polygons that represent isocontours in road networks. All approaches compute exact isocontours, while having low complexity. Their efficient performance is both proven in theory and demonstrated in practice on large, realistic instances. Section 2 states the precise problem and outlines our algorithmic approach. Section 3 attacks the important subproblem of separating the boundaries of a hole-free region by a polygon with minimum number of segments. While it can be solved in $O(n \log n)$ time [28], we propose a simpler algorithm that uses at most two additional segments, runs in linear time, and requires computation of only a single minimum-link path. We also propose a minimum-link path algorithm that is simpler than previous approaches [26]. Section 4 extends these results to the general case, where unreachable parts of the network can induce holes between the boundaries to be separated. As the complexity of this problem is unknown, we focus on efficient heuristic approaches that work well in practice, but do not give (nontrivial) guarantees on the complexity of the resulting range polygons. Section 5 contains our extensive experimental evaluation. It demonstrates that all approaches are fast enough even for use in interactive applications. Section 6 concludes with final remarks. See the full version for omitted details and proofs [3].

2 Problem Statement and General Approach

Let $G = (V, E)$ be a road network, which we consider as a geometric graph where vertices have a fixed position in the plane and edges are straight-line segments between their endpoints. The function $\mathrm{cons}\colon E \to \mathbb{R}$ assigns resource consumption to all edges. A source $s \in V$ and range $r \in \mathbb{R}_{\geq 0}$ partition the graph into two parts, one that is within range r from s, and the part that is not. A vertex v is *reachable* if the resource consumption $\mathrm{cons}(\pi_v)$ on the shortest (wrt. a nonnegative length function on the edges) s–v-path π_v is at most r. An edge (u, v) is *passable* if it can be traversed in at least one direction, i. e., $\mathrm{cons}(\pi_u) + \mathrm{cons}((u, v)) \leq r$ or $\mathrm{cons}(\pi_v) + \mathrm{cons}((v, u)) \leq r$. Let V_r be the set of reachable vertices and E_r the set of passable edges. The *reachable subgraph* is $G_r = (V_r, E_r)$. Let $V_u = V \setminus V_r$ be the set of *unreachable vertices* and E_u the set of *unreachable edges* for which both endpoints are unreachable. The *unreachable subgraph* is $G_u = (V_u, E_u)$. Edges in $E \setminus (E_r \cup E_u)$ are called *boundary edges*. A *range polygon* is a plane (not necessarily simple) polygon P separating G_r and G_u, such that its interior contains G_r and has empty intersection with G_u. Note that if G is not planar, a range polygon may not even exist: If a passable edge intersects an unreachable edge, the requirements of including the passable and excluding the unreachable edge obviously contradict. To resolve this issue, we consider the planarization G_p of G, which is obtained from G by considering each intersection point p as a *dummy vertex* that subdivides all edges of G that contain p. A dummy vertex is reachable if and only if it subdivides a passable edge of the original graph. An edge in G_p is passable if and only if the edge in G containing it is passable. As before, an edge of G_p is unreachable if both endpoints are unreachable. Finally, let the graph G' consist of the union of the reachable and unreachable subgraph of G_p. A face of G' incident to both the reachable and unreachable subgraph is a *border region*.

Given a source $s \in V$ and a range $r \in \mathbb{R}_{\geq 0}$, we seek to compute a range polygon wrt. G_p that has the minimum number of holes, and among these we seek to minimize the complexity of the range polygon. This can be achieved as follows.

1. Compute the reachable and unreachable subgraph of G.
2. Planarize G, compute the reachable and unreachable subgraph of its planarization G_p.
3. Compute the border regions.
4. For each border region B, compute a simple polygon of minimum complexity contained in B that separates the unreachable components incident to B from the reachable component.

Step 1 is solved by a variant of Dijkstra's algorithm [8]. Tailored preprocessing-based algorithms for road networks exist [4, 11]. For Step 2, we planarize G during preprocessing in a single run of the well-known sweep line algorithm [6] to obtain G_p. In a query (i. e., for given $s \in V$ and $r \in \mathbb{R}_{\geq 0}$), reachability of dummy vertices is then determined in a linear scan of all original edges containing a dummy vertex. This produces limited overhead in practice, since the number of dummy vertices in graphs representing road networks is typically small (as large parts of the input are already planar). Border regions are extracted in Step 3 by traversing faces of G_p that contain at least one boundary edge. In the remainder of this work, we focus on Step 4. Each connected component of the boundary of a border region

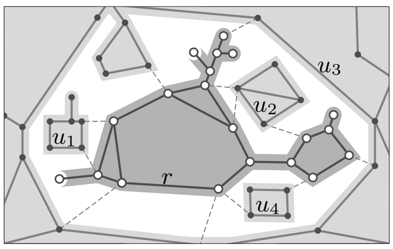


Figure 2 Border region (white), with the reachable boundary $R = \{r\}$ and the unreachable boundary with components $U = \{u_1, u_2, u_3, u_4\}$. Reachable and unreachable parts are shaded.

is a hole-free non-crossing polygon. Note that these polygons are not necessarily simple in the sense that they may contain the same segment twice in different directions; see Figure 2. Each border region is defined by two sets R and U of such polygons, where R contains the reachable components and U contains the unreachable components. We seek a simple polygon with minimum number of segments that separates U from R. Guibas et al. [17] showed that this problem is NP-complete in general. In our case, however, it is $|R| = 1$ since the reachable subgraph is, by definition, connected. Guibas et al. left this case as an open problem and, to the best of our knowledge, it has not been resolved.

3 Range Polygons in Border Regions Without Holes

In this section, we consider the special case of a border region B with $|R| = |U| = 1$. A polygon of minimum complexity that separates the two polygons can be found in $O(n \log n)$ time [28]. However, the algorithm is rather involved and requires computation of several minimum-link paths. We propose a simpler algorithm that uses at most two additional segments, runs in linear time, and computes a single minimum-link path. It adds an edge e to B that connects both boundaries. In the resulting polygon B', it computes a minimum-link path π that connects the two sides of e. The algorithm of Suri [26] finds such a path π in linear time. We obtain a separating polygon by connecting the endpoints of π along e.

We address the subproblem of computing a *minimum-link path* between two edges a and b of a simple polygon P, i. e., a polygonal path with minimum number of segments that connects a and b and lies in the interior of P. The algorithm of Suri [26] starts by triangulating the input polygon. We preprocess this step by triangulating all faces of the planarized input graph only once. Afterwards, in each step of Suri's algorithm a *window* (which we define in a moment) is computed. To this end, several visibilty polygons are constructed. This suffices to prove linear running time, but seems wasteful from a practical point of view. Below, we present a simpler linear-time algorithm for computing the windows, called FMLP (*fast minimum-link path*). It can be seen as a generalization of an algorithm for approximating piecewise linear functions [19].

Windows and Visibility. Let T be the graph obtained by arbitrarily triangulating P. Let t_a and t_b be the triangles incident to a and b, respectively. As T is an outerplanar graph, its (weak) dual graph has a unique path $t_a = t_1, t_2, \dots, t_{k-1}, t_k = t_b$ from t_a to t_b; see Figure 3a. We call the triangles on this path *important* and their position in the path their *index*. The *visibility polygon* $V(a)$ of the edge a in P is the polygon that contains a point p in its interior if and only if there is a point q on a such that the line segment pq lies inside P. Let i be the highest index such that the intersection of the triangle t_i with the visibility polygon $V(a)$ is

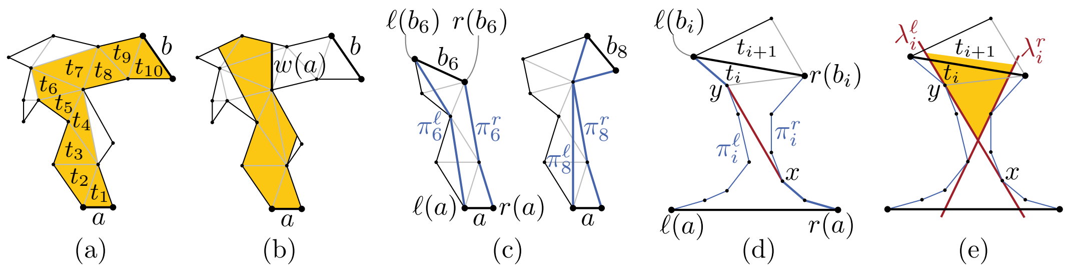


Figure 3 (a) Important triangles wrt. a and b. (b) The window $w(a)$ is an edge of the (shaded) visibility polygon. (c) The left and right shortest paths (blue) intersect for $i = 8$ but not for $i = 6$. (d) The shortest path from $r(a)$ to $\ell(b_i)$ contains the bold prefix of π_i^r, the red segment, and the bold suffix of π_i^ℓ. (e) Visibility lines spanning the (shaded) visibility cone.

not empty. The *window* $w(a)$ is the edge of $V(a)$ that intersects t_i closest (wrt. minimum Euclidean distance) to the edge between t_i and t_{i+1}; see Figure 3b. Note that $w(a)$ separates the polygon P into two parts. Let P' be the part containing the edge b that we want to reach. A minimum-link path from a to b in P can then be obtained by adding an edge from a to $w(a)$ to a minimum-link path from $w(a)$ to b in P'. Thus, the next window is computed in P' starting with the previous window $w(a)$. Below, we first describe how to compute the first window and then discuss what has to be changed to compute the subsequent windows.

Let T_i be the subgraph of T induced by the triangles $t_1, \dots, t_i$ and let P_i be the polygon bounding the outer face of T_i. The polygon P_i has two special edges, namely a and the edge shared by t_i and t_{i+1}, which we call b_i. Let $\ell(a)$ and $r(a)$, and $\ell(b_i)$ and $r(b_i)$ be the endpoints of a and b_i, respectively, such that their clockwise order is $r(a)$, $\ell(a)$, $\ell(b_i)$, $r(b_i)$ (think of $\ell(\cdot)$ and $r(\cdot)$ being the left and right endpoints, respectively); see Figure 3c. We define the *left shortest path* π_i^ℓ to be the shortest polygonal path (shortest in terms of Euclidean length) that connects $\ell(a)$ with $\ell(b_i)$ and lies inside or on the boundary of P_i. The *right shortest path* π_i^r is defined analogously for $r(a)$ and $r(b_i)$; see Figure 3c.

Assume that the edge b_i is visible from a, i.e., there exists a line segment in the interior of P_i that starts at a and ends at b_i. Such a visibility line separates the polygon into a left and a right part. Observe that it follows from the triangle inequality that the left shortest path π_i^ℓ and the right shortest path π_i^r lie inside the left and right part, respectively. Thus, these two paths do not intersect. Moreover, the two shortest paths are *outward convex* in the sense that the left shortest path π_i^ℓ has only left bends when traversing it from $\ell(a)$ to $\ell(b_i)$ (the symmetric property holds for π_i^r); see the case $i = 6$ in Figure 3c. We note that the outward convex paths are sometimes also called “inward convex” and the polygon consisting of the two outward convex paths together with the edges a and b_i is also called *hourglass* [15]. The following lemma, which is similar to a statement shown by Guibas et al. [16, Lemma 3.1], summarizes the above observation.

▶ **Lemma 1.** *If the triangle t_i is visible from a, then the left and right shortest path in P_{i-1} have empty intersection. Moreover, if these paths do not intersect, they are outward convex.*

Guibas et al. [16] argue that the converse of the first statement is also true, i.e., if the two paths have empty intersection, then the triangle t_{i+1} is visible from a. Their main arguments go as follows. The shortest path (wrt. Euclidean length) in the hourglass that connects $r(a)$ with $\ell(b_i)$ is the concatenation of a prefix of π_i^r, a line segment from a vertex x of π_i^r to a vertex y of π_i^ℓ, and a suffix of π_i^ℓ; see Figure 3d. We call the straight line through x and y the *left visibility line* and denote it by λ_i^ℓ. We assume λ_i^ℓ to be oriented from x to y and call

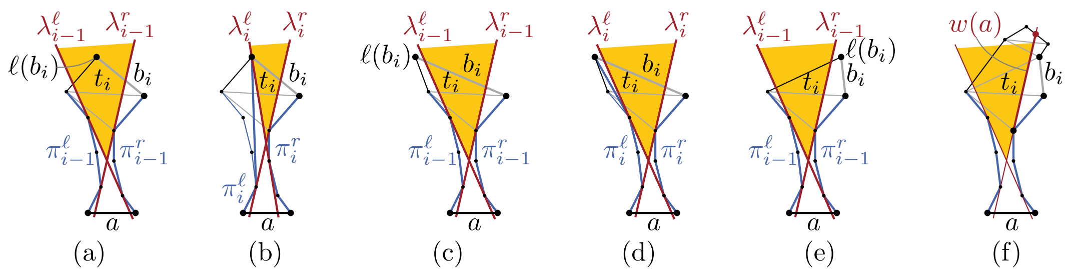

λ_{i-1}^{ℓ} λ_{i-1}^{r} $\ell(b_i)$ t_i b_i π_{i-1}^{ℓ} π_{i-1}^{r} a (a)
λ_i^{ℓ} λ_i^{r} t_i b_i π_i^{r} π_i^{ℓ} a (b)
λ_{i-1}^{ℓ} λ_{i-1}^{r} $\ell(b_i)$ b_i t_i π_{i-1}^{ℓ} π_{i-1}^{r} a (c)
λ_i^{ℓ} λ_i^{r} b_i t_i π_i^{ℓ} π_i^{r} a (d)
λ_{i-1}^{ℓ} λ_{i-1}^{r} $\ell(b_i)$ b_i t_i π_{i-1}^{ℓ} π_{i-1}^{r} a (e)
$w(a)$ b_i t_i a (f)

Figure 4 (a) The new vertex $\ell(b_i)$ lies in the visibility cone. (b) The updated left shortest path π_i^{ℓ} and left visibility line λ_i^{ℓ}. (c) The vertex $\ell(b_i)$ lies to the left of λ_{i-1}^{ℓ}. (d) The left shortest path has to be updated, the left visibility line remains unchanged. (e) The vertex $\ell(b_i)$ lies to the right of λ_{i-1}^{r}, i.e., t_{i+1} is not visible from a. (f) The window $w(a)$ is a segment of λ_{i-1}^{r}.

x and y the *source* and *target* of λ_i^{ℓ}. Analogously, one can define the *right visibility line* λ_i^{r}; see Figure 3e. We call the intersection of the half-plane to the right of λ_i^{ℓ} with the half-plane to the left of λ_i^{r} the *visibility cone*. It follows that the intersection of the visibility cone with the edge b_i is not empty and a point on the edge b_i is visible from a if and only if it lies in this intersection [16]. This directly extends to the following lemma.

▶ **Lemma 2.** *If the left and right shortest path in P_{i-1} have empty intersection, t_i is visible from a. Moreover, a point in t_i is visible from a if and only if it lies in the visibility cone.*

The above observations then justify the following approach for computing the window. We iteratively increase i until the left and the right shortest path of the polygon P_i intersect. We then know that the triangle t_{i+1} is no longer visible; see Lemma 1. Moreover, as the left and the right shortest path did not intersect in P_{i-1}, the triangle t_i is visible from a; see Lemma 2. To find the window, it remains to find the edge of the visibility polygon $V(a)$ that intersects t_i closest to the edge between t_i and t_{i+1}. Thus, by the second statement of Lemma 2, the window must be a segment of one of the two visibility lines. It remains to fill out the details of this algorithm, argue that it runs in overall linear time, and describe what has to be done in later steps, when we start at a window instead of an edge.

Computing the First Window. We start with the details of the algorithm starting from an edge. Assume the triangle t_i is still visible from a, i.e., π_{i-1}^{ℓ} and π_{i-1}^{r} do not intersect. Assume further that we have computed the left and right shortest path π_{i-1}^{ℓ} and π_{i-1}^{r} as well as the corresponding visibility lines λ_{i-1}^{ℓ} and λ_{i-1}^{r} in a previous step. Assume without loss of generality that the three corners of the triangle t_i are $\ell(b_{i-1})$, $\ell(b_i)$, and $r(b_i) = r(b_{i-1})$. There are three possibilities shown in Figure 4, i.e., the new vertex $\ell(b_i)$ lies either in the visibility cone spanned by λ_{i-1}^{ℓ} and λ_{i-1}^{r} (Figure 4a), to the left of the left visibility line λ_{i-1}^{ℓ} (Figure 4c), or to the right of the right visibility line λ_{i-1}^{r} (Figure 4e).

By Lemma 2, a point in t_i is visible from a if and only if it lies inside the visibility cone. Thus, the edge b_i between t_i and t_{i+1} is no longer visible if and only if the new vertex $\ell(b_i)$ lies to the right of λ_{i-1}^{r}; see Figure 4e. In this case, we can stop and the desired window $w(a)$ is the segment of λ_{i-1}^{r} starting at its touching point with π_{i-1}^{r} and ending at its first intersection with an edge of P; see Figure 4f.

In the other two cases (Figure 4a and Figure 4c), we have to compute the new left and right shortest path π_i^{ℓ} and π_i^{r} and the new visibility lines λ_i^{ℓ} and λ_i^{r} (Figure 4b and Figure 4d). Note that the old and new right shortest path π_{i-1}^{r} and π_i^{r} connect the same endpoints $r(a)$ and $r(b_{i-1}) = r(b_i)$. As the path cannot become shorter by going through the new triangle t_i,

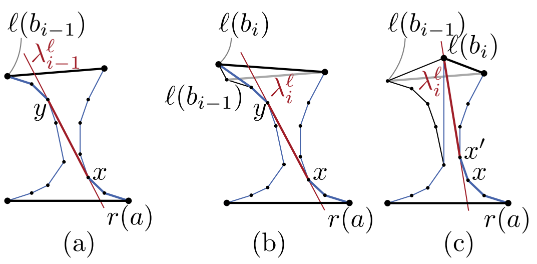


Figure 5 (a) The shortest path from $r(a)$ to $\ell(b_{i-1})$ (bold) defining the left visibility line λ^ℓ_{i-1}. (b) The visibility line does not change if $\ell(b_i)$ lies to the left of λ^ℓ_{i-1}. (c) Illustration how the visibility line changes when $\ell(b_i)$ lies to the right of λ^ℓ_{i-1}.

we have $\pi^r_i = \pi^r_{i-1}$. The same argument shows that $\lambda^r_i = \lambda^r_{i-1}$ (recall that the visibility lines were defined using a shortest path from $\ell(a)$ to $r(b_{i-1}) = r(b_i)$).

We compute the new left shortest path π^ℓ_i as follows. Let x be the latest vertex on π^ℓ_{i-1} such that the prefix of π^ℓ_{i-1} ending at x concatenated with the segment from x to $\ell(b_i)$ is outward convex. We claim that π^ℓ_i is the path obtained by this concatenation, i.e., this path lies inside P_i and there is no shorter path lying inside P_i. It follows by the outward convexity, that there cannot be a shorter path inside P_i from $\ell(a)$ to $\ell(b_i)$. Moreover, by the assumption that π^ℓ_{i-1} was the correct left shortest path in P_{i-1}, the subpath from $\ell(a)$ to x lies inside P_i. Assume for contradiction that the new segment from x to $\ell(b_i)$ does not lie entirely inside P_i. Then it has to intersect the right shortest path and it follows that the right shortest path and the correct left shortest path have non-empty intersection, which is not true by Lemma 1.

To get the new left visibility line λ^ℓ_i, we have to consider the shortest path in P_i that connects $r(a)$ with $\ell(b_i)$. Let x and y be the source and target of λ^ℓ_{i-1}, respectively, i.e., the shortest path from $r(a)$ to $\ell(b_{i-1})$ is as shown in Figure 5a. If the new vertex $\ell(b_i)$ lies to the left of λ^ℓ_{i-1} (Figure 5b), then the shortest path from $r(a)$ to $\ell(b_i)$ also includes the segment from x to y. Thus, $\lambda^\ell_i = \lambda^\ell_{i-1}$ holds in this case. Assume the new vertex $\ell(b_i)$ lies to the right of λ^ℓ_{i-1} (Figure 5c). Let x' be the latest vertex on the path π^r_i such that the concatenation of the subpath from $r(a)$ to x' with the segment from x' to the new vertex $\ell(b_i)$ is outward convex in the sense that it has only right bends; see Figure 5c. We claim that this path lies inside P_i and that there is no shorter path inside P_i. Moreover, we claim that x' is either a successor of x in π^r_{i-1} or $x' = x$. Clearly, the concatenation of the path from $r(a)$ to x with the segment from x to $\ell(b_i)$ is outward convex, thus the latter claim follows. It follows that the segment from x' to $\ell(b_i)$ lies to the right of the old visibility line λ^ℓ_{i-1}. Thus, it cannot intersect the path π^ℓ_i (except in its endpoint $\ell(b_i)$), as π^ℓ_{i-1} lies to the left of λ^ℓ_{i-1}. Moreover, as we chose x' to be the last vertex on π^r_{i-1} with the above property, this new segment does not intersect π^r_i (except in x'). Hence, the segment from x' to $\ell(b_i)$ lies inside P_i. As before, it follows from the convexity that there is no shorter path inside P_i. Thus, λ^ℓ_i is the line through x' and $\ell(b_i)$ (x' is the new source and $\ell(b_i)$ is the new target).

▶ **Lemma 3.** *Let t_h be the triangle with the highest index that is visible from a. Then, the algorithm FMLP computes the first window $w(a)$ in $O(h)$ time.*

Computation of Subsequent Windows. As mentioned before, the first window $w(a)$ we compute separates P into two smaller polygons. Let P' be the part including the edge b (and not a). In the following, we denote $w(a)$ by a'. To get the next window $w(a')$, we have to apply the above procedure to P' starting with a'. However, this would require to partially

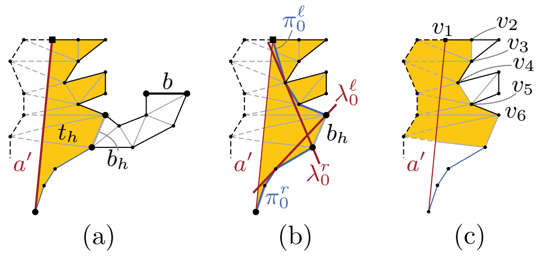


Figure 6 (a) Polygon P' after computing the window a'; P'_0 is shaded. (b) Shortest paths π_0^ℓ and π_0^r (blue) and visibility lines (red). (c) Sequence $v_1, \dots, v_6$, triangles intersected by a' are shaded.

retriangulate the polygon P'. More precisely, let t_h be the triangle with the highest index that is visible from a and let b_h be the edge between t_h and t_{h+1}; see Figure 6a. Then b_h separates P' into an initial part P'_0 (the shaded part in Figure 6a) and the rest (having b on its boundary). The latter part is properly triangulated, however, the initial part P'_0 is not. The conceptually simplest solution is to retriangulate P'_0. However, this would require an efficient subroutine for triangulation (and dynamic data structures that allow us to update P and T, which produces overhead in practice). Instead, we propose a much simpler method for computing the next window. The general idea is to compute the shortest paths in P'_0 from $\ell(a')$ to $\ell(b_h)$ and from $r(a')$ to $r(b_h)$; see Figure 6b. We denote these paths by π_0^ℓ and π_0^r, respectively. Moreover, we want to compute the corresponding visibility lines λ_0^ℓ and λ_0^r. Afterwards, we can continue with the correctly triangulated part as before.

Concerning the shortest paths, note that the right shortest path π_0^r is a suffix of the previous right shortest path, which we already know. For the left shortest path π_0^ℓ, first consider the polygon induced by the triangles intersected by a'; see Figure 6c. Let $v_1, \dots, v_g$ be the path on the outer face of this polygon (in clockwise direction) from $\ell(a') = v_1$ to $\ell(b_h) = v_g$. We obtain π_0^ℓ using *Graham's scan* [14] on the sequence $v_1, \dots, v_g$, i. e., starting with an empty path, we iteratively append the next vertex of the sequence $v_1, \dots, v_g$ while maintaining the path's outward convexity by successively removing the second to last vertex, if necessary. It remains to compute the visibility lines λ_0^ℓ and λ_0^r in the hourglass consisting of a', b_h, and the paths π_0^ℓ and π_0^r. Note that the whole edge b_h is visible from a', since a' intersects the triangle t_h. Thus, the visibility lines go through the endpoints of b_h. It follows that λ_0^ℓ is the line that goes through $\ell(b_h)$ and the unique vertex on π_0^r such that it is tangent to π_0^r; see Figure 6b. This can clearly be found in linear time in the length of π_0^r. The same holds for the right visibility line.

▶ **Lemma 4.** *The algorithm FMLP computes the initial left and right shortest paths π_0^ℓ and π_0^r as well as the corresponding visibility lines in $O(|P'_0|)$ time.*

We compute subsequent windows until we find the last edge b. A minimum-link path π is obtained by connecting each window $w(a)$ to its corresponding first edge a with a straight line [26]. In our implementation, we do not construct P and its triangulation T explicitly, but work directly on the triangulated input graph. The next important triangle is then computed on-the-fly as follows. Consider an important triangle $t_i = uvw$, and let uv be the edge shared by the current and the previous important triangle. Clearly, exactly one endpoint of uv is part of the reachable boundary, so without loss of generality let u be this endpoint. Then the next important triangle is the triangle sharing vw with t_i if w is reachable, and the triangle sharing uw with t_i otherwise. In other words, the next triangle is determined by the unique edge that has exactly one reachable endpoint. Linear running time of the algorithm follows immediately from Lemma 3 and Lemma 4. Theorem 5 summarizes our findings.

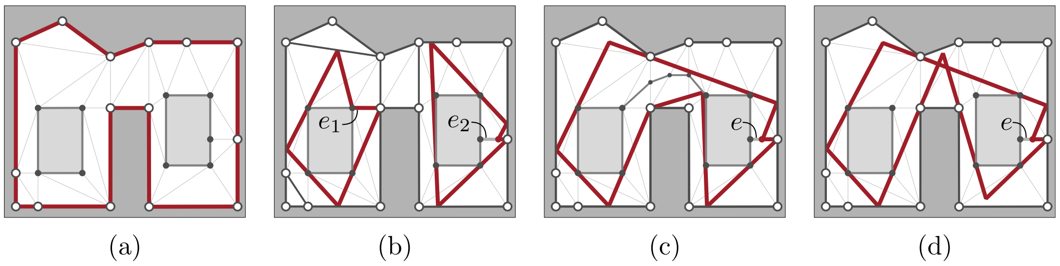


Figure 7 Results of `RP-RC` (a), `RP-TS` (b), `RP-CU` (c), and `RP-SI` (d), starting at indicated edges.

▶ **Theorem 5.** *Given two edges a and b of a simple polygon P, the algorithm FMLP computes a minimum-link path from a to b contained in P in linear time.*

4 Heuristic Approaches for General Border Regions

A border region B may consist of several unreachable components, i.e., $|U| > 1$. In this general case, it is not clear whether one can compute a (non-intersecting) range polygon of minimum complexity in polynomial time [17]. Even for the simpler subproblem of computing a minimum-link path in a polygon with holes (without assigning the holes to the reachable or unreachable part), the fastest known algorithm has quadratic running time [21, 24]. This is impractical for large instances. We propose heuristics with (almost) linear running time (in the size of B) that are simple and fast in practice. Figure 7 shows examples.

The first approach, `RP-RC` (*range polygon*, extracted *reachable components*), simply extracts and returns the reachable boundary R; see Figure 7a. The result resembles previous approaches [12, 22], so it can be seen as an efficient implementation of the state-of-the-art.

Separating Border Regions Along the Triangulation. The second approach, denoted `RP-TS` (*triangular separators*), works as follows. For each border region B, we consider its triangulation. We add all edges of the triangulation that either connect two reachable vertices or two unreachable vertices of G_p to the boundary of B, possibly splitting B into several regions $B' = R' \cup U'$ (see bold edges separating the border region in Figure 7b). For each region B', we obtain $|U'| \leq 1$, since two components of U must be connected by an edge of the triangulation or separated by an edge with two endpoints in R. Then, we run the algorithm presented in Section 3 on each instance B' with $|U'| = 1$ to get the range polygon. Linear running time follows, as FMLP is run on disjoint subregions of B.

Clearly, the set of edges added to B is not minimal (we could possibly omit some and still obtain $|U'| \leq 1$). However, it allows us to implement `RP-TS` without explicitly constructing B': Starting from an arbitrary (unvisited) boundary edge in B with one endpoint in each R and U, we run FMLP and determine the important triangles on-the-fly (as described in Section 3). We mark encountered boundary edges and repeat this procedure until all boundary edges in B (with endpoints in both R and U) were visited. Note that FMLP becomes even simpler in this variant, because regions B' contain only important triangles. However, the number of modified regions B' can become quite large. Below, we propose a more sophisticated approach to obtain regions with a single unreachable component.

Connecting Unreachable Components. Third, `RP-CU` (*connecting unreachable* components) adds new edges to border regions B with $|U| > 1$ to connect all components in U without intersecting the reachable boundary; see Figure 7c.

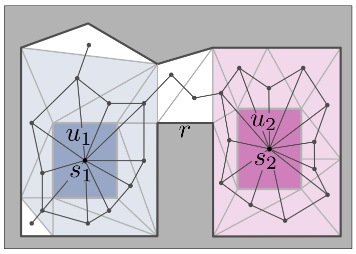


Figure 8 Dual graph with super sources s_1 and s_2. Shaded triangles are assigned to u_1 and u_2, respectively.

Given a border region B with $|U| > 1$, the heuristic starts by checking for each pair of components in U whether it can be connected directly by an edge of the triangulation. This requires traversal of the vertices in unreachable components, scanning for each vertex its incident edges in the triangulation. Edges that connect two unreachable components are added to the boundary of B and the adjacent components are merged (i. e., considered equal in the further course of the algorithm). To connect all remaining unreachable components after this first step, we consider the (weak) dual graph of the triangulation of B; see Figure 8. Since no pair of remaining unreachable components can be connected by a single edge in the primal graph, each triangle intersects at most one unreachable component. We assign a component to each dual vertex, namely, the reachable component if the corresponding triangle contains only reachable vertices, and the unique unreachable component it intersects, otherwise. For each unreachable component, we add a *super source* to the dual graph that is connected to all vertices assigned to this component. Since we want to add as few edges to the primal graph as possible, our goal is to find a tree of minimum total length in the modified dual graph that connects all super sources. Finding such a minimum Steiner tree is NP-hard [13], so we run a heuristic search. It iteratively adds shortest paths between two sources that are not yet connected in a greedy fashion. This is achieved by a multi-source variant of a breadth-first search (BFS) starting from all super sources. Whenever a path connecting two super sources is found, the corresponding components are merged. The algorithm stops when all super sources are connected. Finally, we add new vertices and edges to B along the obtained paths to connect all components in U, as illustrated in Figure 7c. We add further edges to maintain the triangulation, if necessary. The resulting border region B' is solved by the algorithm presented in Section 3. Making use of a union-find data structure, the BFS runs in $O(n\alpha(n))$ time [27], where n is the size of B and α the inverse Ackermann function. All remaining steps run in linear time, so the overall running time is almost linear.

A crucial observation is that realistic instances of border regions often consist of one major unreachable component and many tiny components, as illustrated in Figure 2. To significantly reduce the (empirical) running time, we start the BFS from all but the largest component. This requires little overhead (traversing unreachable components to identify the largest one), but searches from small components are likely to quickly converge to the large component. Moreover, after extracting the next vertex from the queue, we first check whether its source was connected to the largest component in the meantime. If this is the case, we prune the search at this vertex, because it now represents the search from the largest component. Similarly, before running the BFS we also omit traversal of the vertices of the largest component when checking for edges in the triangulation that connect two components. For further (practical) speedup, we modify the BFS to always expand the search from the component that is currently the smallest. This can be done by using a priority queue whose

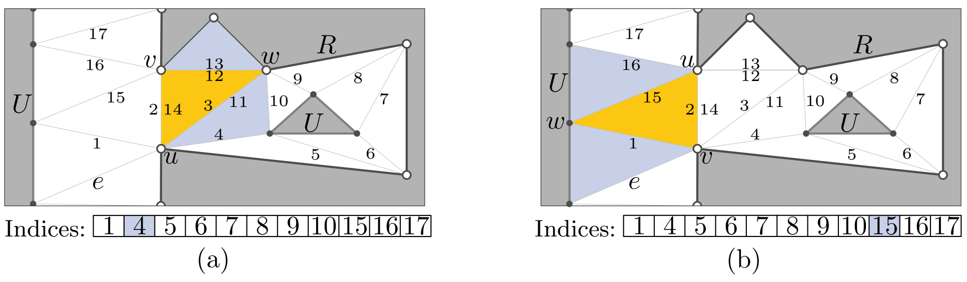

17
16
15
1
U
e
v
u
w
R
13
12
2 14
3
11
4
9
8
10
7
U
5
6
Indices: 1 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 15 | 16 | 17
(a)
17
16
15
1
U
w
e
u
v
R
13
12
2 14
3
11
4
9
8
10
7
U
5
6
Indices: 1 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 15 | 16 | 17
(b)

Figure 9 Edge indices starting at e for each direction of traversal (∞ if not specified). Separator edge indices are given in the list (the next index is shaded). (a) The second triangle (shaded yellow) has two possible next triangles t_{uw} and t_{vw} (shaded blue). The next one is t_{uw}, since the index of the edge vw with higher index (12) is greater than the next separator edge index (4). Observe that the yellow triangle is visited a second time during the course of the algorithm. (b) The next triangle is t_{uw}. The index of the next separator edge is updated from 15 to 16.

elements are components. Additionally, we maintain a queue for each component that stores the vertices visited by the BFS (extracting them in first-in-first-out order). In each step of the BFS, we check for the smallest component in the priority queue, and extract the next vertex from the queue of this component. Note that the use of a priority queue actually increases the asymptotic running time of the BFS, however, we observe a significant speedup in practice.

Self-Intersecting Minimum-Link Paths. Our last approach is denoted `RP-SI` (*self intersecting* polygons). It computes a minimum-link path that separates reachable and unreachable boundaries of B. This path has at most $\mathrm{OPT}+1$ segments (inducing a lower bound for B), but may intersect itself; see Figure 7d. To obtain a range polygon, we add more segments to resolve intersections. Below, we generalize FMLP to border regions with several unreachable components. Note that the (weak) dual graph of the triangulation of B is not outerplanar if $|U| > 1$. Thus, paths between dual vertices are no longer unique and we have to compute a *shortest* path in the dual graph that separates the boundaries. Vertices may even occur multiple times in such a path; see the triangles crossed by the polygon in Figure 7d.

Given a boundary edge e with endpoints in R and U, we compute a sequence $t_1, \dots, t_k$ of important triangles that must be passed in this order by a minimum-link path (between the two sides of e) that separates R and U. Our approach runs in two phases. Exploiting that the reachable boundary is connected, the first phase traverses it starting from e. It assigns *indices* to all edges in the triangulation incident to the reachable boundary, according to the order in which they are traversed. In doing so, we distinguish both sides of edges; see Figure 9. For consistency, sides of edges that are not traversed get the index ∞. During this traversal, we also collect an ordered list of indices corresponding to *separator edges* in the triangulation, i. e., edges with one endpoint in each R and U. Every separator edge in B is traversed exactly once. Moreover, the minimum-link path must intersect these edges in the same order (lest having unreachable components on both sides of the path).

The second phase uses this information to compute the actual sequence of important triangles. This sequence must pass all separator edges exactly once and in increasing order of their indices. Therefore, we obtain the sequence of important triangles by computing shortest paths in the dual graph between pairs of consecutive separator edges. We maintain the index of the next separator edge that was not traversed yet, initialized to the first element of the list. Starting at the triangle t_1 containing the first edge e, we add triangles to the sequence

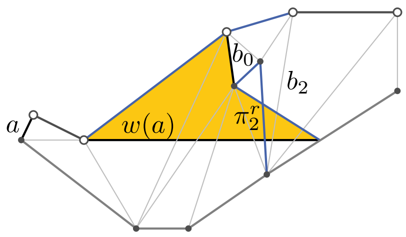


Figure 10 The path π_2^r (blue) between the right endpoints of $w(a)$ and b_2 intersects itself; P_0' is shaded. Note that two unreachable vertices (black) are not connected to the remaining unreachable boundary.

of important triangles until e is reached again. Let $t_i = uvw$ denote the previous triangle appended to this sequence, and uv the edge shared by t_i and t_{i-1} (if $i = 1$, let $uv = e$). We determine the next important triangle t_{i+1}; see Figure 9. We consider the possible next triangles t_{uw} containing the edge uw and t_{vw} containing vw. Without loss of generality, let the index of uw be lower than the index of vw (and thus, finite). This implies that uw is not contained in the boundary of B. If both u and w are part of the reachable boundary, uw separates B into two subregions; see Figure 9a. Thus, t_{uw} is the next triangle if and only if the subregion containing t_{uw} contains a separator edge that was not passed yet. Therefore, we continue with t_{uw} if and only if the index of the other edge vw is higher than the index of the next separator edge. If either u or w is part of the unreachable boundary, uw is the next separator edge; see Figure 9b. We update the index of the next separator edge to the next element in the according list. We continue until the first edge e is reached again. Note that this second phase can be performed on-the-fly within FMLP.

To preserve correctness of FMLP, further modifications are necessary, as a path in the hourglass may intersect itself. Figure 10 shows an example where the subpath of the right shortest path π_2^r starting at edge b_0 intersects the segment from the right endpoint of $w(a)$ to the right endpoint of b_0. The last segment of the right shortest path π_2^r from $w(a)$ to b_2 is called *visibility-intersecting*, as it reaches into the area P_0' visible from $w(a)$. Visibility-intersecting segments may lead to wrong results in certain cases [3]. However, one can show that a segment is visibility-intersecting if and only if it intersects the previous window. Moreover, it can safely be omitted from the shortest path computed by the algorithm without affecting correctness. Thus, FMLP is restored with a simple additional intersection test. In summary, our algorithm consists of two steps (traversing the reachable boundary, running a modified version of FMLP), which clearly run in linear time. The resulting polygon has at most OPT + 2 segments. We rearrange it at intersections to obtain the range polygon [3].

5 Experiments

We implemented all approaches in C++, using g++ 4.8.3 (-O3) as compiler. Experiments were conducted on a single core of a 4-core Intel Xeon E5-1630v3 clocked at 3.7 GHz, with 128 GiB of DDR4-2133 RAM, 10 MiB of L3, and 256 KiB of L2 cache.

Our graph is based on the road network of Western Europe, kindly provided by PTV AG (`ptvgroup.com`). Edge lengths are set to given travel times. For EV range visualization, we also consider energy consumption derived from a detailed micro-scale emission model [18]. Removing edges without reasonable energy consumption (e. g., due to missing elevation data), we obtain a graph with 22 198 628 vertices and 51 088 095 edges. To improve spatial locality, we reorder these vertices according to a vertex partition [5]. During preprocessing, the graph is planarized (654 765 split edges, 293 741 dummy vertices) and triangulated.

Table 1 Computing isochrones and EV range for medium and long ranges. We report average figures for the number of components of the range polygon (Cp.), complexity of the range polygon (Seg.), number of self-intersections (Int.), and running time of the algorithm in ms (Time).

		Med. (16 kWh / 60 min)				Long (85 kWh / 500 min)			
	Algorithm	Cp.	Seg.	Int.	Time	Cp.	Seg.	Int.	Time
EV Range	`RP-RC`	41	19 396	—	4.50	131	92 554	—	9.46
	`RP-TS`	69	610	—	4.30	219	1 973	—	7.78
	`RP-CU`	41	561	—	10.15	131	1 820	—	25.11
	`RP-SI`	41	549	4.79	7.52	131	1 781	15.06	22.25
Isochrones	`RP-RC`	53	22 458	—	4.75	231	238 123	—	20.25
	`RP-TS`	151	1 076	—	4.65	694	4 981	—	14.96
	`RP-CU`	53	913	—	12.11	231	4 208	—	65.09
	`RP-SI`	53	881	9.95	8.70	231	4 055	45.80	51.94

We consider two scenarios, namely isochrones (i. e., travel time is the consumed resource) and range visualization of an EV. For both, we evaluate queries of medium (60 min and 16 kWh, respectively) and long ranges (500 min and 85 kWh). We focus on Steps 2 to 4 outlined in Section 2, since implementation of the first step was examined in previous work [4].

Evaluating Queries. Table 1 shows results for the different scenarios. Each figure is the average of 1 000 queries, with source vertices picked uniformly at random. For `RP-SI`, figures are reported as-is after running FMLP (i. e., for polygons with self-intersections). Thus, figures slightly change after resolving the intersections (both the number of components and the complexity may increase). All approaches perform excellently in practice, with timings of at most 65 ms even for long ranges. The simpler algorithms, `RP-RC` and `RP-TS` are faster by a factor of 2 to 5. On the other hand, range polygons generated by `RP-RC` have a much higher complexity, exceeding the optimum by more than an order of magnitude. For long ranges, polygons consist of more than 200 000 segments. This clearly justifies the use of our novel algorithms. Besides a more appealing visualization, a significant decrease in complexity enables fast rendering and more efficient transmission over mobile networks. The heuristic `RP-TS` provides much better results in terms of complexity, but is still outperformed by the other two approaches. Moreover, the triangular separation increases the number of components (i. e., the number of holes) in the result by up to a factor of 3, while all other approaches are optimal in this criterion. The two more involved approaches, `RP-CU` and `RP-SI`, keep the complexity close to the optimum, so the additional effort clearly pays off. Deriving lower bounds from the results of `RP-SI`, the average relative error of both `RP-CU` (at most 7 %) and `RP-SI` (4 %) is negligible in practice. The number of intersections produced by `RP-SI` is also rather low, but the majority of computed range polygons contains at least one intersection. Isochrones are slightly harder to solve in all cases. For long-range queries, this is due to larger border regions. Setting the resource limit to 500 minutes for isochrones yields one of the hardest scenarios in our instance. For medium ranges, border region sizes are similar in both scenarios. Here, differences in performance can be explained by different shapes of the border regions: Isocontours representing the range of an EV typically have a more circular shape (highways allow to move faster, but also consume more energy). On the contrary, isochrones require more segments and yield more challenging scenarios.

Table 2 Different phases (isochrones, 500 min), showing average time (in ms) for border region extraction (BE), connecting components (CC), range polygon computation with FMLP (RP), testing for self-intersections (SI), and total time (Total).

Algo.	BE	CC	RP	SI	Total
`RP-RC`	12.01	—	—	—	20.25
`RP-TS`	—	—	6.45	—	14.96
`RP-CU`	26.66	22.99	7.81	—	65.09
`RP-SI`	31.79	—	9.53	2.34	51.94

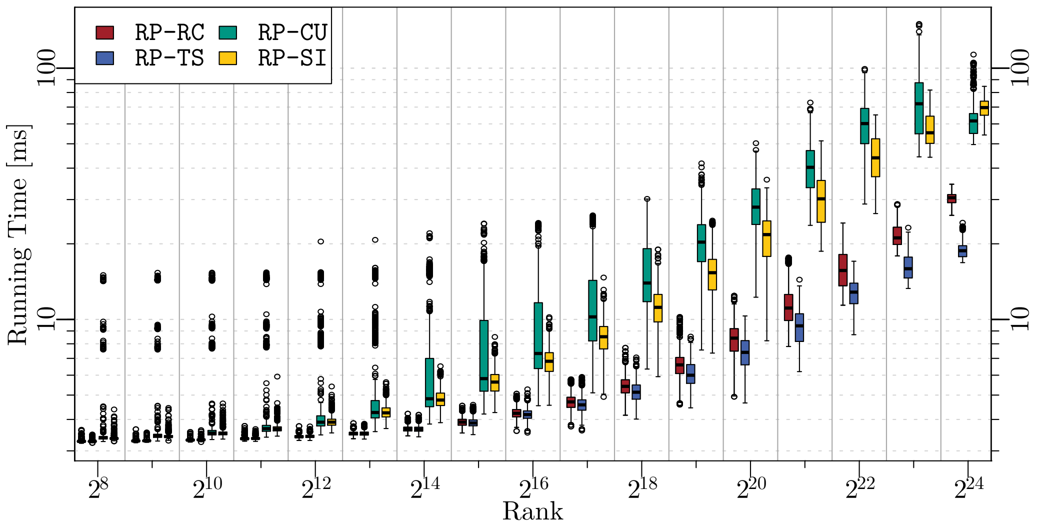


Figure 11 Running times of all approaches subject to Dijkstra rank. Smaller ranks indicate queries of shorter range. For each rank, we report results of 1 000 random queries.

Table 2 shows details on different phases of the algorithms for the hardest scenario (isochrones, 500 min). Step 2 (transferring input to the planar graph) is identical for all approaches, taking 8.2 ms on average (not reported in the table). Border region extraction does not apply to `RP-TS`, where this is done implicitly. Since `RP-RC` extracts only the reachable boundary, this takes less than half the time compared to `RP-CU` (the reachable boundary is typically smaller). On the other hand, `RP-SI` spends most time in this step, since it runs on the triangulated graph.

Despite its simplicity, extraction takes a major fraction of the total effort. This is due to the size of the border regions (500 000 segments per query), while only parts of them are visited in later phases. Consequently, connecting unreachable components takes less time for `RP-CU`. Running FMLP is fastest for `RP-TS`, since it visits only important triangles. The slowest approach in this phase is `RP-SI`, mostly because no artificial edges are added to border regions (windows become longer on average, increasing the number of visited triangles).

Evaluating Scalability. Figure 11 analyzes scalability of our algorithms. We follow the methodology of Dijkstra ranks [1], defined as the number of queue extractions performed by Dijkstra's algorithm in a shortest-path query. Higher ranks reflect harder queries. To generate queries, we ran Dijkstra's algorithm 1 000 times from sources chosen uniformly at random. For a source s, consider the resource consumption c at the vertex extracted from the queue in step 2^i of the algorithm. We consider a query from s with range c as a query of rank 2^i. For each rank in $\{2^1, \dots, 2^{\log |V|}\}$, we obtain 1 000 queries this way.

Query times of all approaches increase with the Dijkstra rank, which correlates well with the complexity of the border region. Scaling behavior is similar for all approaches: In accordance with our theoretical findings, it increases linearly in the size of the border region for queries beyond a rank of 2^{12}. For queries of lower rank, transferring the input to the planar graph dominates running time (which is linear in the graph size). The approach RP-TS is consistently the fastest on average for ranks beyond 2^{16}. Except for very few outliers, query times are well below 100 ms. For more local queries (i. e., smaller ranges), query times are much faster (20 ms and below if the rank is at most 2^{20}, corresponding to about a million vertices visited by Dijkstra's algorithm). Interestingly, the more expensive approaches have a higher variance and produce more outliers, which is explained by their more complex phases. For example, the performance of the BFS used in RP-CU heavily depends on how close unreachable components of the border region are in the dual graph.

Minimum-Link Path Computation. In Table 3, we evaluate FMLP in the four main scenarios (ranges for 16 kWh and 85 kWh batteries and isochrones for 60 min and 500 min). For each of the 1 000 queries per scenario, we modified the largest border region such that $|U| = 1$ (using RP-CU). Then, we added an edge connecting the two remaining components and computed a minimum-link path between its two sides. We also report figures for Suri's algorithm [26], which finds the next window starting from a window a by computing multiple visibility polygons in the following way. Starting with the polygon bounding all important triangles intersected by a, it doubles the number of important triangles until a triangle is (partially) invisible from a. To obtain the window, a final visibility polygon is computed in a polygon bounding the same set of important triangles together with all non-important triangles whose closest important triangle (wrt. distance in the dual graph) belongs to this set. The next window is an edge of this visibility polygon. While a fair comparison with running times of Suri's algorithm is beyond the scope of this work (as it requires an equally tuned implementation), we provide implementation-independent measures. In particular, we report the total number of visible triangles per query, in all polygons that require visibility computation. A recent experimental study on visibility polygon computation [7] presents a practical algorithm based on triangulations that processes only visible triangles, making this figure a good indicator for running time of an efficient implementation of Suri's algorithm.

Different scenarios in Table 3 represent certain levels of difficulty. As expected, the number of segments of the resulting path and the running time of FMLP increase with the complexity of the input. In all scenarios, Suri's algorithm requires several thousand calls to its subroutine for computing visibility polygons. The total number of segments in the input of this subroutine is beyond 1.5 million for long-range isochrones, which even rules out their explicit construction in practice. Additionally, the total number of triangles visited by Suri's algorithm (using the proposed subroutine [7]) is larger by a factor of 5 to 6. Moreover, we argue that this factor is a rather conservative estimate on the resulting speedup: While the workload per triangle is very low for FMLP, the proposed subroutine of Suri's algorithm is recursive and therefore possibly less cache efficient. Suri's algorithm also requires additional

Table 3 Average performance of minimum-link path algorithms. For each scenario, we report complexity of the input polygon ($|P|$) and minimum number of links in resulting paths (Seg.). For Suri's algorithm [26], we show the number of computed visibility polygons (V. Pol.), the total number of segments in the input for these computations (Pol. Seg.), and the total number of visible triangles (Trng.). For FMLP, we provide the number of visited triangles (Trng.) and running time in ms.

			Suri [26]			FMLP	
Scenario	$\|P\|$	Seg.	V. Pol.	Pol. Seg.	Trng.	Trng.	Time
EV, 16 kWh	134 049	415	2 010	307 583	48 762	8 901	0.74
Iso, 60 min	135 112	700	3 413	320 244	57 549	11 250	1.05
EV, 85 kWh	357 335	1 328	6 442	850 293	178 574	31 657	3.17
Iso, 500 min	637 224	3 203	15 655	1 547 962	359 969	66 163	6.67

overhead for generating input polygons and determining the actual windows. Given its simplicity, we conclude that FMLP is much more suitable for practical use. Even for input consisting of more than half a million segments, it takes less than 7 ms.

6 Conclusion

This work introduced algorithms for computing isocontours in large-scale road networks. Following the objectives of exact results, low result complexity, and practical performance, we presented three novel algorithms to compute near-optimal solutions in (almost) linear time. Their key ingredient is a new linear-time algorithm for minimum-link paths in simple polygons, making it the first practical approach to a problem well-studied in theory [21, 23, 24, 26]. Our experimental evaluation reveals that all approaches are fast enough for interactive applications on inputs of continental scale.

There are multiple lines of future work. Extending our algorithms to the case $|R| > 1$ is an open problem relevant for multi-source isocontours and multimodal networks [12]. For aesthetic reasons, one could aim at avoiding long straight segments in the range polygon (which are likely to occur in faces encompassing large areas corresponding to, e. g., big lakes or mountains). Such constraints could be integrated by adding (during preprocessing) artificial boundaries to faces whose area exceeds a certain threshold. Alternatively, one could further reduce result complexity at the cost of inexact results. However, such methods should avoid intersections between different components of the range polygon (i. e., maintain its topology), and error measures should consider the graph-based distance from the source to parts of the network that are classified incorrectly (since vertices close to each other wrt. Euclidean distance may in fact be far apart in the graph).

Acknowledgements. We thank Roman Prutkin for interesting discussions.

References

1 Hannah Bast, Daniel Delling, Andrew V. Goldberg, Matthias Müller–Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. Route Planning in Transportation Networks. Technical Report abs/1504.05140, ArXiv e-prints, 2015.

2 Veronika Bauer, Johann Gamper, Roberto Loperfido, Sylvia Profanter, Stefan Putzer, and Igor Timko. Computing Isochrones in Multi-Modal, Schedule-Based Transport Networks.

In *Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS'08)*, pages 78:1–78:2. ACM, 2008.

3 Moritz Baum, Thomas Bläsius, Andreas Gemsa, Ignaz Rutter, and Franziska Wegner. Scalable Isocontour Visualization in Road Networks via Minimum-Link Paths. Technical Report abs/1602.01777, ArXiv e-prints, 2016.

4 Moritz Baum, Valentin Buchhold, Julian Dibbelt, and Dorothea Wagner. Fast Exact Computation of Isochrones in Road Networks. In *Proceedings of the 15th International Symposium on Experimental Algorithms (SEA'16)*, volume 9685 of *Lecture Notes in Computer Science*, pages 17–32. Springer, 2016.

5 Moritz Baum, Julian Dibbelt, Thomas Pajor, and Dorothea Wagner. Energy-Optimal Routes for Electric Vehicles. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS'13)*, pages 54–63. ACM, 2013.

6 John L. Bentley and Thomas A. Ottmann. Algorithms for Reporting and Counting Geometric Intersections. *IEEE Transactions on Computers*, 28(9):643–647, 1979.

7 Francisc Bungiu, Michael Hemmer, John E. Hershberger, Kan Huang, and Alexander Kröller. Efficient Computation of Visibility Polygons. In *Proceedings of the 30th European Workshop on Computational Geometry (EuroCG'14)*, 2014.

8 Edsger W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

9 Alexandros Efentakis, Sotiris Brakatsoulas, Nikos Grivas, Giorgos Lamprianidis, Kostas Patroumpas, and Dieter Pfoser. Towards a Flexible and Scalable Fleet Management Service. In *Proceedings of the 6th ACM SIGSPATIAL International Workshop on Computational Transportation Science (IWTCS'13)*, pages 79–84. ACM, 2013.

10 Alexandros Efentakis, Nikos Grivas, George Lamprianidis, Georg Magenschab, and Dieter Pfoser. Isochrones, Traffic and DEMOgraphics. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS'13)*, pages 548–551. ACM, 2013.

11 Alexandros Efentakis and Dieter Pfoser. GRASP. Extending Graph Separators for the Single-Source Shortest-Path Problem. In *Proceedings of the 22nd Annual European Symposium on Algorithms (ESA'14)*, volume 8737 of *Lecture Notes in Computer Science*, pages 358–370. Springer, 2014.

12 Johann Gamper, Michael Böhlen, and Markus Innerebner. Scalable Computation of Isochrones with Network Expiration. In *Proceedings of the 24th International Conference on Scientific and Statistical Database Management (SSDBM'12)*, volume 7338 of *Lecture Notes in Computer Science*, pages 526–543. Springer, 2012.

13 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., 1979.

14 Ronald L. Graham. An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set. *Information Processing Letters*, 1(4):132–133, 1972.

15 Leonidas J. Guibas and John E. Hershberger. Optimal Shortest Path Queries in a Simple Polygon. *Journal of Computer and System Sciences*, 39(2):126–152, 1989.

16 Leonidas J. Guibas, John E. Hershberger, Daniel Leven, Micha Sharir, and Robert E. Tarjan. Linear-Time Algorithms for Visibility and Shortest Path Problems Inside Triangulated Simple Polygons. *Algorithmica*, 2(1):209–233, 1987.

17 Leonidas J. Guibas, John E. Hershberger, Joseph S. B. Mitchell, and J. S. Snoeyink. Approximating Polygons and Subdivisions with Minimum-Link Paths. *International Journal of Computational Geometry & Applications*, 3(4):383–415, 1993.

18 Stefan Hausberger, Martin Rexeis, Michael Zallinger, and Raphael Luz. Emission Factors from the Model PHEM for the HBEFA Version 3. Technical Report I-20/2009, University of Technology, Graz, 2009.

19 Hiroshi Imai and Masao Iri. An Optimal Algorithm for Approximating a Piecewise Linear Function. *Journal of Information Processing*, 9(3):159–162, 1987.

20 Markus Innerebner, Michael Böhlen, and Johann Gamper. ISOGA: A System for Geographical Reachability Analysis. In *Proceedings of the 12th International Conference on Web and Wireless Geographical Information Systems (W2GIS'13)*, volume 7820 of *Lecture Notes in Computer Science*, pages 180–189. Springer, 2013.

21 Irina Kostitsyna, Maarten Löffler, Valentin Polishchuk, and Frank Staals. On the Complexity of Minimum-Link Path Problems. In *Proceedings of the 32nd Annual Symposium on Computational Geometry (SoCG'16)*, volume 51 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 49:1–49:16. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2016.

22 Sarunas Marciuska and Johann Gamper. Determining Objects within Isochrones in Spatial Network Databases. In *Proceedings of the 14th East European Conference on Advances in Databases and Information Systems (ADBIS'10)*, volume 6295 of *Lecture Notes in Computer Science*, pages 392–405. Springer, 2010.

23 Joseph S. B. Mitchell, Valentin Polishchuk, and Mikko Sysikaski. Minimum-Link Paths Revisited. *Computational Geometry*, 47(6):651–667, 2014.

24 Joseph S. B. Mitchell, Günter Rote, and Gerhard Woeginger. Minimum-Link Paths Among Obstacles in the Plane. *Algorithmica*, 8(1):431–459, 1992.

25 David O'Sullivan, Alastair Morrison, and John Shearer. Using Desktop GIS for the Investigation of Accessibility by Public Transport: An Isochrone Approach. *International Journal of Geographical Information Science*, 14(1):85–104, 2000.

26 Subhash Suri. A Linear Time Algorithm for Minimum Link Paths Inside a Simple Polygon. *Computer Vision, Graphics, and Image Processing*, 35(1):99–110, 1986.

27 Robert E. Tarjan. Efficiency of a Good But Not Linear Set Union Algorithm. *Journal of the ACM*, 22(2):215–225, 1975.

28 Cao An Wang. Finding Minimal Nested Polygons. *BIT Numerical Mathematics*, 31(2):230–236, 1991.

Computing Equilibria in Markets with Budget-Additive Utilities

Xiaohui Bei[1], **Jugal Garg**[2], **Martin Hoefer**[3], **and Kurt Mehlhorn**[4]

1 **Nanyang Technological University, Singapore**
`xhbei@ntu.edu.sg`

2 **University of Illinois at Urbana-Champaign, USA**
`jugal@illinois.edu`

3 **MPI für Informatik and Saarland University, Saarbrücken, Germany**
`mhoefer@mpi-inf.mpg.de`

4 **MPI für Informatik, Saarbrücken, Germany**
`mehlhorn@mpi-inf.mpg.de`

Abstract

We present the first analysis of Fisher markets with buyers that have budget-additive utility functions. Budget-additive utilities are elementary concave functions with numerous applications in online adword markets and revenue optimization problems. They extend the standard case of linear utilities and have been studied in a variety of other market models. In contrast to the frequently studied CES utilities, they have a global satiation point which can imply multiple market equilibria with quite different characteristics. Our main result is an efficient combinatorial algorithm to compute a market equilibrium with a Pareto-optimal allocation of goods. It relies on a new descending-price approach and, as a special case, also implies a novel combinatorial algorithm for computing a market equilibrium in linear Fisher markets. We complement this positive result with a number of hardness results for related computational questions. We prove that it is NP-hard to compute a market equilibrium that maximizes social welfare, and it is PPAD-hard to find any market equilibrium with utility functions with separate satiation points for each buyer and each good.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Budget-Additive Utility, Market Equilibrium, Equilibrium Computation

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.8

1 Introduction

The concept of market equilibrium is a fundamental and well-established notion in economics to analyze and predict the outcomes of strategic interaction in large markets. Initiated by Walras in 1874, the study of market equilibrium has become a cornerstone of microeconomic analysis, mostly due to general results that established existence under very mild conditions [2]. Since efficient computation is a fundamental criterion to evaluate the plausibility of equilibrium concepts, the algorithmic aspects of market equilibrium are one of the central domains in algorithmic game theory. Over the last decade, several new algorithmic approaches to compute market equilibria were discovered. Efficient algorithms based on convex programming techniques can compute equilibria in a large variety of domains [12, 22, 25]. More importantly, several approaches were proposed that avoid the use of heavy algorithmic machinery and follow combinatorial strategies [17, 26, 29, 32, 20, 19], or even work as a tâtonnement process in unknown market environments [13, 10, 4]. Designing such combinatorial algorithms is useful also beyond the study of markets, since the underlying ideas can be applied in other areas. Variants of these algorithms were shown to solve scheduling [23, 24] and cloud computing problems [15], or can be used for fair allocation of indivisible items [14].

In this paper, we design a new combinatorial polynomial time algorithm for computing equilibria in Fisher markets with budget-additive utilities. In a Fisher market, there is a single seller with a set $G = \{1, \ldots, m\}$ of goods. W.l.o.g. we assume that the total quantity of each good is 1. There is a set $B = \{1, \ldots, n\}$ of buyers. Each buyer i has a budget $M_i > 0$ of money and a utility function u_i. For budget-additive utilities, $u_{ij} \geq 0$ is the utility of buyer i if one unit of good j is allocated to her. There is a *happiness cap* $c_i > 0$, and the utility function is

$$u_i(\mathbf{x}_i) = \min\left\{ c_i, \sum_{j \in G} u_{ij} x_{ij} \right\} \quad ,$$

where $\mathbf{x}_i = (x_{ij})_{j \in G}$ is any bundle of goods assigned to buyer i. If $u_i(\mathbf{x}_i) = c_i$, then buyer i is called *capped buyer* for allocation $\mathbf{x}$. We assume all u_{ij}, c_i, M_i are rational numbers.

Our goal is to compute an *allocation* $\mathbf{x} = (\mathbf{x}_i)_{i \in B}$ of goods and *prices* $\mathbf{p} = (p_j)_{j \in G}$ such that the pair $(\mathbf{x}, \mathbf{p})$ is a market equilibrium. Given prices $\mathbf{p}$, a *demand bundle* $\mathbf{x}_i^*$ of buyer i is a bundle of goods that maximizes the utility of buyer i for its budget, i.e., $\mathbf{x}_i^* \in \arg\max \left\{ u_i(\mathbf{x}_i) \mid \sum_j p_j x_{ij} \leq M_i \right\}$. Note that $\sum_j u_{ij} x_{ij}^* > c_i$ is allowed. A *market equilibrium* $(\mathbf{x}, \mathbf{p})$ is a pair such that

- $\mathbf{p} \geq 0$ (prices are nonnegative),
- $\sum_i x_{ij} \leq 1$ for every $j \in G$ (no overallocation),
- $\mathbf{x}_i$ is a demand bundle for every $i \in B$, and
- Walras' law holds: $p_j(1 - \sum_i x_{ij}) = 0$ for every $j \in G$.

Note that if $\sum_i x_{ij} < 1$, then $p_j = 0$. An equilibrium $(\mathbf{x}, \mathbf{p})$ is *Pareto-optimal* if there is no equilibrium $(\mathbf{x}', \mathbf{p}')$ such that $u_i(\mathbf{x}) \leq u_i(\mathbf{x}')$ for all i and $u_i(\mathbf{x}) < u_i(\mathbf{x}')$ for at least one i.

Budget-additive utility functions are a simple class of submodular and concave functions and a natural generalization of the standard and well-understood case of linear utilities. These utility functions arise naturally in cases where agents have an intrinsic upper bound on their utility. For example, if the goods are food and the utility of a food item for a particular buyer is its calorie content, calories above a certain threshold do not increase the utility of the buyer. In addition, there are a variety of further applications in adword auctions and revenue maximization problems [1, 3, 8, 6]. Recently, market models where agents have budget-additive utilities attracted a significant amount of research interest, e.g., for the allocation of indivisible goods in offline [1, 3, 8] and online [6, 27] scenarios, for truthful mechanism design [7], and for the study of Walrasian equilibrium with quasi-linear utilities [21, 18, 30]. As simple variants of submodular functions, they capture many of the inherent difficulties of more general domains. Given this amount of interest, it is perhaps surprising that they are not well-understood within the classic Fisher and exchange markets.

Results and Contribution. We study Fisher markets with budget-additive utilities. Our initial observations about these markets reveal that they have different properties than the ones with CES utilities usually studied in the literature. Due to the satiated nature of the utilities, capped buyers might not spend all their money or spend money on goods that do not give them maximum utility per unit of money, so prices and utilities in market equilibrium are not unique and can be quite different. It is possible to simply ignore the satiation and assume linear utilities. Then a variety of existing algorithms [17, 29, 20, 19, 4] can be used to compute a market equilibrium. It continues to be a market equilibrium for the market with budget-additive utilities. However, this equilibrium may be undesirable, as in many cases it does not even satisfy Pareto-optimality of the allocation.

▶ **Example 1.** Consider a *linear* market with two buyers and two goods, $u_{11} = 5$, $u_{21} = 2$, and $u_{12} = u_{22} = 1$. The budgets are $M_1 = 3$ and $M_2 = 1$. For the unique equilibrium we allocate good 1 completely to buyer 1 and good 2 completely to buyer 2, i.e., $x_{11} = x_{22} = 1$. The buyers' utilities amount to 5 and 1, resp., and the prices are $p_1 = 3$ and $p_2 = 1$.

Now suppose buyer 1 has a budget-additive utility function with cap $c_1 = 1$. Then $(\mathbf{x}, \mathbf{p})$ described above remains an equilibrium, since both buyers obtain a demand bundle (buyer 1 now has utility 1 instead of 5). Alternatively, suppose we allocate good 1 completely to buyer 2 and good 2 completely to buyer 1, i.e., $x_{12} = x_{21} = 1$. The utilities amount to 1 and 2, resp., and the prices can be chosen as $p_1 = 1$ and $p_2 \in [0.5, 3]$. Here buyer 1 buys a bundle of goods with optimal utility of 1. Buyer 2 buys a demand bundle since he spends all its budget on a good that gives him the maximum *bang-per-buck ratio*. All goods are exactly allocated, and Walras' law holds. Thus, it represents another market equilibrium. Note if $p_2 < 3$, buyer 1 does not spend all of its money, but it is still a demand bundle for because he achieves the maximum utility. Furthermore, such an equilibrium Pareto-dominates the one derived from the linear case in terms of utilities. ◀

We strive to compute a market equilibrium with a Pareto-optimal allocation and focus on a subset of market equilibria, in which we restrict the allocation to demand bundles which we call thrifty and modest – buyers spend the least amount of money that can achieve their optimal utilities and receive a bundle of goods that has a minimality property. In Section 2, we show that such *modest MBB equilibria* can be captured by a generalization of the classic Eisenberg-Gale convex program, their utilities are unique, and their allocation is always Pareto-optimal (w.r.t. all possible allocations, attainable in market equilibrium or not). We highlight that the set of modest MBB equilibria can be partially ordered with respect to their price vectors and forms a lattice. As such, there are modest MBB equilibria with pointwise largest and smallest prices, resp. Among all modest MBB equilibria they yield maximum and minimum revenue for the seller, resp.

Section 3 contains our main contribution – a combinatorial algorithm that computes price and allocation vectors of a modest MBB equilibrium in time $O(mn^6(\log(m + n) + (m + n)\log U))$, where n is the number of agents, m the number of goods, and U the largest integer in the market parameters. The computed equilibrium has a Pareto-optimal allocation, as well as pointwise largest prices and maximum revenue among all modest MBB equilibria.

Our algorithm represents a novel approach to compute market equilibria based on the idea of descending prices. While some parts of our algorithm are in spirit of combinatorial algorithms for linear markets [17, 20, 19, 4], all these approaches are ascending-price algorithms. This technique and its usual analysis based on the 1-norm of excess money does not apply in our case, since the norm is non-monotonic and cannot be used to measure progress towards equilibrium. Surprisingly, our novel descending-price approach overcomes the 1-norm issue, but we need to address additional challenges in establishing polynomial running time due to varying and non-increasing active budgets, and in showing that intermediate prices remain polynomially bounded. Note that, as a special case, this also yields a new combinatorial descending-price algorithm for linear Fisher markets.

In Section 4 we exploit the lattice structure of modest MBB equilibria and design a procedure, using which we can turn any modest MBB equilibrium into one with smallest prices and minimum revenue. In combination with the descending-price algorithm, it computes a modest MBB equilibrium with minimum revenue within the same asymptotic time bound.

Finally, we study two extensions in Section 5. Facing multiple equilibria, a natural goal is to compute an allocation that maximizes utilitarian social welfare. We prove that this problem is NP-hard, even when social welfare is measured by a k-norm of the vector of buyer utilities, for any constant $k > 0$. Moreover, we consider a variant of budget-additive utilities with a satiation point for *each buyer and each good.* They constitute a special class of separable piecewise-linear concave (SPLC) utilities, where each piecewise-linear component consists of two segments with the second one being constant. We show that even in this very special case computing any market equilibrium becomes PPAD-hard.

Related Work. The computation of market equilibria is a central area in algorithmic game theory. There are a variety of polynomial-time algorithms to compute approximate market equilibria based on solving different convex programming formulations [12, 22, 25]. Our paper is closer to work on markets with linear utilities and combinatorial algorithms that compute an exact equilibrium in polynomial time [17, 29, 20, 19]. Directly related to our approach is the classic combinatorial algorithm for linear Fisher markets [17]. In contrast, our algorithm is based on a new descending price approach where buyers are always saturated and goods have non-negative surplus. Further, the *active* budgets of buyers vary with the price change, which creates new challenges in establishing a polynomial bound on the number of iterations and the representation size of intermediate prices.

Independently of our work, Devanur et al [16] very recently presented the same convex program for Fisher markets with satiated buyers. They propose a polynomial-time algorithm for finding an arbitrary modest MBB equilibrium, but it is based on the ellipsoid method without any explicit running time bound.

Recently, algorithmic work has also started to address unknown markets, where utilities and budgets of buyers are unknown. Instead, algorithms iteratively set prices and query a demand oracle. In this domain, tâtonnement dynamics have been studied for Fisher markets and extensions with concave utilities. For many classes of these markets, a notion of $(1+\epsilon)$-approximate market equilibrium can be reached after a convergence time polynomial in $1/\epsilon$ and other market parameters [13, 10, 11, 5]. In some cases, the convergence time can even be reduced to $\log(1/\epsilon)$ [10]. A similar convergence rate is obtained by a more general algorithm even for general unknown exchange markets with weak gross-substitutes property, and even for linear markets with non-continuous demands and oracles using suitable tie-breaking [4].

Allocation of indivisible items to agents with budget-additive utilities is an active area of research interest. There are constant-factor approximation algorithms for optimizing the allocation in offline [1, 3, 8] and online [6, 27] scenarios. Closer to our work is the study of markets with money. The existence of Walrasian equilibrium with quasi-linear utilities and algorithmic issues of bundling items were studied in [21, 18, 30]. Strategic agents and truthful mechanisms for budget-additive markets have been analyzed in [7]. There are strong lower bounds for the approximation ratio of certain classes of truthful mechanisms, and a truthful mechanism with constant-factor approximation for budget-additive utilities is one of the most interesting open problems in combinatorial auctions.

2 Preliminaries

For a given price vector $\mathbf{p}$ and buyer i, we denote the *maximum bang-per-buck (MBB)* ratio by $\alpha_i = \max_j u_{ij}/p_j$, where we make the assumption that $0/0 = 0$. Budget-additive utilities strictly generalize linear utilities: when all c_i's are large enough, they are equivalent to linear utilities. If buyer i is uncapped in a market equilibrium $(\mathbf{x}, \mathbf{p})$, it behaves as in the linear case, spends all its budget, and buys only MBB goods ($x_{ij} > 0$ only if $u_{ij}/p_j = \alpha_i$). Otherwise, if buyer i is capped in $(\mathbf{x}, \mathbf{p})$, it might buy non-MBB goods and not spend all of its budget. This implies that unlike the case of linear utilities, market equilibrium prices and utilities are not unique with budget-additive utilities.

It is easy to see that we can obtain one market equilibrium by simply ignoring the happiness caps and treating the market as a linear one. However, this equilibrium is often undesirable since it is not always Pareto-optimal.

Our main goal in this paper is to find a market equilibrium that is Pareto-optimal. More generally, we will also be concerned with finding a (Pareto-optimal) market equilibrium that can maximize social welfare $\sum_{i\in B} u_i(\mathbf{x}_i)$. For the former we provide a polynomial-time algorithm, the latter we prove it to be NP-hard.

Modest MBB Equilibria, Pareto-Optimality, and Uniqueness. The main challenges in budget-additive markets arise from capped buyers, who may possibly have multiple choices for the demand bundle. Let us introduce two convenient restrictions on the allocation to capped buyers.

- An allocation $\mathbf{x}_i$ for buyer i is called *modest* if $\sum_j u_{ij}x_{ij} \le c_i$. By definition, for uncapped buyers every demand bundle is modest. For capped buyers, a modest bundle of goods $\mathbf{x}_i$ is such that utility breaks even between the linear part and c_i, i.e., $c_i = \sum_j u_{ij}x_{ij}$.
- A demand bundle $\mathbf{x}_i$ is called *thrifty* or *MBB* if it consists of only MBB goods: $x_{ij} > 0$ only if $u_{ij}/p_j = \alpha_i$. As noted above, for uncapped buyers every demand bundle is MBB.

We call a market equilibrium $(\mathbf{x}, \mathbf{p})$ a *modest MBB equilibrium* if $\mathbf{x}_i$ is modest and MBB for every buyer $i \in B$. We show an algorithm to compute in polynomial time such an equilibrium where $\mathbf{x}$ is also Pareto-optimal. Such an equilibrium is also desirable because it captures the behavioral assumption that each buyer is thrifty and spends the least amount of money in order to obtain a utility maximizing bundle of goods.

Consider the following Eisenberg-Gale program (1), which allows us to find a modest and Pareto-optimal allocation.

$$\begin{array}{lrcll}
\text{Max.} & \displaystyle\sum_{i\in B} M_i \log \sum_{j\in G} u_{ij}x_{ij} & & & \\
\text{s.t.} & \displaystyle\sum_{j\in G} u_{ij}x_{ij} & \le & c_i & i \in B \\
 & \displaystyle\sum_{i\in B} x_{ij} & \le & 1 & j \in G \\
 & x_{ij} & \ge & 0 & i \in B,\ j \in G
\end{array} \tag{1}$$

By standard arguments, we consider the dual for (1) using dual variables γ_i and p_j for the first two constraints, resp., and the KKT conditions read:

1. $p_j/u_{ij} \ge M_i/u_i - \gamma_i$
2. $x_{ij} > 0 \Rightarrow p_j/u_{ij} = M_i/u_i - \gamma_i$
3. $p_j \ge 0$ and $p_j > 0 \Rightarrow \sum_{i\in B} x_{ij} = 1$
4. $\gamma_i \ge 0$ and $\gamma_i > 0 \Rightarrow u_i = c_i$

Observe that the Lagrange multiplier γ_i indicates if the cap c_i represents a tight constraint in the optimum solution. The dual variables p_j can be interpreted as prices. Note that conditions 1 and 2 imply that $x_{ij} > 0$ if and only if $j \in \arg\min_{j'} p_{j'}/u_{ij'} = \arg\max_{j'} u_{ij'}/p_{j'}$, i.e., all agents purchase goods with maximum bang-per-buck. Hence, similarly as for linear markets [31], the KKT conditions imply that an optimal solution to the EG program (1) and corresponding dual prices constitute a market equilibrium, in which every agent buys goods that have maximum bang-per-buck. The KKT conditions postulate this also for agents whose utility reaches the cap. Thus, the optimal solution to this program is a modest MBB equilibrium. Furthermore, we obtain the following favorable analytical properties.

▶ **Proposition 2.** *The optimal solutions to* (1) *are exactly the modest MBB equilibria. The utility vector is unique across all such equilibria and each such equilibrium is Pareto-optimal. In particular, there is a unique set of capped buyers. Non-capped buyers spend all their money. Capped buyers do not overspend.*

While utilities are unique, allocation and prices of modest MBB equilibria might not be unique. Consider a market with two identical buyers and two goods, where $u_{11} = u_{12} = u_{21} = u_{22} = 1$, $c_1 = c_2 = 1$, and $M_1 = M_2 = 5$. The unique equilibrium utility of both buyers is $u_1 = u_2 = 1$, which can be obtained for any $p_1 = p_2 = p$, where $p \in [0, 5]$ and allocation $\mathbf{x}$ satisfying $x_{11} + x_{12} = 1$; $x_{21} + x_{22} = 1$; $x_{11} + x_{21} = 1$; $x_{12} + x_{22} = 1$.

▶ **Example 1** (continued)**.** For our example above, the modest MBB equilibrium obtained from solving the convex program is $x_{11} = 1/5$, $x_{12} = 0$, $x_{21} = 4/5$ and $x_{22} = 1$ with prices $p_1 = 10/13$ and $p_2 = 5/13$. Buyer 1 spends 2/13, buyer 2 spends the entire budget. The utilities are 1 and 13/5. It is easy to see that the KKT conditions hold. This equilibrium is Pareto-optimal and also the best equilibrium in terms of social welfare. ◀

Structure of Modest MBB Equilibria. Let us characterize the set of price vectors of modest MBB equilibria, which we denote by $\mathcal{P} = \{\mathbf{p} \mid (\mathbf{x}, \mathbf{p}) \text{ is modest MBB equilibrium }\}$. We consider the coordinate-wise comparison, i.e., $\mathbf{p} \geq \mathbf{p}'$ iff $p_j \geq p'_j$ for all $j \in G$.

▶ **Theorem 3.** *The pair* $(\mathcal{P}, \geq)$ *is a lattice.*

Given $\mathbf{p}$ and $\mathbf{p}'$, we partition the set of goods into three sets: $S_0 = \{j \mid p_j = p'_j\}$, $S_1 = \{j \mid p_j < p'_j\}$ and $S_2 = \{j \mid p_j > p'_j\}$. Let $\Gamma(T, \mathbf{p}) = \{i \mid x_{ij} > 0 \text{ for some } j \in T\}$ denote the set of buyers who are allocated a nonzero amount of any good in set T in the equilibrium $(\mathbf{x}, \mathbf{p})$. The proof of Theorem 3 exploits the following properties about the sets S_0, S_1 and S_2.

▶ **Lemma 4.** *Given any two modest MBB equilibria* $(\mathbf{x}, \mathbf{p})$ *and* $(\mathbf{x}', \mathbf{p}')$*, we have*
- **(i)** $\Gamma(S_i, \mathbf{p}) = \Gamma(S_i, \mathbf{p}')$ *for* $i = 0, 1, 2$*, i.e., the set of buyers who buy the goods of* S_i *with respect to prices* $\mathbf{p}$ *and* $\mathbf{p}'$ *are same.*
- **(ii)** $\Gamma(S_0, \mathbf{p}), \Gamma(S_1, \mathbf{p})$ *and* $\Gamma(S_2, \mathbf{p})$ *are mutually disjoint.*
- **(iii)** *All buyers in* $\Gamma(S_1, \mathbf{p})$ *and* $\Gamma(S_2, \mathbf{p})$ *are capped buyers in both* $(\mathbf{x}, \mathbf{p})$ *and* $(\mathbf{x}', \mathbf{p}')$.

Thus, there exists a modest MBB equilibrium with coordinate-wise highest (resp. lowest) prices. It yields the maximum (resp. minimum) revenue for the seller among all modest MBB equilibria.

▶ **Example 5.** Consider the following market with two buyers and two goods. Let $u_{11} = u_{12} = u_{22} = 1$ and $u_{21} = 0$. Let $M_1 = M_2 = 1$ and $c_1 = 1$. Then $x_{11} = x_{22} = 1$, $x_{12} = 0$, $p_1 = p_2 = 1$ is a modest MBB equilibrium with maximum revenue. A modest MBB equilibrium with minimum revenue has the same allocation and $p_1 = 0$ and $p_2 = 1$.

3 Computing a Modest MBB Equilibrium with Maximum Revenue

In this section, we describe an efficient algorithm to compute a modest MBB equilibrium. In fact, we compute the one with coordinate-wise highest prices and maximum revenue among all modest MBB equilibria. We define the *active budget* of buyer i at prices $\mathbf{p}$ as $M_i^a = \min\{M_i, c_i/\alpha_i\}$, where $\alpha_i = \max_{j\in G} u_{ij}/p_j$ is the MBB ratio. The active budget of buyer i is the minimum of M_i and the minimum amount of money needed to buy a bundle of goods with utility c_i. If $M_i^a = c_i/\alpha_i$, then buyer i is capped, otherwise uncapped.

3.1 Flow Network and Initialization

Given prices $\mathbf{p}$, let $A = \{(i,j) \subseteq B \times G \mid u_{ij}/p_j = \alpha_i\}$ be the set of equality edges, and the bipartite graph $(B \cup G, A)$ be the *equality graph*. We set up the following flow network N_p using the equality graph by adding a source s and sink t. It has nodes $\{s,t\} \cup B \cup G$ and edges (s,i) for $i \in B$, (j,t) for $j \in G$ and the equality edges. The edge (s,i) has capacity M_i^a, and the edge (j,t) has capacity p_j. The equality edges have infinite capacity. The flow in the network corresponds to money. We will maintain the following invariants throughout the algorithm.

Invariants:

- The edges out of s are saturated.
- Prices and active budgets never increase.
- Total utility of a buyer never decreases. Once a buyer is capped, it stays capped.

We initialize the prices to large values, namely $p_j = \sum_i M_i$. W.l.o.g. we will assume that all budgets, caps, and utilities are integers.

The surplus (residual capacity) of good j is $r_j = p_j - f_{jt}$, where f_{jt} is the flow from good j to t. Then $1 - f_{jt}/p_j$ is the fraction of good j that is not sold. We also keep track of the allocations x_{ij}. There might be prices equal to zero, and then the allocation cannot be computed from the money flow. Goods that have price zero have no surplus. There is no money flowing through them, although they may be (partially) allocated.

A subset T of buyers is called *tight* with respect to prices $\mathbf{p}$ if $\sum_{i\in T} M_i^a = \sum_{j\in\Gamma(T)} p_j$, where $\Gamma(T) \subseteq S$ is the set of goods connected to T in the equality graph.

A *balanced flow* is a maximum flow in N_p which minimizes the 2-norm of surplus vector r. Let $|r| = |r_1| + \ldots + |r_n|$ and $\|r\| = (r_1^2 + \ldots + r_n^2)^{1/2}$ be the ℓ_1 and ℓ_2 norm of r, resp.

3.2 The Algorithm

The complete algorithm is shown in Figure 1. We initialize price p_j of each good j to $\sum_i M_i$. This ensures that the invariants are satisfied, namely a maximum flow in network N_p saturates all edges out of s. We initialize every active budget $M_i^a = \min\{M_i, \min_j c_i p_j/u_{ij}\}$, and flow f and allocation x equal to zero.

The algorithm is divided into a set of phases, and each phase is further divided into a set of iterations. A phase starts with the computation of a balanced flow in N_p. Let the surplus of good j be $p_j - f_{jt}$. We pick a good j with maximum surplus, and we compute a set of goods S containing j and the goods which can reach j in the residual network corresponding to N_p without using nodes s and t. The surplus of each good in S is the same, and maximum among all goods. We denote by B' the set of buyers who have equality edges to goods in S, and by B'_c and B'_u the sets of capped and uncapped buyers in B', resp. Note that $x_{ij} = 0$ for all $i \in B'$ and $j \notin S$, since $x_{ij} > 0$ would imply $j \in S$.

We begin with an iteration, where we use a factor x to set the price of each good $j \in S$ to xp_j and the active budget of each buyer $i \in B'_c$ to xM_i^a. The prices and active budgets of the remaining goods and buyers remain unchanged. We decrease $x \leq 1$ continuously until

```
Input: A market with a set of buyers $B$ and a set of goods $G$;
       Budget $M_i$, happiness cap $c_i$, and utility parameters $u_{ij}$, $\forall i \in B, j \in G$;
Output: Equilibrium prices $\mathbf{p}$, allocation $x$;
$n \leftarrow |B|$; $m \leftarrow |G|$; $U \leftarrow \max_{i\in B, j\in G}\{M_i, c_i, u_{ij}\}$; $\epsilon \leftarrow 1/((m+n)U^{4(m+n)})$;
Initialize price $p_j \leftarrow \sum_i M_i$ for each good $j$;
Initialize active budget $M_i^a \leftarrow \min\{M_i, \min_j c_i p_j/u_{ij}\}$ for each buyer $i$;
$f_{ij} \leftarrow 0$, $x_{ij} \leftarrow 0$, $\forall i \in B, j \in G$;
Repeat // phase
    $f \leftarrow$ balanced flow in $N_p$; $x_{ij} \leftarrow f_{ij}/p_j$ if $p_j \neq 0$; $r_j \leftarrow p_j - f_{jt}$;
    $\delta \leftarrow \max_j r_j$; Pick a good $j$ with surplus $\delta$;
    $S \leftarrow \{j\} \cup \{k \in G \mid k$ can reach $j$ in the residual network w.r.t. $f$ in $N_p \setminus \{s,t\}\}$;
    Repeat // iteration
        $B' \leftarrow$ Set of buyers who have incident equality edges to $S$;
        $B'_c \leftarrow$ Set of capped buyers in $B'$ (a buyer $i$ is capped if $M_i^a = \min_j {}^{c_i p_j}/_{u_{ij}}$);
        $B'_u \leftarrow B' \setminus B'_c$ (set of uncapped buyers);
        $x \leftarrow 1$; Set prices and active budgets as follows:
            $p_j \leftarrow x p_j$, $\forall j \in S$; $M_i^a \leftarrow x M_i^a$, $\forall i \in B'_c$;
        Decrease $x$ continuously down from 1 until one of the following events occurs
        Event 1: An uncapped buyer becomes capped
        Event 2: A new equality edge appears
            Recompute $N_p$;
            $f \leftarrow$ balanced flow in $N_p$; $x_{ij} \leftarrow f_{ij}/p_j$ if $p_j \neq 0$;
            $S \leftarrow S \cup \{j \in G \mid j$ can reach $S$ in the residual network w.r.t. $f$ in $N_p \setminus \{s,t\}\}$;
        Event 3: A subset of $B'$ becomes tight // phase ends
    Until Event 3 occurs;
Until $|r| \leq \epsilon$;
Recompute $N_p$;
$f \leftarrow$ balanced flow in $N_p$; $x_{ij} \leftarrow f_{ij}/p_j$ if $p_j \neq 0$;
```

Figure 1 The complete algorithm.

some structural change happens. Our goal here is to decrease prices as much as possible. By changing prices in this manner, all the equality edges between B' and S stay intact and the equality edges between B' and $G \setminus S$ become non-equality.

A possible structural change is that an uncapped buyer becomes capped. When a buyer $i \in B'$ is uncapped, $M_i < \min_j c_i p_j/u_{ij}$. Prices are decreasing, so this may become an equality. We term the first such change Event 1. Then we move buyer i from B'_u to B'_c.

Another possible change is that a new equality edge appears from a buyer in $B \setminus B'$ to a good in S. Prices of goods in S are decreasing, so goods in S are becoming attractive to buyers outside B'. Note that there cannot be a new equality edge from a buyer in B' to a good outside S. We term the first such change Event 2. Then we recompute the flow network N_p and a balanced flow in N_p. Next, we compute the set S' of goods $j \in G \setminus S$ that can reach a good in S in the residual graph corresponding to N_p without using the nodes s and t. Due to the property of balanced flows, the surplus of each good in S' is at least the surplus of some good in S. Finally, we add goods in S' to S.

Apart from the structural changes, we also maintain the invariants. The only invariant that can become violated with these changes is that the edges out of s are saturated. Hence, we need to stop when a subset T of B' becomes tight. Clearly, if prices are decreased further, then buyers in T will not be saturated, so we stop decreasing prices at this stage. We term this Event 3, and then the phase ends. We show in Lemma 10 below that during a phase, the 2-norm of the surplus vector decreases geometrically. The last phase ends when the total surplus becomes tiny. In fact, we will show that the surplus is actually zero at this point. We recompute a balanced flow and terminate.

When the prices of a set of goods hit zero in an iteration of the algorithm, then we do not change the allocation of these goods, and all the buyers interested in these goods must be capped. Since each buyer gets a modest allocation before the prices hit zero, the same allocation remains modest. None of the goods in the set is completely allocated. We delete these goods and the buyers to which they are allocated from consideration.

▶ **Example 1** (continued). Consider our algorithm applied to the example market above. We initialize p_1 and p_2 to $M_1 + M_2 = 4$. The active budgets become $M_1^a = \min_j c_1 p_j / u_{ij} = 4/5$ and $M_2^a = 1$. The edges $(1,1)$ and $(2,1)$ are equality edges and the balanced flow is $f_{11} = 4/5$, $f_{21} = 1$, and $f_{12} = f_{22} = 0$. The surpluses are $r_1 = 4 - 9/5 = 11/5$ and $r_2 = 4 - 0 = 4$. Thus $S = \{2\}$. We decrease p_2 to xp_2. At $x = 1/2$, the edge $(2,2)$ becomes an equality edge. Now $p_2 = 2$. The balanced flow does not change and hence $r_1 = 11/5$ and $r_2 = 2$. Thus $S = \{1\}$. We decrease M_1^a to $4x/5$ and p_1 to $4x$. At $x = 5/16$, B' becomes tight. We now have $M_1^a = 1/4$ and $p_1 = 5/4$. The balanced flow is $f_{11} = 1/4$ and $f_{21} = 1$. Thus $r_1 = 0$ and $r_2 = 2$. So $S = \{2\}$. We change p_2 to $p_2 x$. At $x = 5/16$, the edge $(2,2)$ becomes an equality edge. Now $p_2 = 5/8$. The balanced flow is $f_{11} = 1/4$, $f_{21} = 11/16$, and $f_{22} = 5/16$. Then $r_1 = r_2 = 5/16$. Thus $S = \{1,2\}$. We now decrease M_1^a to $x \cdot 1/4$, p_1 to $5x/4$ and p_2 to $5x/8$. At $x = 8/13$, B' becomes tight and we have $p_1 = 10/13$, $p_2 = 5/13$, $M_1^a = 2/13$, $x_{11} = 1/5$, $x_{21} = 4/5$, $x_{22} = 1$, $f_{11} = 2/13$, $f_{21} = 8/13$, and $f_{22} = 5/13$. ◀

3.3 Analysis

▶ **Lemma 6.** *The invariants hold during the run of the algorithm.*

Phases consist of iterations, which end with Event 1, 2, or 3. A phase ends with Event 3.

▶ **Lemma 7.** *Each phase has at most $2n$ iterations.*

Our next goal is to show that the 2-norm of the surplus vector decreases substantially during a phase. Let r and r' be the surplus vectors at the beginning and at the end of a phase, resp. For the purpose of our analysis we also maintain an intermediate flow f continuously as we change prices in each iteration; this flow is *not* maintained by the algorithm. When we recompute a balanced flow during Event 2, then f will be reset to the balanced flow. It is defined as $\forall i \in B'_c : f_{ij} \leftarrow x f_{ij}$ and $\forall i \in B'_u : f_{ij} \leftarrow f_{ij}$. f ensures that all buyers are saturated. If the surplus of a good j becomes zero corresponding to f, then we keep its surplus equal to zero and reroute extra flow from j to some other good with positive surplus, using a path in the residual network corresponding to f. If there is no such path, then this implies Event 3 has occurred, in which case the current phase is done. Consider an intermediate iteration t. With respect to f, let $r^t = (r_1^t, \ldots, r_m^t)$ be the surplus vector at the beginning of iteration t, and let $\tilde{r}^t = (\tilde{r}_1^t, \ldots, \tilde{r}_m^t)$ be the surplus vector before we recompute a balanced flow in iteration t if Event 2 occurs.

▶ **Lemma 8.** *$\tilde{r}_j^t \le r_j^t$, $\forall j \in G$, and $\|r^{t+1}\| \le \|\tilde{r}^t\| \le \|r^t\|$.*

▶ **Lemma 9.** *[17] Suppose f and f^* are a feasible and a balanced flow in N_p, resp., and r and r^* are the surplus vectors w.r.t. f and f^*, resp. If $r_j^* = r_j - \delta$ for some good j and $\delta > 0$, then $\|r^*\|^2 \le \|r\|^2 - \delta^2$.*

▶ **Lemma 10.** $\|r'\|^2 \leq (1 - 1/4mn)\|r\|^2$.

Proof. Consider the value of $\gamma = \min\{r_j \mid j \in S\}$ during a phase. When the phase begins, $\gamma = \delta$, and when it ends $\gamma = 0$. Recall that S only grows, and when we add a new good k to S, then the surplus of k is at least the surplus of some good already in S. This implies that γ does not change when we add new goods to S.

Let $t_1, \ldots, t_l$ be the iterations where γ decreases, and let $\delta_i > 0$ be the amount of decrease in iteration t_i. Further we break each δ_i into two parts δ_{i1} and δ_{i2} such that $\delta_i = \delta_{i1} + \delta_{i2}$. Here δ_{i1} is the amount of decrease due to the flow change before we recompute balanced flow, and δ_{i2} is the amount of decrease due to recomputation of balanced flow. Next consider only positive δ_{i1}'s and δ_{i2}'s. Clearly, $l \leq 2n$ and $\sum_{i:\delta_{i1}>0,\delta_{i2}>0}(\delta_{i1} + \delta_{i2}) \geq \delta$. Using Lemmas 8 and 9, we have $\|r'\|^2 \leq \|r\|^2 - (\delta_{11}^2 + \delta_{12}^2 + \cdots + \delta_{l1}^2 + \delta_{l2}^2) \leq \|r\|^2 - \delta^2/4n$. Since $\|r\|^2 \leq m\delta^2$, we have $\|r'\|^2 \leq (1 - 1/4mn)\|r\|^2$. ◀

Polynomial Running Time. In each iteration, the prices of goods in S are multiplied by a value that itself depends on the prices. It is not obvious why the size of the numbers in the computation is polynomially bounded. Here we show that, indeed, the sizes of all intermediate prices and flows in our algorithm remain polynomially bounded.

▶ **Lemma 11.** *All goods in S are connected by equality edges at all times. There is no flow from buyers in B' to goods outside S.*

Cap-events occur only at a cap-event prices. A cap-event price is any price p with $M_i = c_i p/u_{ij}$ for some i and j. Let $P_c = \{M_i u_{ij}/c_i \mid 1 \leq i \leq n, 1 \leq j \leq m\}$.

Let A' be any subset of the edge set with positive utility such that the graph formed by it is connected. Let B' and G' be the buyers and goods in this connected graph. The prices in the component have only one degree of freedom, i.e., we can select one of the prices, say p, as a base price and express any other price in the component as αp, where α is a rational whose numerator and denominator are products of at most m utilities. Consider an arbitrary partition of B' into capped buyers B'_c and uncapped buyers B'_u; B'_u must be nonempty. The budget of a capped buyer is of the form $c\alpha p$, where c is a cap and α is as above. If there are no surpluses, p must satisfy that (budget of capped buyers + budget of uncapped buyers) equals sum of the prices (in the component). We call a price that can be obtained in this way a submarket price; note that not all submarket prices can actually occur. Let P_m be the set of *submarket prices.*

Let P_i be the set consisting of the initial price and zero. A price is *1-linked* if it is of the form $(U_1/U_2)p$ where $p \in P_c \cup P_m \cup P_i$ and U_1 and U_2 are products of at most n utilities each. A price is *2-linked* if it of the form $(U_1/U_2)p$, where p is 1-linked and U_1 and U_2 are products of at most n utilities each.

▶ **Lemma 12.** *Assuming that all budgets, happiness caps and utilities are integers bounded by U, 1-linked and 2-linked prices are rational numbers whose bit-length is at most* $\log(m+n) + 3(m+n)\log U$.

▶ **Lemma 13.** *At the beginning of a phase, all prices are 1-linked. During a phase, prices outside S are 1-linked. At the end of each iteration, prices in S are 2-linked.*

▶ **Theorem 14.** *The algorithm in Figure 1 computes a modest MBB equilibrium.*

Proof. When the algorithm terminates, we claim that at this stage total surplus $\sum_j r_j = 0$. This will imply that the algorithm in Figure 1 computes a market equilibrium. Consider any good j and the component C of the equality graph containing good j. The total surplus

$\sum_{j\in C} r_j$ in the component is $\sum_{j\in C} p_j - \sum_{i\in C} M_i^a$. This is non-negative and less than ϵ. All prices and active budgets of capped buyers can be expressed in terms of one price variable p using equality relations. By Lemma 12, p is a rational number with bounded denominator, and the above inequalities imply $\sum_{j\in C} r_j = 0$. Thus $r_j = 0$ for all j. ◀

Let x_{ts} denote the value of x when Event 3 occurs in the algorithm. Next we show that x_{ts} can be computed using at most n max-flow computations. This is a generalization of a procedure in [17] for computing tight set in case of *linear* Fisher markets.

▶ **Lemma 15.** *x_{ts} can be computed using at most n max-flow computation.*

Maximum Revenue. Finally, we show that our algorithm gives a modest MBB equilibrium with maximum revenue among all modest MBB equilibria.

▶ **Lemma 16.** *Consider the price vector $\mathbf{p}$ at the end of any phase of the algorithm. We have $\mathbf{p} \geq \mathbf{p}'$ for any price vector $\mathbf{p}' \in \mathcal{P}$ of a modest MBB equilibrium.*

For the main result in this section, assume that all budgets, happiness caps and utilities are integers bounded by U.

▶ **Theorem 17.** *The algorithm in Figure 1 computes a modest MBB equilibrium with maximum revenue in $O(mn^6(\log(m+n) + (m+n)\log U))$ time.*

Proof. In the beginning, the 2-norm of surplus vector r satisfies $\|r\|^2 \leq mn^2U^2$. By Theorem 14, the algorithm will terminate before the norm becomes $\|r'\|^2 = 1/m(m+n)^2U^{8(m+n)}$. Let k denote the number of phases when the surplus becomes r'. From Lemma 10, we have $\|r\|^2(1-1/4mn)^k = \|r'\|^2$, which implies that the total number of phases in the algorithm is $O(mn(\log(m+n) + (m+n)\log U))$.

In each phase, we have at most $2n$ iterations, and in each iteration we need to compute the maximum $0 \leq x \leq 1$ when one of the three events occurs. Let x_c, x_{eq} and x_{ts} respectively denote the maximum value of x where Event 1, 2 and 3 occurs. Clearly, x_c can be obtained in $O(n)$ time, x_{eq} can be obtained in $O(mn)$ time, and x_{ts} can be obtained using at most n max-flow computations due to Lemma 15. Further, we recompute a balanced flow in case of Event 2 which further requires at most n max-flow computations [17]. Since a max-flow can be obtained in $O(n^3)$ time, each iteration can be implemented in $O(n^4)$ time. Hence, the total running time of the algorithm is $O(mn^6(\log(m+n) + (m+n)\log U))$. ◀

We conjecture that the running time in Theorem 17 can be reduced by a factor of $\tilde{O}(n^2)$ using the perturbation technique from [19] which requires a max-flow to be computed only in a network with forest structure. We have not worked out the details.

4 Computing a Modest MBB Equilibrium with Minimum Revenue

In this section, we show how to transform in polynomial time any modest MBB equilibrium into one with minimum revenue using the postprocessing procedure in Fig. 2.

▶ **Theorem 18.** *The algorithm in Figure 2 computes a modest MBB equilibrium with minimum revenue.*

Proof. It is easy to check that throughout the algorithm, $(\mathbf{x}, \mathbf{p})$ always remains a modest MBB equilibrium. Assume by contradiction that at the end of the algorithm, $(\mathbf{x}, \mathbf{p})$ is not an equilibrium with smallest prices. Let $(\mathbf{x}', \mathbf{p}')$ be an equilibrium with smallest prices, and

```
Input: A market with a set of buyers $B$ and a set of goods $G$;
        Budget $M_i$, happiness cap $c_i$, and utility parameters $u_{ij}$, $\forall i \in B, j \in G$;
        Any modest MBB equilibrium $(\mathbf{x}, \mathbf{p})$;
Output: A modest MBB equilibrium $(\mathbf{x}, \mathbf{p})$ with minimum revenue;
Initialize active budget $M_i' \leftarrow \min\{M_i, \min_j c_i p_j / u_{ij}\}$ for each buyer $i$;
$S \leftarrow \{j | p_j > 0$ and $j$ does not have incident equality edges to any uncapped buyer$\}$;
While $S \neq \emptyset$
    $B' \leftarrow$ Set of buyers who have incident equality edges to $S$;
    $x \leftarrow 1$; Define prices and active budgets as follows:
        $p_j \leftarrow x p_j$, $\forall j \in S$; $M_i^a \leftarrow x M_i^a$, $\forall i \in B_c'$;
    Decrease $x$ continuously down from 1 until one of the following events occurs
    Event 1: $x$ becomes zero;
    Event 2: A new equality edge appears
    Recompute $N_p$ and $S$;
EndWhile
```

Figure 2 The postprocessing algorithm for an equilibrium with minimum revenue.

define $S_1 = \{j \mid p_j > p_j'\}$. By Lemma 4 property (3), all buyers in $\Gamma(S_1, \mathbf{p})$ are capped buyers. Because prices of goods in set S_1 decrease from $\mathbf{p}$ to $\mathbf{p}'$, every buyer i incident to S_1 in the equality graph with prices $\mathbf{p}$ will only have equality edges to S_1 with prices $\mathbf{p}'$. Therefore we have $i \in \Gamma(S_1, \mathbf{p}') = \Gamma(S_1, \mathbf{p})$ (the equality is again by Lemma 4). This implies $\Gamma(S_1, \mathbf{p})$ is also the set of buyers who have incident equality edges to S_1 with prices $\mathbf{p}$. Hence, set S is nonempty for the While loop, and the algorithm should not terminate. ◀

5 Extensions

In the previous section, we proposed an algorithm for computing a modest MBB equilibrium, which has a Pareto-optimal allocation. When we depart from the set of such equilibria, then utilities in market equilibrium are not uniquely determined. In fact, we show that market equilibria with maximum social welfare might not be modest MBB equilibria, and computing such optimal equilibria becomes NP-hard. As a corollary, we note that the proof can also be used to show NP-hardness for optimizing any constant norm of utility values.

▶ **Theorem 19.** *It is* NP*-hard to compute a market equilibrium that maximizes social welfare.*

▶ **Corollary 20.** *It is* NP*-hard to compute a market equilibrium* $(\mathbf{x}, \mathbf{p})$ *that maximizes* $\sum_i (u_i(\mathbf{x}))^\rho$*, for every constant* $\rho > 0$.

There are several ways of introducing satiation points into the utility function. Instead of a global cap, let us assume there is a cap c_{ij} for the utility buyer i can obtain from good j. A good-based budget-additive utility of buyer i is then $u_i(\mathbf{x}_i) = \sum_j \min(c_{ij}, u_{ij} x_{ij})$. This variant turns out to be an elementary special case of separable piecewise-linear concave (SPLC) utilities, in which every piece consists of a linear segment followed by a constant segment. We show that even finding a single market equilibrium here becomes PPAD-hard. The proof adjusts a construction put forward in [9].

▶ **Theorem 21.** *It is* PPAD*-hard to compute a market equilibrium in Fisher markets with good-based budget-additive utilities.*

References

1 Nir Andelman and Yishay Mansour. Auctions with budget constraints. In *Proc. 9th Scandinavian Workshop Algorithm Theory (SWAT)*, pages 26–38, 2004.

2 Kenneth Arrow and Gerard Debreu. Existence of an equilibrium for a competitive economy. *Econometrica*, 22(3):265–290, 1954.

3 Yossi Azar, Benjamin Birnbaum, Anna Karlin, Claire Mathieu, and C. Thach Nguyen. Improved approximation algorithms for budgeted allocations. In *Proc. 35th Intl. Coll. Automata, Languages and Programming (ICALP)*, volume 1, pages 186–197, 2008.

4 Xiaohui Bei, Jugal Garg, and Martin Hoefer. Ascending-price algorithms for unknown markets. In *Proc. 17th Conf. Economics and Computation (EC)*, 2016. To appear.

5 Benjamin Birnbaum, Nikhil Devanur, and Lin Xiao. Distributed algorithms via gradient descent for Fisher markets. In *Proc. 12th Conf. Electronic Commerce (EC)*, pages 127–136, 2011.

6 Niv Buchbinder, Kamal Jain, and Joseph Naor. Online primal-dual algorithms for maximizing ad-auctions revenue. In *Proc. 15th European Symp. Algorithms (ESA)*, pages 253–264, 2007.

7 Dave Buchfuhrer, Shaddin Dughmi, Hu Fu, Robert Kleinberg, Elchanan Mossel, Christos Papadimitriou, Michael Schapira, Yaron Singer, and Chris Umans. Inapproximability for VCG-based combinatorial auctions. In *Proc. 21st Symp. Discrete Algorithms (SODA)*, pages 518–536, 2010.

8 Deeparnab Chakrabarty and Gagan Goel. On the approximability of budgeted allocations and improved lower bounds for submodular welfare maximization and gap. *SIAM J. Comput.*, 39(6):2189–2211, 2010.

9 Xi Chen and Shang-Hua Teng. Spending is not easier than trading: On the computational equivalence of Fisher and Arrow-Debreu equilibria. In *Proc. 20th Intl. Symp. Algorithms and Computation (ISAAC)*, pages 647–656, 2009.

10 Yun Kuen Cheung, Richard Cole, and Nikhil Devanur. Tatonnement beyond gross substitutes? Gradient descent to the rescue. In *Proc. 45th Symp. Theory of Computing (STOC)*, pages 191–200, 2013.

11 Yun Kuen Cheung, Richard Cole, and Ashish Rastogi. Tatonnement in ongoing markets of complementary goods. In *Proc. 13th Conf. Electronic Commerce (EC)*, pages 337–354, 2012.

12 Bruno Codenotti and Kasturi Varadarajan. Computation of market equilibria by convex programming. In Nisan et al. [28], chapter 8.

13 Richard Cole and Lisa Fleischer. Fast-converging tatonnement algorithms for one-time and ongoing market problems. In *Proc. 40th Symp. Theory of Computing (STOC)*, pages 315–324, 2008.

14 Richard Cole and Vasilis Gkatzelis. Approximating the Nash social welfare with indivisible items. In *Proc. 47th Symp. Theory of Computing (STOC)*, pages 371–380, 2015.

15 Nikhil Devanur, Jugal Garg, Ruta Mehta, Vijay Vazirani, and Sadra Yazdanbod. A market for scheduling, with applications to cloud computing. CoRR abs/1511.08748, 2015.

16 Nikhil Devanur, Kamal Jain, Tung Mai, Vijay Vazirani, and Sadra Yazdanbod. New convex programs for Fisher's market model and its generalizations, 2016. CoRR abs/1603.01257.

17 Nikhil Devanur, Christos Papadimitriou, Amin Saberi, and Vijay Vazirani. Market equilibrium via a primal–dual algorithm for a convex program. *J. ACM*, 55(5), 2008.

18 Shahar Dobzinski, Michal Feldman, Inbal Talgam-Cohen, and Omri Weinstein. Welfare and revenue guarantees for competitive bundling equilibrium. In *Proc. 11th Conf. Web and Internet Economics (WINE)*, pages 300–313, 2015.

19 Ran Duan, Jugal Garg, and Kurt Mehlhorn. An improved combinatorial polynomial algorithm for the linear Arrow-Debreu market. In *Proc. 27th Symp. Discrete Algorithms (SODA)*, pages 90–106, 2016.

20 Ran Duan and Kurt Mehlhorn. A combinatorial polynomial algorithm for the linear Arrow-Debreu market. *Inf. Comput.*, 243:112–132, 2015.

21 Michal Feldman, Nick Gravin, and Brendan Lucier. Combinatorial Walrasian equilibrium. *SIAM J. Comput.*, 45(1):29–48, 2016.

22 Gagan Goel and Vijay Vazirani. A perfect price discrimination market model with production, and a rational convex program for it. *Math. Oper. Res.*, 36(4):762–782, 2011.

23 Sungjin Im, Janardhan Kulkarni, and Kamesh Munagala. Competitive algorithms from competitive equilibria: non-clairvoyant scheduling under polyhedral constraints. In *Proc. 46th Symp. Theory of Computing (STOC)*, pages 313–322, 2014.

24 Sungjin Im, Janardhan Kulkarni, and Kamesh Munagala. Competitive flow time algorithms for polyhedral scheduling. In *Proc. 56th Symp. Foundations of Computer Science (FOCS)*, pages 506–524, 2015.

25 Kamal Jain. A polynomial time algorithm for computing the Arrow-Debreu market equilibrium for linear utilities. *SIAM J. Comput.*, 37(1):306–318, 2007.

26 Kamal Jain and Vijay Vazirani. Eisenberg-Gale markets: Algorithms and game-theoretic properties. *Games Econom. Behav.*, 70(1):84–106, 2010.

27 Michael Kapralov, Ian Post, and Jan Vondrák. Online submodular welfare maximization: Greedy is optimal. In *Proc. 24th Symp. Discrete Algorithms (SODA)*, pages 1216–1225, 2013.

28 Noam Nisan, Éva Tardos, Tim Roughgarden, and Vijay Vazirani, editors. *Algorithmic Game Theory*. Cambridge University Press, 2007.

29 James Orlin. Improved algorithms for computing Fisher's market clearing prices. In *Proc. 42nd Symp. Theory of Computing (STOC)*, pages 291–300, 2010.

30 Tim Roughgarden and Inbal Talgam-Cohen. Why prices need algorithms. In *Proc. 16th Conf. Economics and Computation (EC)*, pages 19–36, 2015.

31 Vijay Vazirani. Combinatorial algorithms for market equilibria. In Nisan et al. [28], chapter 7.

32 László Végh. A strongly polynomial algorithm for generalized flow maximization. In *Proc. 46th Symp. Theory of Computing (STOC)*, pages 644–653, 2014.

On the Lattice Distortion Problem*†

Huck Bennett[1], **Daniel Dadush**[‡2], **and Noah Stephens-Davidowitz**[3]

1 **Department of Computer Science, Courant Institute of Mathematical Sciences, New York University, New York, USA**
`hbennett@cs.nyu.edu`
2 **Centrum Wiskunde & Informatica, Amsterdam, The Netherlands**
`dadush@cwi.nl`
3 **Department of Computer Science, Courant Institute of Mathematical Sciences, New York University, New York, USA**
`noahsd@gmail.com`

Abstract

We introduce and study the *Lattice Distortion Problem* (LDP). LDP asks how "similar" two lattices are. I.e., what is the minimal distortion of a linear bijection between the two lattices? LDP generalizes the Lattice Isomorphism Problem (the lattice analogue of Graph Isomorphism), which simply asks whether the minimal distortion is one.

As our first contribution, we show that the distortion between any two lattices is approximated up to a $n^{O(\log n)}$ factor by a simple function of their successive minima. Our methods are constructive, allowing us to compute low-distortion mappings that are within a $2^{O(n \log \log n / \log n)}$ factor of optimal in polynomial time and within a $n^{O(\log n)}$ factor of optimal in singly exponential time. Our algorithms rely on a notion of basis reduction introduced by Seysen (Combinatorica 1993), which we show is intimately related to lattice distortion. Lastly, we show that LDP is NP-hard to approximate to within any constant factor (under randomized reductions), by a reduction from the Shortest Vector Problem.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases lattices, lattice distortion, lattice isomoprhism, geometry of numbers, basis reduction

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.9

1 Introduction

An n-dimensional *lattice* $\mathcal{L} \subset \mathbb{R}^n$ is the set of all integer linear combinations of linearly independent vectors $B = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$ with $\mathbf{b}_i \in \mathbb{R}^n$. We write the lattice generated by basis B as $\mathcal{L}(B) = \{\sum_{i=1}^n a_i \mathbf{b}_i : a_i \in \mathbb{Z}\}$.

Lattices are very well-studied classical mathematical objects (e.g., [25, 9]), and over the past few decades, computational problems on lattices have found a remarkably large number of applications in computer science. Algorithms for lattice problems have proven to be quite useful, and they have therefore been studied extensively (e.g., [20, 16, 3, 24]). And, over the past twenty years, many strong cryptographic primitives have been constructed with

* The full version of this paper is available at `http://arxiv.org/abs/1605.03613`.
† This material is based upon work partially supported by the National Science Foundation under Grant No. CCF-1320188 and Grant No. CCF-1423228. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.
‡ Daniel Dadush was supported by the NWO Veni grant 639.071.510.

their security based on the (worst-case) hardness of various computational lattice problems (e.g., [1, 23, 12, 11, 28, 8]).

In this paper, we address a natural question: how "similar" are two lattices? I.e., given lattices $\mathcal{L}_1, \mathcal{L}_2$, does there exist a linear bijective mapping $T : \mathcal{L}_1 \to \mathcal{L}_2$ that does not change the distances between points by much? If we insist that T exactly preserves distances, then this is the *Lattice Isomorphism Problem* (LIP), which was studied in [26, 32, 15, 21]. We extend this to the *Lattice Distortion Problem* (LDP), which asks how well such a mapping T can *approximately* preserve distances between points.

Given two lattices $\mathcal{L}_1, \mathcal{L}_2$, we define the *distortion* between them as

$$\mathcal{D}(\mathcal{L}_1, \mathcal{L}_2) = \min\{\|T\|\|T^{-1}\| \,:\, T(\mathcal{L}_1) = \mathcal{L}_2\} \ ,$$

where $\|T\| = \sup_{\|\mathbf{x}\|=1} \|T\mathbf{x}\|$ is the *operator norm.* The quantity $\kappa(T) = \|T\| \cdot \|T^{-1}\|$ is the *condition number* of T, which measures how much T "distorts distances" (up to a fixed scaling). It is easy to check that $\mathcal{D}(\mathcal{L}_1, \mathcal{L}_2)$ bounds the ratio between most natural geometric parameters of $\mathcal{L}_1$ and $\mathcal{L}_2$ (up to scaling), and hence $\mathcal{D}(\mathcal{L}_1, \mathcal{L}_2)$ is a strong measure of "similarity" between lattices. In particular, $\mathcal{D}(\mathcal{L}_1, \mathcal{L}_2) = 1$ if and only if $\mathcal{L}_1, \mathcal{L}_2$ are isomorphic (i.e., if and only if they are related by a scaled orthogonal transformation).

The *Lattice Distortion Problem* (LDP) is then defined in the natural way as follows. The input is two n-dimensional lattices $\mathcal{L}_1, \mathcal{L}_2$ (each represented by a basis), and the goal is to compute a bijective linear transformation T mapping $\mathcal{L}_1$ to $\mathcal{L}_2$ such that $\kappa(T) = \mathcal{D}(\mathcal{L}_1, \mathcal{L}_2)$. In this work, we study the approximate search and decisional versions of this problem, defined in the usual way. We refer to them as γ-LDP and γ-GapLDP respectively, where $\gamma = \gamma(n) \geq 1$ is the approximation factor. (See Section 2.4 for precise definitions.)

1.1 Our Contribution

As our first main contribution, we show that the distortion between any two lattices can be approximated by a natural function of geometric lattice parameters. Indeed, our proof techniques are constructive, leading to our second main contribution: an algorithm that computes low-distortion mappings, with a trade-off between the running time and the approximation factor. Finally, we show hardness of approximating lattice distortion.

To derive useful bounds on the distortion between two lattices, it is intuitively clear that one should study the "different scales over which the two lattices live." A natural notion of this is given by the successive minima, which are defined as follows. The i^{th} successive minimum, $\lambda_i(\mathcal{L})$, of $\mathcal{L}$ is the minimum radius $r > 0$ such that $\mathcal{L}$ contains i linearly independent vectors of norm at most r. For example, a lattice generated by a basis of orthogonal vectors of lengths $0 < a_1 \leq \cdots \leq a_n$ has successive minima $\lambda_i(\mathcal{L}) = a_i$. Since low-distortion mappings approximately preserve distances, it is intuitively clear that two lattices can only be related by a low-distortion mapping if their successive minima are close to each other (up to a fixed scaling).

Concretely, for two n-dimensional lattices $\mathcal{L}_1, \mathcal{L}_2$, we define

$$M(\mathcal{L}_1, \mathcal{L}_2) = \max_{i \in [n]} \frac{\lambda_i(\mathcal{L}_2)}{\lambda_i(\mathcal{L}_1)} \ , \tag{1}$$

which measures how much we need to scale up $\mathcal{L}_1$ so that its successive minima are at least as large as those of $\mathcal{L}_2$. For any linear map T from $\mathcal{L}_1$ to $\mathcal{L}_2$, it is easy to see that $\lambda_i(\mathcal{L}_2) \leq \|T\|\lambda_i(\mathcal{L}_1)$. Thus, by definition $M(\mathcal{L}_1, \mathcal{L}_2) \leq \|T\|$. Applying the same reasoning for T^{-1}, we derive the following simple lower bound on distortion.

$$\mathcal{D}(\mathcal{L}_1, \mathcal{L}_2) \geq M(\mathcal{L}_1, \mathcal{L}_2) \cdot M(\mathcal{L}_2, \mathcal{L}_1) \ . \tag{2}$$

We note that this lower bound is tight when $\mathcal{L}_1, \mathcal{L}_2$ are each generated by bases of orthogonal vectors. But, it is a priori unclear if any comparable upper bound should hold for general lattices, since the successive minima are a very "coarse" characterization of the geometry of the lattice. Nevertheless, we show a corresponding upper bound.

▶ **Theorem 1.** *Let $\mathcal{L}_1, \mathcal{L}_2$ be n-dimensional lattices. Then,*

$$M(\mathcal{L}_1, \mathcal{L}_2) \cdot M(\mathcal{L}_2, \mathcal{L}_1) \le \mathcal{D}(\mathcal{L}_1, \mathcal{L}_2) \le n^{O(\log n)} \cdot M(\mathcal{L}_1, \mathcal{L}_2) \cdot M(\mathcal{L}_2, \mathcal{L}_1) \ .$$

In particular, Theorem 1, together with standard transference theorems (e.g., [7]), implies that $n^{O(\log n)}$-GapLDP is in $\mathsf{NP} \cap \mathsf{coNP}$. While the factor on the right-hand side of the theorem might be far from optimal, we show in Section 5.1 that it cannot be improved below $\Omega(\sqrt{n})$. Intuitively, this is because there exist lattices that are much more dense than $\mathbb{Z}^n$ over large scales but still have $\lambda_i(\mathcal{L}) = \Theta(1)$ for all i. I.e., there exist very dense lattice sphere packings (see, e.g., [31]).

To prove the above theorem, we make use of the intuition that a low-distortion mapping T from $\mathcal{L}_1$ to $\mathcal{L}_2$ should map a "short" basis B_1 of $\mathcal{L}_1$ to a "short" basis B_2 of $\mathcal{L}_2$. (Note that the condition $TB_1 = B_2$ completely determines $T = B_2 B_1^{-1}$.) The difficulty here is that standard notions of "short" fail for the purpose of capturing low-distortion mappings. In particular, in Section 5.2, we show that Hermite-Korkine-Zolotarev (HKZ) reduced bases, one of the strongest notions of "shortest possible" lattice bases, do not suffice by themselves for building low-distortion mappings. (See Section 2.6 for the definition of HKZ-reduced bases.) In particular, we give a simple example of a lattice $\mathcal{L}$ where an HKZ-reduced basis of $\mathcal{L}$ misses the optimal distortion $\mathcal{D}(\mathbb{Z}^n, \mathcal{L})$ by an exponential factor.

Fortunately, we show that a suitable notion of shortness does exist for building low-distortion mappings by making a novel connection between low-distortion mappings and a notion of basis reduction introduced by Seysen [30]. In particular, for a basis $B = [\mathbf{b}_1, \dots, \mathbf{b}_n]$ and dual basis $B^* = B^{-T} = [\mathbf{b}_1^*, \dots, \mathbf{b}_n^*]$, Seysen's condition number is defined as

$$S(B) = \max_{i \in [n]} \|\mathbf{b}_i\| \|\mathbf{b}_i^*\| \ .$$

Note that we always have $\langle \mathbf{b}_i, \mathbf{b}_i^* \rangle = 1$, so this parameter measures how tight the Cauchy-Schwarz inequality is over all primal-dual basis-vector pairs. We extend this notion and define $S(\mathcal{L})$ as the minimum of $S(B)$ over all bases B of $\mathcal{L}$. Using this notion, we give an effective version of Theorem 1 as follows.

▶ **Theorem 2.** *Let $\mathcal{L}_1, \mathcal{L}_2$ be n-dimensional lattices. Let $B_1, B_2 \in \mathbb{R}^{n \times n}$ be bases of $\mathcal{L}_1, \mathcal{L}_2$ whose columns are sorted in non-decreasing order of length. Then, we have that*

$$M(\mathcal{L}_1, \mathcal{L}_2) M(\mathcal{L}_2, \mathcal{L}_1) \le \kappa(B_2 B_1^{-1}) \le n^4 S(B_1)^2 S(B_2)^2 \cdot M(\mathcal{L}_1, \mathcal{L}_2) M(\mathcal{L}_2, \mathcal{L}_1) \ .$$

In particular, we have that

$$M(\mathcal{L}_1, \mathcal{L}_2) M(\mathcal{L}_2, \mathcal{L}_1) \le \mathcal{D}(\mathcal{L}_1, \mathcal{L}_2) \le n^4 S(\mathcal{L}_1)^2 S(\mathcal{L}_2)^2 \cdot M(\mathcal{L}_1, \mathcal{L}_2) M(\mathcal{L}_2, \mathcal{L}_1) \ .$$

From here, the bound in Theorem 1 follows directly from the following (surprising) theorem of Seysen.

▶ **Theorem 3** (Seysen [30]). *For any $\mathcal{L} \subset \mathbb{R}^n$, $S(\mathcal{L}) \le n^{O(\log n)}$.*

This immediately yields an algorithm for approximating the distortion between two lattices, by using standard lattice algorithms to approximate $M(\mathcal{L}_1, \mathcal{L}_2)$ and $M(\mathcal{L}_2, \mathcal{L}_1)$. But, Seysen's proof of the above theorem is actually constructive! In particular, he shows how to efficiently convert any suitably reduced lattice basis into a basis with a low Seysen condition number. (See Section 2.6.2 for details.) Using this methodology, combined with standard basis reduction techniques, we derive the following time-approximation trade-off for γ-LDP.

▶ **Theorem 4** (Algorithm for LDP). *For any $\log n \leq k \leq n$, there is an algorithm solving $k^{O(n/k+\log n)}$-LDP in time $2^{O(k)}$.*

In other words, using the bounds in Theorem 1 together with known algorithms, we are able to approximate the distortion between two lattices. But, with a bit more work, we are able to solve *search* LDP by explicitly computing a low-distortion mapping between the input lattices.

We also prove the following lower bound for LDP.

▶ **Theorem 5** (Hardness of LDP). *γ-GapLDP is NP-hard under randomized polynomial-time reductions for any constant $\gamma \geq 1$.*

In particular, we show a reduction from approximating the (decisional) Shortest Vector Problem (GapSVP) over lattices to γ-GapLDP, where the approximation factor that we obtain for GapSVP is $O(\gamma)$. Since hardness of GapSVP is quite well-studied [2, 22, 17, 14], we are immediately able to import many hardness results to GapLDP. (See Corollary 30 and Theorem 31 for the precise statements.)

1.2 Comparison to related work

The main related work of which we are aware is that of Haviv and Regev [15] on the Lattice Isomorphism Problem (LIP). In their paper, they give an $n^{O(n)}$-time algorithm for solving LIP exactly, which proceeds by cleverly identifying a small candidate set of bases of $\mathcal{L}_1$ and $\mathcal{L}_2$ that must be mapped to each other by any isomorphism. One might expect that such an approach should also work for the purpose of solving LDP either exactly or for approximation factors below $n^{O(\log n)}$. However, the crucial assumption in LIP, that vectors in one lattice must be mapped to vectors of the same length in the other, completely breaks down in the current context. We thus do not know how to extend their techniques to LDP.

Much more generally, we note that LIP is closely related to the Graph Isomorphism Problem (GI). For example, both problems are in SZK but not known to be in P (although recent work on algorithms for GI has been quite exciting [6]!), and GI reduces to LIP [32]. Therefore, LDP is qualitatively similar to the Approximate Graph Isomorphism Problem, which was studied by Arora, Frieze, and Kaplan [4], who showed an upper bound, and Arvind, Köbler, Kuhnert, and Vasudev [5], who proved both upper and lower bounds. In particular, [5] showed that various versions of this problem are NP-hard to approximate to within a constant factor. Qualitatively, these hardness results are similar to our Theorem 5.

1.3 Conclusions and Open Problems

In conclusion, we introduce the Lattice Distortion Problem and show a connection between LDP and the notion of Seysen-reduced bases. We use this connection to derive time-approximation trade-offs for LDP. We also prove approximation hardness for GapLDP, showing a qualitative difference with LIP (which is unlikely to be NP-hard under reasonable complexity theoretic assumptions).

One major open question is what the correct bound in Theorem 3 is. In particular, there are no known families of lattices for which the Seysen condition number is provably superpolynomial, and hence it is possible that $S(\mathcal{L}) = \mathrm{poly}(n)$ for any n-dimensional lattice $\mathcal{L}$. A better bound would immediately improve our Theorem 2 and give a better approximation factor for GapLDP.

We also note that all of our algorithms solve LDP for arguably very large approximation factors $n^{\Omega(\log n)}$. We currently do not even know whether there exists a fixed-dimension polynomial-time algorithm for γ-LDP for any $\gamma = n^{o(\log n)}$. The main problem here is that we do not have any good characterization of nearly optimal distortion mappings between lattices.

Organization. In Section 2, we present necessary background material. In Section 3, we give our approximations for lattice distortion, proving Theorems 2 and 4. In Section 4, we give the hardness for lattice distortion, proving Theorem 5. In Section 5, we give some illustrative example instances of lattice distortion.

2 Preliminaries

For $\mathbf{x} \in \mathbb{R}^n$, we write $\|\mathbf{x}\|$ for the Euclidean norm of $\mathbf{x}$. We omit any mention of the bit length in the running time of our algorithms. In particular, all of our algorithms take as input vectors in $\mathbb{Q}^n$ and run in time $f(n) \cdot \mathrm{poly}(m)$ for some f, where m is the maximal bit length of an input vector. We therefore suppress the factor of $\mathrm{poly}(m)$.

2.1 Lattices

The i^{th} *successive minimum* of a lattice $\mathcal{L}$ is defined as $\lambda_i(\mathcal{L}) = \inf \{r > 0 : \dim(\mathrm{span}(rB_2^n \cap \mathcal{L})) \geq i\}$. That is, the first successive minimum is the length of the shortest non-zero lattice vector, the second successive minimum is the length of the shortest lattice vector which is linearly independent of a vector achieving the first, and so on. When $\mathcal{L}$ is clear from context, we simply write λ_i.

The *dual lattice* of $\mathcal{L}$ is defined as $\mathcal{L}^* = \{\mathbf{x} \in \mathbb{R}^n : \forall \mathbf{y} \in \mathcal{L}\ \langle \mathbf{x}, \mathbf{y} \rangle \in \mathbb{Z}\}$. If $\mathcal{L} = \mathcal{L}(B)$ then $\mathcal{L}^* = \mathcal{L}(B^*)$ where $B^* = B^{-T}$, the inverse transpose of B. We call $B^* = [\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*]$ the *dual basis* of B, and write $\lambda_i^* = \lambda_i(\mathcal{L}^*)$. We will repeatedly use Banaszczyk's Transference Theorem, which relates the successive minima of a lattice to those of its dual.

▶ **Theorem 6** (Banaszczyk's Transference Theorem [7]). *For every rank n lattice $\mathcal{L}$ and every $i \in [n]$, $1 \leq \lambda_i(\mathcal{L})\lambda_{n-i+1}(\mathcal{L}^*) \leq n$.*

Given a lattice $\mathcal{L}$, we define the *determinant* of $\mathcal{L}$ as $\det(\mathcal{L}) := |\det(B)|$, where B is a basis with $\mathcal{L}(B) = \mathcal{L}$. Since two bases B, B' of $\mathcal{L}$ differ by a unimodular transformation, we have that $|\det(B)| = |\det(B')|$ so that $\det(\mathcal{L})$ is well-defined.

We sometimes work with lattices that do not have full rank—i.e., lattices generated by d linearly independent vectors $\mathcal{L} = \mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ with $d < n$. In this case, we simply identify $\mathrm{span}(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ with $\mathbb{R}^d$ and consider the lattice to be embedded in this space.

2.2 Linear mappings between lattices

We next characterize linear mappings between lattices in terms of bases.

▶ **Lemma 7.** *Let $\mathcal{L}_1, \mathcal{L}_2$ be full-rank lattices. Then a mapping $T : \mathcal{L}_1 \to \mathcal{L}_2$ is bijective and linear if and only if $T = BA^{-1}$ for some bases A, B of $\mathcal{L}_1, \mathcal{L}_2$ respectively. In particular, for any basis A of $\mathcal{L}_1$, $T(A)$ is a basis of $\mathcal{L}_2$.*

Proof. We first show that such a mapping is a bijection from $\mathcal{L}_1$ to $\mathcal{L}_2$. Let $T = BA^{-1}$ where $A = [\mathbf{a}_1, \dots, \mathbf{a}_n]$ and $B = [\mathbf{b}_1, \dots, \mathbf{b}_n]$ are bases of $\mathcal{L}_1, \mathcal{L}_2$ respectively. Because T has full rank, it is injective as a mapping from $\mathbb{R}^n$ to $\mathbb{R}^n$, and it is therefore injective as a mapping from $\mathcal{L}_1$ to $\mathcal{L}_2$. We have that for every $\mathbf{w} \in \mathcal{L}_2$, $\mathbf{w} = \sum_{i=1}^n c_i \mathbf{b}_i$ with $c_i \in \mathbb{Z}$. Let $\mathbf{v} = \sum_{i=1}^n c_i \mathbf{a}_i \in \mathcal{L}_1$. Then, $T(\mathbf{v}) = T(\sum_{i=1}^n c_i \mathbf{a}_i) = \sum_{i=1}^n c_i \mathbf{b}_i = \mathbf{w}$. Therefore, T is a bijection from $\mathcal{L}_1$ to $\mathcal{L}_2$.

We next show that any linear map T with $T(\mathcal{L}_1) = \mathcal{L}_2$ must have this form. Let $A = [\mathbf{a}_1, \dots, \mathbf{a}_n]$ be a basis of $\mathcal{L}_1$, and let $B = T(A)$. We claim that $B = [\mathbf{b}_1, \dots, \mathbf{b}_n]$ is a basis of $\mathcal{L}_2$.

Let $\mathbf{w} \in \mathcal{L}_2$. Because T is a bijection between $\mathcal{L}_1$ and $\mathcal{L}_2$, there exists $\mathbf{v} \in \mathcal{L}_1$ such that $T\mathbf{v} = w$. Using the definition of a basis and the linearity of T,

$$\mathbf{w} = T\mathbf{v} = T\big(\sum_{i=1}^n c_i \mathbf{a}_i\big) = \sum_{i=1}^n c_i \mathbf{b}_i,$$

for some $c_1, \dots, c_n \in \mathbb{Z}$. Because $\mathbf{w}$ was picked arbitrarily, it follows that B is a basis of $\mathcal{L}_2$. ◀

2.3 Seysen's condition number $S(B)$

Seysen shows how to take any basis with relatively low multiplicative drop in its Gram-Schmidt vectors and convert it into a basis with relatively low $S(B) = \max_i \|\mathbf{b}_i\| \|\mathbf{b}_i^*\|$ [30]. By combining this with Gama and Nguyen's slide reduction technique [10], we obtain the following result.

▶ **Theorem 8.** *For every* $\log n \leq k \leq n$ *there exists an algorithm that takes a lattice* $\mathcal{L}$ *as input and computes a basis* B *of* $\mathcal{L}$ *with* $S(B) \leq k^{O(n/k + \log k)}$ *in time* $2^{O(k)}$.

In particular, applying Seysen's procedure to slide-reduced bases suffices. We include a proof of Theorem 8 and a high-level description of Seysen's procedure in Section 2.6.

2.4 The Lattice Distortion Problem

▶ **Definition 9.** For any $\gamma = \gamma(n) \geq 1$, the γ-Lattice Distortion Problem (γ-LDP) is the search problem defined as follows. The input consists of two lattices $\mathcal{L}_1, \mathcal{L}_2$ (represented by bases $B_1, B_2 \in \mathbb{Q}^{n \times n}$). The goal is to output a matrix $T \in \mathbb{R}^{n \times n}$ such that $T(\mathcal{L}_1) = \mathcal{L}_2$ and $\kappa(T) \leq \gamma \cdot \mathcal{D}(\mathcal{L}_1, \mathcal{L}_2)$.

▶ **Definition 10.** For any $\gamma = \gamma(n) \geq 1$, the γ-GapLDP is the promise problem defined as follows. The input consists of two lattices $\mathcal{L}_1, \mathcal{L}_2$ (represented by bases $B_1, B_2 \in \mathbb{Q}^{n \times n}$) and a number $c \geq 1$. The goal is to decide between a 'YES' instance where $\mathcal{D}(\mathcal{L}_1, \mathcal{L}_2) \leq c$ and a 'NO' instance where $\mathcal{D}(\mathcal{L}_1, \mathcal{L}_2) > \gamma \cdot c$.

2.5 Complexity of LDP

We show some basic facts about the complexity of GapLDP. First, we show that the Lattice Isomorphism Problem (LIP) corresponds to the special case of GapLDP where $c = 1$. LIP takes bases of $\mathcal{L}_1, \mathcal{L}_2$ as input and asks if there exists an orthogonal linear transformation O such that $O(\mathcal{L}_1) = \mathcal{L}_2$. Haviv and Regev [15] show that there exists an $n^{O(n)}$-time algorithm for LIP, and that LIP is in the complexity class SZK.

▶ **Lemma 11.** *There is a polynomial-time reduction from LIP to* 1*-GapLDP.*

Proof. Let $\mathcal{L}_1, \mathcal{L}_2$ be an LIP instance. First check that $\det(\mathcal{L}_1) = \det(\mathcal{L}_2)$. If not, then output a trivial 'NO' instance of 1-GapLDP. Otherwise, map the LIP instance to the 1-GapLDP instance with the same input bases and $c = 1$. For any $T : \mathcal{L}_1 \to \mathcal{L}_2$, we must have $\det(T) = 1$, and therefore $\kappa(T) = 1$ if and only if $\|T\| = \|T^{-1}\| = 1$. So, this is a 'YES' instance of GapLDP if and only if $\mathcal{L}_1, \mathcal{L}_2$ are isomorphic. ◀

▶ **Lemma 12.** 1-*GapLDP is in* NP.

Proof. Let $I = (\mathcal{L}_1, \mathcal{L}_2, c)$ be an instance of GapLDP, and let s be the length of I. We will show that for a 'YES' instance, there are bases A, B of $\mathcal{L}_1, \mathcal{L}_2$ respectively such that $T = BA^{-1}$ requires at most poly(s) bits to specify and $\kappa(T) \le c$. Assume without loss of generality that $\mathcal{L}_1, \mathcal{L}_2 \subseteq \mathbb{Z}^n$. Otherwise, scale the input lattices to achieve this at the expense of a factor s blow-up in input size.

To satisfy $\|T\| \|T^{-1}\| \le c$, we must have that $|t_{ij}| \le \|T\| \le c \cdot \det(\mathcal{L}_2)/\det(\mathcal{L}_1) \le c \cdot \det(\mathcal{L}_2)$ for each entry t_{ij} of T. By Cramer's rule, each entry of A^{-1} and hence T will be an integer multiple of $\frac{1}{\det \mathcal{L}_1}$, so we can assume without loss of generality that the denominator of each entry of T is $\det \mathcal{L}_1$.

Combining these bounds and applying Hadamard's inequality, we get that $|t_{ij}|$ takes at most

$$\log(c \cdot \det(\mathcal{L}_1)\det(\mathcal{L}_2)) \le \log\Big(c \cdot \prod_{i=1}^{n} \|\mathbf{a}_i\| \prod_{i=1}^{n} \|\mathbf{b}_i\|\Big)$$

bits to specify. Accounting for the sign of each t_{ij}, it follows that T takes at most $n^2 \cdot \log(2c \cdot \prod_{i=1}^{n} \|\mathbf{a}_i\| \|\mathbf{b}_i\|) \le n^2 \cdot (s+1)$ bits to specify. ◀

We remark that we can replace c with the quantity $n^{O(\log n)} M(\mathcal{L}_1, \mathcal{L}_2) M(\mathcal{L}_2, \mathcal{L}_1)$ (as given by the upper bound in Theorem 1) in the preceding argument to obtain an upper bound on the distortion of an *optimal* mapping T that does not depend on c.

2.6 Basis reduction

In this section, we define various notions of basis reductions and show how to use them to prove Theorem 8.

For a basis $B = [\mathbf{b}_1, \dots, \mathbf{b}_n]$, we write $\pi_i^{(B)} := \pi_{\{\mathbf{b}_1, \dots, \mathbf{b}_{i-1}\}^\perp}$ to represent projection onto the subspace $\{\mathbf{b}_1, \dots, \mathbf{b}_{i-1}\}^\perp$. We then define the *Gram-Schmidt* orthogonalization of B, $(\tilde{b}_1, \dots, \tilde{b}_n)$ as $\tilde{b}_i = \pi_i^{(B)}(\mathbf{b}_i)$. By construction the vectors $\tilde{b}_1, \dots, \tilde{b}_n$ are orthogonal, and each $\mathbf{b}_i$ is a linear combination of $\tilde{b}_1, \dots, \tilde{b}_i$. We define $\mu_{ij} = \frac{\langle \mathbf{b}_i, \tilde{b}_j \rangle}{\langle \tilde{b}_j, \tilde{b}_j \rangle}$.

We define the QR-decomposition of a full-rank matrix B as $B = QR$ where Q has orthonormal columns, and R is upper triangular. The QR-decomposition of a matrix is unique, and can be computed efficiently by applying Gram-Schmidt orthogonalization to the columns of B.

Unimodular matrices, denoted $GL(n, \mathbb{Z})$, form the multiplicative group of $n \times n$ matrices with integer entries and determinant ± 1.

▶ **Fact 13.** $\mathcal{L}(B) = \mathcal{L}(B')$ *if and only if there exists* $U \in GL(n, \mathbb{Z})$ *such that* $B' = B \cdot U$.

Based on this, a useful way to view basis reduction is as right-multiplication by unimodular matrices.

2.6.1 Slide-reduced bases

A very strong notion of basis reduction introduced by Korkine and Zolotareff [18] gives one way of formalizing what it means to be a "shortest-possible" lattice basis.

▶ **Definition 14** ([18], Definition 1 in [30]). Let B be a basis of $\mathcal{L}$. $B = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$ is HKZ (Hermite-Korkine-Zolotareff) reduced if

1. $\forall j < i$, $|\mu_{ij}| \leq \frac{1}{2}$;
2. $\|b_1\| = \lambda_1(\mathcal{L}(B))$; and
3. if $n > 1$, then $[\pi_2^{(B)}(\mathbf{b}_2), \ldots, \pi_2^{(B)}(\mathbf{b}_n)]$ is an HKZ basis of $\pi_2^{(B)}(\mathcal{L})$.

By definition, the first vector $\mathbf{b}_1$ in an HKZ basis is a shortest vector in the lattice. Furthermore, computing an HKZ basis can be achieved by making n calls to an SVP oracle. So, the two problems have the same time complexity up to a factor of n. In particular, computing HKZ bases is NP-hard.

Gama and Nguyen (building on the work of Schnorr [29]) introduced the notion of slide-reduced bases [10], which can be thought of as a relaxed notion of HKZ bases that can be computed more efficiently.

▶ **Definition 15** ([10, Definition 1]). Let B be a basis of $\mathcal{L} \subset \mathbb{Q}^n$ and $\varepsilon > 0$. We say that B is ε-DSVP (dual SVP) reduced if its corresponding dual basis $[\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*]$ satisfies $\|\mathbf{b}_n^*\| \leq (1+\varepsilon) \cdot \lambda_1(\mathcal{L}^*)$.

Then, for $k \geq 2$ an integer dividing n, we say that $B = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$ is (ε, k)-slide reduced if

1. $\forall j < i$, $|\mu_{ij}| \leq \frac{1}{2}$;
2. $\forall 0 \leq i \leq n/k - 1$, the "projected truncated basis" $[\pi_{ik+1}^{(B)}(\mathbf{b}_{ik+1}), \ldots, \pi_{ik+1}^{(B)}(\mathbf{b}_{ik+k})]$ is HKZ reduced; and
3. $\forall 0 \leq i \leq n/k-2$, the "shifted projected truncated basis" $[\pi_{ik+2}^{(B)}(\mathbf{b}_{ik+2}), \ldots, \pi_{ik+2}^{(B)}(\mathbf{b}_{ik+k+1})]$ is ε-DSVP reduced.

▶ **Theorem 16** ([10]). *There is an algorithm that takes as input a lattice $\mathcal{L} \subset \mathbb{Q}^n$, $\varepsilon > 0$, and integer $k \geq \log n$ dividing n and outputs a (k, ε)-slide-reduced basis of $\mathcal{L}$ in time* $\mathrm{poly}(1/\varepsilon) \cdot 2^{O(k)}$.

We will be particularly concerned with the ratios between the length of the Gram-Schmidt vectors of a given basis. We prefer bases whose Gram-Schmidt vectors do not "decay too quickly," and we measure this decay by

$$\eta(B) = \max_{i \leq j} \frac{\|\tilde{b}_i\|}{\|\tilde{b}_j\|} \ .$$

Previous work bounded $\eta(B)$ for HKZ-reduced bases as follows.

▶ **Theorem 17** ([19, Proposition 4.2]). *For any HKZ-reduced basis B over $\mathbb{Q}^n$, $\eta(B) \leq n^{O(\log n)}$.*

Using Theorem 17 and some of the results in [10] we get a bound on $\eta(B)$ for slide-reduced bases.

▶ Proposition 18. For any $\log n \leq k \leq n$, there is an algorithm that takes as input a lattice $\mathcal{L} \subset \mathbb{Q}^n$ and outputs a basis B of $\mathcal{L}$ such that $\eta(B) \leq k^{O(n/k + \log k)}$. Furthermore, the algorithm runs in time $2^{O(k)}$.

See the full version of this paper for a proof of Proposition 18.

2.6.2 Seysen bases

Although slide-reduced bases B consist of short vectors and have bounded $\eta(B)$, they make only weak guarantees about the length of vectors in the dual basis B^*. Of course, one way to compute a basis whose dual will contain short dual basis is short is to simply compute B such that B^* is a suitably reduced basis of $\mathcal{L}^*$. Such a basis B is called a dual-reduced basis, and sees use in applications such as [15].

However, we would like to compute a basis such that the vectors in B and B^* are both short, which Seysen addressed in his work [30]. Seysen's main result finds a basis B such that both B and B^* are short by dividing this problem into two subproblems. The first involves finding a basis with small $\eta(B)$, as in Section 2.6.1. The second subproblem, discussed in [30, Section 3], involves conditioning unipotent matrices. Let $N(n, \mathbb{R})$ be the multiplicative group of unipotent $n \times n$-matrices. That is, a matrix $A \in N(n, \mathbb{R})$ if $a_{ii} = 1$ and $a_{ij} = 0$ for $i > j$ (i.e., A is upper triangular and has ones on the main diagonal). Let $N(n, \mathbb{Z})$ be the subgroup of $N(n, \mathbb{R})$ with integer entries. Because $N(n, \mathbb{Z})$ is a subset of $GL(n, \mathbb{Z})$, we trivially have that $\mathcal{L}(B) = \mathcal{L}(B \cdot U)$ for every $U \in N(n, \mathbb{Z})$.

Let $\|B\|_\infty := \max_{i,j \in [n]} |b_{ij}|$ denote the largest magnitude of an entry in B. We follow Seysen [30] and define $S'(B) = \max\{\|B\|_\infty, \|B^{-1}\|_\infty\}$. We also let

$$\zeta(n) = \sup_{A \in N(n,\mathbb{R})} \Big\{ \inf_{U \in N(n,\mathbb{Z})} \{S'(A \cdot U)\} \Big\}.$$

▶ **Theorem 19** ([30, Prop. 5 and Thm. 6]). *There exists an algorithm* Seysen *that takes as input $A \in N(n, \mathbb{R})$ and outputs $A \cdot U$ where $U \in N(n, \mathbb{Z})$ and $S'(A \cdot U) \le n^{O(\log n)}$ in time $O(n^3)$. In particular, $\zeta(n) \le n^{O(\log n)}$.*

Let $B = QR$ be a QR-decomposition of B. We may further decompose R as $R = DR'$, where $d_{ii} = \|\tilde{b}_i\|$ and

$$r'_{ij} = \begin{cases} 0 & \text{if } j < i, \\ 1 & \text{if } j = i, \\ \mu_{ji} & \text{if } j > i. \end{cases}$$

In particular, note that $R' \in N(n, \mathbb{R})$. It is easy to see that $\eta(B)$ controls $\|D\|\|D^{-1}\|$. On the other hand, using the bound on $\zeta(n)$, we can always multiply B on the right by $U \in N(n, \mathbb{Z})$ to control the size of $\|R'\|\|R'^{-1}\|$. Roughly speaking, these two facts imply Theorem 20.

▶ **Theorem 20** ([30, Theorem 7]). *Let B =* Seysen*(B') where B' is a matrix. Then $S(B) \le n \cdot \eta(B') \cdot \zeta(n)^2$.*

Proof of Theorem 8. Let $B =$ Seysen(B'), where B' is a basis as computed in Proposition 18. We then have that

$$\begin{aligned} S(B) &\le n \cdot \eta(B') \cdot \zeta(n)^2 && \text{(by Theorem 20)} \\ &\le n \cdot k^{O(n/k + \log k)} \cdot \zeta(n)^2 && \text{(by Proposition 18)} \\ &\le n \cdot k^{O(n/k + \log k)} \cdot (n^{O(\log n)})^2 && \text{(by Theorem 19)} \\ &\le k^{O(n/k + \log k)}. \end{aligned}$$

We can compute B' in $2^{O(k)}$ time using Proposition 18. Moreover, by Theorem 19, Seysen runs in $O(n^3)$ time. Therefore the algorithm runs in $2^{O(k)}$ time. ◀

3 Approximating lattice distortion

In this section, we show how to compute low-distortion mappings between lattices by using bases with low $S(B)$.

3.1 Basis length bounds in terms of $S(B)$

Call a basis $B = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$ *sorted* if $\|\mathbf{b}_1\| \le \cdots \le \|\mathbf{b}_n\|$. Clearly, $\|\mathbf{b}_i\| / \lambda_i \ge 1$ for a sorted basis B. Note that sorting B does not change $S(B)$, since $S(\cdot)$ is invariant under permutations of the basis vectors.

A natural way to quantify the "shortness" of a lattice basis is to upper bound $\|\mathbf{b}_k\| / \lambda_k$ for all $k \in [n]$. For example, [19] shows that $\|\mathbf{b}_k\| / \lambda_k \le \sqrt{n}$ when B is an HKZ basis. We prove a characterization of Seysen-reduced bases, showing that *both* the primal basis vectors and the dual basis vectors are not much longer than the successive minima. Namely, we show that $S(B)$ is an upper bound on both $\|\mathbf{b}_k\| / \lambda_k$ and $\|\mathbf{b}_k^*\| / \lambda_{n-k+1}^*$ for sorted bases B. This characterization is key to bounding the distortion between two lattices, and it might be of independent interest.

▶ **Lemma 21.** *Let B be a sorted basis of $\mathcal{L}$. Then $\max_{k \in [n]} \|\mathbf{b}_k\| / \lambda_k \le S(B)$.*

Proof. For every $k \in [n]$, we have

$$\begin{aligned}
\|\mathbf{b}_k\| / \lambda_k &\le \|\mathbf{b}_k\| \lambda_{n-k+1}^* && \text{(by the lower bound in Theorem 6)} \\
&\le \|\mathbf{b}_k\| \max_{i \in \{k,\ldots,n\}} \|\mathbf{b}_i^*\| && \text{(the } \mathbf{b}_i^* \text{ are linearly independent)} \\
&\le \max_{i \in \{k,\ldots,n\}} \|\mathbf{b}_i\| \|\mathbf{b}_i^*\| && (B \text{ is sorted}) \\
&\le S(B).
\end{aligned}$$

◀

▶ **Lemma 22.** *Let B be a sorted basis of $\mathcal{L}$. Then $\max_{k \in [n]} \|\mathbf{b}_k^*\| / \lambda_{n-k+1}^* \le S(B)$.*

Proof. For every $k \in [n]$, we have

$$\frac{\|\mathbf{b}_k^*\|}{\lambda_{n-k+1}^*} \le \frac{\|\mathbf{b}_k\| \|\mathbf{b}_k^*\|}{\lambda_k \lambda_{n-k+1}^*} \le \max_{i \in [n]} \|\mathbf{b}_i\| \|\mathbf{b}_i^*\| = S(B).$$

The first inequality follows from the assumption that B is sorted, and the second follows from the lower bound in Theorem 6. ◀

3.2 Approximating LDP using Seysen bases

In this section, we bound the distortion $\mathcal{D}(\mathcal{L}_1, \mathcal{L}_2)$ between lattices $\mathcal{L}_1, \mathcal{L}_2$. The upper bound is constructive and depends on $S(B_1), S(B_2)$, which naturally leads to Theorem 4.

▶ **Lemma 23.** *Let $A = [\mathbf{a}_1, \ldots, \mathbf{a}_n]$ and $B = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$ be sorted bases of $\mathcal{L}_1, \mathcal{L}_2$ respectively. Then,*

$$\|BA^{-1}\| \le n^2 S(A) S(B) M(\mathcal{L}_1, \mathcal{L}_2).$$

Proof.

$$\begin{aligned}
\|BA^{-1}\| &= \Big\| \sum_{i=1}^{n} \mathbf{b}_i (\mathbf{a}_i^*)^T \Big\| \\
&\le \sum_{i=1}^{n} \|\mathbf{b}_i (\mathbf{a}_i^*)^T\| && \text{(by triangle inequality)} \\
&= \sum_{i=1}^{n} \|\mathbf{b}_i\| \, \|\mathbf{a}_i^*\| \\
&\le n \max_{i \in [n]} \|\mathbf{b}_i\| \, \|\mathbf{a}_i^*\| \\
&\le nS(B) \max_{i \in [n]} \lambda_i(\mathcal{L}_2) \, \|\mathbf{a}_i^*\| && \text{(by Lemma 21)} \\
&\le nS(A)S(B) \max_{i \in [n]} \lambda_i(\mathcal{L}_2) \lambda_{n-i+1}^*(\mathcal{L}_1) && \text{(by Lemma 22)} \\
&\le n^2 S(A)S(B)M(\mathcal{L}_1, \mathcal{L}_2). && \text{(by Theorem 6)}
\end{aligned}$$

◀

Proof of Theorem 2. Note that by definition there always exist bases B_1, B_2 of $\mathcal{L}_1, \mathcal{L}_2$ respectively achieving $S(B_i) = S(\mathcal{L}_i)$. Therefore, applying Lemma 23 twice to bound both $\|B_2 B_1^{-1}\|$ and $\|B_1 B_2^{-1}\|$, we get the upper bound.

For the lower bound, let $\mathbf{v}_1, \ldots, \mathbf{v}_n \in \mathcal{L}_1$ be linearly independent vectors such that $\|\mathbf{v}_i\| = \lambda_i(\mathcal{L}_1)$ for every i. Then, for every i,

$$\lambda_i(\mathcal{L}_2) \le \max_{j \in [i]} \|T\mathbf{v}_j\| \le \|T\| \max_{j \in [i]} \|\mathbf{v}_j\| = \|T\| \, \lambda_i(\mathcal{L}_1).$$

Rearranging, we get that $\lambda_i(\mathcal{L}_2)/\lambda_i(\mathcal{L}_1) \le \|T\|$. This holds for arbitrary i, so in particular $\max_{i \in [n]} \lambda_i(\mathcal{L}_2)/\lambda_i(\mathcal{L}_1) = M(\mathcal{L}_1, \mathcal{L}_2) \le \|T\|$. The same computation with $\mathcal{L}_1, \mathcal{L}_2$ reversed shows that $M(\mathcal{L}_2, \mathcal{L}_1) \le \|T^{-1}\|$. Multiplying these bounds together implies the lower bound in the theorem statement. ◀

We can now prove Theorem 4.

Proof of Theorem 4. Let $(\mathcal{L}_1, \mathcal{L}_2)$ be an instance of LDP. For $i = 1, 2$, compute a basis B_i of $\mathcal{L}_i$ using the algorithm described in Theorem 8 with parameter k. We have that $S(B_i) \le k^{O(n/k + \log k)}$. This computation takes $2^{O(k)}$ time. The algorithm then simply outputs $T = B_2 B_1^{-1}$.

By Lemma 23 and the upper bounds on $S(B_i)$, we get that $\kappa(T) \le k^{O(n/k + \log k)} \cdot M(\mathcal{L}_1, \mathcal{L}_2) \cdot M(\mathcal{L}_2, \mathcal{L}_1)$. This is within a factor of $k^{O(n/k + \log k)} \cdot n^{O(\log n)} = k^{O(n/k + \log k)}$ of $\mathcal{D}(\mathcal{L}_1, \mathcal{L}_2)$ by Theorem 1. So, the algorithm is correct. ◀

4 Hardness of LDP

In this section, we prove the hardness of γ-GapLDP. (See Theorem 31.) Our reduction works in two steps. First, we show how to use an oracle for GapLDP to solve a variant of GapCVP that we call γ-GapCVP$^\alpha$. (See Definition 24 and Theorem 26.) Given a CVP instance consisting of a lattice $\mathcal{L}$ and a target vector $\mathbf{t}$, our idea is to compare "$\mathcal{L}$ with $\mathbf{t}$ appended to it" to "$\mathcal{L}$ with an extra orthogonal vector appended to it." (See Eq. (3).) We show that, if $\operatorname{dist}(\mathbf{t}, \mathcal{L})$ is small, then these lattices will be similar. On the other hand, if (1) $\operatorname{dist}(k\mathbf{t}, \mathcal{L})$ is large for all non-zero integers k, and (2) $\lambda_1(\mathcal{L})$ is not too small; then the two lattices must be quite dissimilar.

We next show that γ-GapCVP$^\alpha$ is as hard as GapSVP. (See Theorem 29.) This reduction is a variant of the celebrated reduction of [13]. It differs from the original in that it "works in base p" instead of in base two, and it "adds an extra coordinate to $\mathbf{t}$." We show that this is sufficient to satisfy the promises required by γ-GapCVP$^\alpha$.

Both reductions are relatively straightforward.

4.1 Reduction from a variant of CVP

▶ **Definition 24.** For any $\gamma = \gamma(n) \geq 1$ and $\alpha = \alpha(n) > 0$, γ-GapCVP$^\alpha$ is the promise problem defined as follows. The input is a lattice $\mathcal{L} \subset \mathbb{Q}^n$, a target $\mathbf{t} \in \mathbb{Q}^n$, and a distance $d > 0$. It is a 'YES' instance if $\mathrm{dist}(\mathbf{t}, \mathcal{L}) \leq d$ and a 'NO' instance if $\mathrm{dist}(k\mathbf{t}, \mathcal{L}) > \gamma d$ for all non-zero integers k *and* $d < \alpha \cdot \lambda_1(\mathcal{L})$.

We will need the following characterization of the operator norm of a matrix in terms of its behavior over a lattice. Intuitively, this says that "a lattice has a point in every direction."

▶ **Fact 25.** *For any matrix $A \in \mathbb{R}^{n\times n}$ and (full-rank) lattice $\mathcal{L} \subset \mathbb{R}^n$,*

$$\|A\| = \sup_{\mathbf{y}\in\mathcal{L}\setminus\{0\}} \frac{\|A\mathbf{y}\|}{\|\mathbf{y}\|} \ .$$

Proof. It suffices to note that, for any $\mathbf{x} \in \mathbb{R}^n$ with $\|\mathbf{x}\| = 1$ and any full-rank lattice $\mathcal{L} \subset \mathbb{R}^n$, there is a sequence $\mathbf{y}_1, \mathbf{y}_2, \ldots$ of vectors $\mathbf{y}_i \in \mathcal{L}$ such that

$$\lim_{m\to\infty} \frac{\mathbf{y}_m}{\|\mathbf{y}_m\|} = \mathbf{x} \ .$$

Indeed, this follows immediately from the fact that the rationals are dense in the reals. ◀

▶ **Theorem 26.** *For any $\gamma = \gamma(n) \geq 1$, there is an efficient reduction from γ'-*GapCVP$^{1/\gamma'}$ *to γ-GapLDP, where $\gamma' = O(\gamma)$.*

Proof. On input $\mathcal{L} \subset \mathbb{Q}^n$ with basis $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$, $\mathbf{t} \in \mathbb{Q}^n$, and $d > 0$, the reduction behaves as follows. Let $\mathcal{L}_1 := \mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_n, r \cdot \mathbf{e}_{n+1})$ with $r > 0$ to be set in the analysis. Let $\mathcal{L}_2 := \mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_n, \mathbf{t} + r \cdot \mathbf{e}_{n+1})$. I.e.,

$$\mathcal{L}_1 = \mathcal{L}\begin{pmatrix} B & \mathbf{0} \\ 0 & r \end{pmatrix} \qquad\qquad \mathcal{L}_2 = \mathcal{L}\begin{pmatrix} B & \mathbf{t} \\ 0 & r \end{pmatrix} . \tag{3}$$

(Formally, we must embed the $\mathbf{b}_i$ and $\mathbf{t}$ in $\mathbb{Q}^{n+1}$ under the natural embedding, but we ignore this for simplicity.) The reduction then calls its γ-GapLDP oracle with input $\mathcal{L}_1$, $\mathcal{L}_2$, and $c > 0$ to be set in the analysis and outputs its response.

It is clear that the reduction runs in polynomial time. Suppose that $\mathrm{dist}(\mathbf{t}, \mathcal{L}) \leq d$. We note that $\mathcal{L}_2$ does not change if we shift $\mathbf{t}$ by a lattice vector. So, we may assume without loss of generality that $\mathbf{0}$ is a closest lattice vector to $\mathbf{t}$ and therefore $\|\mathbf{t}\| \leq d$.

Let $B_1 := [\mathbf{b}_1, \ldots, \mathbf{b}_n, r \cdot \mathbf{e}_{n+1}]$ and $B_2 := [\mathbf{b}_1, \ldots, \mathbf{b}_n, \mathbf{t} + r \cdot \mathbf{e}_{n+1}]$ be the bases from the reduction. It suffices to show that $\kappa(B_2 B_1^{-1})$ is small. Indeed, for any $\mathbf{y} \in \mathcal{L}_1$, we can write $\mathbf{y} = (\mathbf{y}', kr)$ for some $k \in \mathbb{Z}$ and $\mathbf{y}' \in \mathcal{L}$. Then, we have

$$\|B_2 B_1^{-1}\mathbf{y}\| = \|(\mathbf{y}' + k\mathbf{t}, kr)\| \leq \|(\mathbf{y}', kr)\| + |k|\|\mathbf{t}\| \leq (1 + d/r)\|\mathbf{y}\| \ .$$

Similarly, $\|B_2B_1^{-1}\mathbf{y}\| \geq \|\mathbf{y}\| - |k|\|\mathbf{t}\| \geq (1-d/r)\|\mathbf{y}\|$. Therefore, by Fact 25, $\kappa(B_2B_1^{-1}) \leq (1+d/r)/(1-d/r)$. So, we take $c := (1+d/r)/(1-d/r)$, and the oracle will therefore output 'YES'.

Now, suppose $\operatorname{dist}(z\mathbf{t}, \mathcal{L}) > 10\gamma d$ for all non-zero integers z, and $\lambda_1(\mathcal{L}) > 10\gamma d$. (I.e., we take $\gamma' = 10\gamma = O(\gamma)$.) Let A be a linear map with $A\mathcal{L}_1 = \mathcal{L}_2$. Note that A has determinant one, so that $\kappa(A) \geq \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|}$ for any $\mathbf{x} \in \mathbb{Q}^{n+1} \setminus \{\mathbf{0}\}$. We have that $A(\mathbf{0}, r) = (\mathbf{y}', kr)$ for some $\mathbf{y}' \in \mathcal{L} + k\mathbf{t}$ and $k \in \mathbb{Z}$. If $k \neq 0$, then $\|A(\mathbf{0}, r)\| \geq \operatorname{dist}(k\mathbf{t}, \mathcal{L}) > 10\gamma d$. So, $\kappa(A) \geq \|A(\mathbf{0}, r)\|/r > 10\gamma d/r$.

If, on the other hand, $k = 0$, then $\mathbf{y}' \in \mathcal{L} \setminus \{\mathbf{0}\}$ and $\|A(\mathbf{0}, r)\| = \|(\mathbf{y}', 0)\| \geq \lambda_1(\mathcal{L}) > 10\gamma d$, so that we again have $\kappa(A) \geq \|A(\mathbf{0}, r)\|/r > 10\gamma d/r$. Taking $r = 2\gamma d$ gives $\kappa(A) > \gamma \cdot c$, so that the oracle will output 'NO', as needed. ◀

4.2 Hardness of This Variant of GapCVP

We recall the definition of (the decision version of) γ-GapSVP.

▶ **Definition 27.** For any $\gamma = \gamma(n) \geq 1$, γ-GapSVP is the promise problem defined as follows: The input is a lattice $\mathcal{L} \subset \mathbb{Q}^n$, and a distance $d > 0$. It is a 'YES' instance if $\lambda_1(\mathcal{L}) \leq d$ and a 'NO' instance if $\lambda_1(\mathcal{L}) > \gamma d$.

Haviv and Regev (building on work of Ajtai, Micciancio, and Khot [2, 22, 17]) proved the following strong hardness result for γ-GapSVP [14].

▶ **Theorem 28** ([14, Theorem 1.1]).
1. *γ-GapSVP is* NP*-hard under randomized polynomial-time reductions for any constant $\gamma \geq 1$. I.e., there is no randomized polynomial-time algorithm for γ-GapSVP unless* NP $\subseteq$ RP.
2. *$2^{\log^{1-\varepsilon} n}$-GapSVP is* NP*-hard under randomized quasipolynomial-time reductions for any constant $\varepsilon > 0$. I.e., there is no randomized polynomial-time algorithm for $2^{\log^{1-\varepsilon} n}$-GapSVP unless* NP $\subseteq$ RTIME$(2^{\mathrm{polylog}(n)})$.
3. *$n^{c/\log\log n}$-GapSVP is* NP*-hard under randomized subexponential-time reductions for some universal constant $c > 0$. I.e., there is no randomized polynomial-time algorithm for $n^{c/\log\log n}$-GapSVP unless* NP $\subseteq$ RSUBEXP $:= \bigcap_{\delta>0}$ RTIME(2^{n^δ}).

In particular, to prove Theorem 5, it suffices to reduce γ'-GapSVP to γ-CVP$^{1/\gamma}$ for $\gamma' = O(\gamma)$.

▶ **Theorem 29.** *For any $1 \leq \gamma = \gamma(n) \leq \mathrm{poly}(n)$, there is an efficient reduction from γ'-GapSVP to γ-*GapCVP$^{1/\gamma}$*, where $\gamma' = \gamma \cdot (1 + o(1))$.*

Proof. Let p be a prime with $10\gamma n \leq p \leq 20\gamma n \leq \mathrm{poly}(n)$. We take $\gamma' = \gamma \cdot (1 + o(1))$ so that

$$\gamma = \frac{\gamma'}{\sqrt{1 - \gamma'^2/(p-1)^2}}.$$

On input a basis $B := [\mathbf{b}_1, \ldots, \mathbf{b}_n]$ for a lattice $\mathcal{L} \subset \mathbb{Q}^n$, and $d > 0$, the reduction behaves as follows. For $i = 1, \ldots, n$, let $\mathcal{L}_i := \mathcal{L}(\mathbf{b}_1, \ldots, p\mathbf{b}_i, \ldots, \mathbf{b}_n)$ be "$\mathcal{L}$ with its ith basis vector multiplied by p." And, for all i and $1 \leq j < p$, let $\mathbf{t}_{i,j} := j\mathbf{b}_i + r\mathbf{e}_{n+1}$, with $r > 0$ to be set in the analysis. For each i, j, the reduction calls its γ-GapCVP$^{1/\gamma}$ oracle on input $\mathcal{L}_i$, $\mathbf{t}_{i,j}$, and $d' := \sqrt{d^2 + r^2}$. Finally, it outputs 'YES' if the oracle answered 'YES' for any query. Otherwise, it outputs 'NO'.

It is clear that the algorithm is efficient. Note that

$$\operatorname{dist}(j\mathbf{b}_i, \mathcal{L}_i) = \min\left\{ \Big\| \sum_{\ell=1}^{n} a_\ell \mathbf{b}_\ell \Big\| \ : \ a_\ell \in \mathbb{Z},\ a_i \equiv j \bmod p \right\} .$$

In particular, $\lambda_1(\mathcal{L}) = \min_{i,j} \operatorname{dist}(j\mathbf{b}_i, \mathcal{L}_i)$.

So, suppose $\lambda_1(\mathcal{L}) \le d$. Then, there must be some i, j such that $\operatorname{dist}(\mathbf{t}_{i,j}, \mathcal{L}_i)^2 \le r^2 + \lambda_1(\mathcal{L})^2 \le r^2 + d^2 = d'^2$. So, the oracle answers 'YES' at least once.

Now, suppose $\lambda_1(\mathcal{L}) > \gamma' d$. Since $\mathcal{L}_i \subset \mathcal{L}$, we have $\lambda_1(\mathcal{L}_i) \ge \lambda_1(\mathcal{L}) > \gamma' d$, and therefore $d < \lambda_1(\mathcal{L}_i)/\gamma' < \lambda_1(\mathcal{L}_i)/\gamma$, as needed. And, by the above observation, we have $\operatorname{dist}(j\mathbf{b}_i, \mathcal{L}_i) \ge \lambda_1(\mathcal{L}) > \gamma' d$ for all $1 \le i \le n$ and $1 \le j < p$. Furthermore, for any integer $1 \le z < p$, we have $\operatorname{dist}(zj\mathbf{b}_i, \mathcal{L}_i) = \operatorname{dist}((zj \bmod p) \cdot \mathbf{b}_i, \mathcal{L}_i) > \gamma' d$, where we have used the fact that p is prime so that $zj \not\equiv 0 \bmod p$. It follows that $\operatorname{dist}(z\mathbf{t}_{i,j}, \mathcal{L}_i) > \operatorname{dist}(zj\mathbf{b}_i, \mathcal{L}_i) > \gamma' d$. And, for $z \ge p$, it is trivially the case that $\operatorname{dist}(z\mathbf{t}_{i,j}, \mathcal{L}_i) \ge zr \ge pr$. Taking $r := \gamma' d/(p-1)$, we have that in both cases

$$\operatorname{dist}(z\mathbf{t}_{i,j}, \mathcal{L}_i) > \gamma' d = \frac{\gamma' d'}{\sqrt{1 - r^2}} = \frac{\gamma' d'}{\sqrt{1 - \gamma'^2/(p-1)^2}} = \gamma d .$$

So, the oracle will always answer 'NO'. ◀

▶ **Corollary 30.** *For any $1 \le \gamma = \gamma(n) \le \operatorname{poly}(n)$, there is an efficient reduction from γ'-GapSVP to γ-GapLDP, where $\gamma' = O(\gamma)$.*

Proof. Combine Theorems 26 and 29. ◀

With this, the proof of our main hardness result is immediate.

▶ **Theorem 31.** *The three hardness results in Theorem 28 hold with GapLDP in place of GapSVP.*

Proof. Combine Theorem 28 with Corollary 30. ◀

5 Some illustrative examples

5.1 Separating distortion from the successive minima

We now show that, for every n, there exists a $\mathcal{L}$ such that $\mathcal{D}(\mathcal{L}, \mathbb{Z}^n) \ge \Omega(\sqrt{n}) \cdot M(\mathcal{L}, \mathbb{Z}^n) \cdot M(\mathbb{Z}^n, \mathcal{L})$. Indeed, it suffices to take any lattice with $\det(\mathcal{L})^{1/n} \le O(n^{-1/2})$ but $\lambda_i(\mathcal{L}) = \Theta(1)$. (This is true for almost all lattices in a certain precise sense. See, e.g., [31].)

▶ **Lemma 32.** *For any $n \ge 1$, there is a lattice $\mathcal{L} \subset \mathbb{Q}^n$ such that $\det(\mathcal{L})^{1/n} \le O(n^{-1/2})$ and $\lambda_i(\mathcal{L}) = \Theta(1)$ for all i.*

▶ Proposition 33. For any $n \ge 1$, there exists a lattice $\mathcal{L} \subset \mathbb{Q}^n$ such that

$$\mathcal{D}(\mathcal{L}, \mathbb{Z}^n) \ge \Omega(\sqrt{n}) \cdot M(\mathcal{L}, \mathbb{Z}^n) \cdot M(\mathbb{Z}^n, \mathcal{L}) .$$

Proof. Let $\mathcal{L} \subset \mathbb{Q}^n$ be any lattice as in Lemma 32. In particular, $M(\mathcal{L}, \mathbb{Z}^n) \cdot M(\mathbb{Z}^n, \mathcal{L}) = O(1)$. However, for any linear map T with $T(\mathcal{L}) = \mathbb{Z}^n$, we of course have

$$\|T\| \ge |\det(T)|^{1/n} = \det(\mathbb{Z}^n)^{1/n} / \det(\mathcal{L})^{1/n} \ge \Omega(\sqrt{n}) .$$

(To see the first inequality, it suffices to recall that $|\det(T)| = \prod \sigma_i$ and $\|T\| = \max \sigma_i$, where the σ_i are the singular values of T.) And, $T^{-1}\mathbf{e}_1$ must be a non-zero lattice vector, so $\|T^{-1}\| \ge \|T^{-1}\mathbf{e}_1\| \ge \lambda_1(\mathcal{L}) \ge \Omega(1)$. Therefore, $\kappa(T) = \|T\|\|T^{-1}\| \ge \Omega(\sqrt{n})$, as needed. ◀

5.2 Non-optimality of HKZ bases for distortion

We show an example demonstrating that mappings between lattices built using HKZ bases are non-optimal in terms of their distortion. Namely, we give a family of $n \times n$ HKZ bases $\{B_n\}$ such that $\mathcal{D}(\mathbb{Z}^n, \mathcal{L}(B_n)) \leq n^{O(\log n)}$, but where the mapping $T = B_n$ from $\mathbb{Z}^n$ to $\mathcal{L}(B_n)$ has exponential distortion. This shows the necessity of using Seysen reduction in addition to HKZ reduction.

▶ **Theorem 34.** *For every $n \geq 1$, there exists an $n \times n$ HKZ basis B such that $\mathcal{D}(\mathbb{Z}^n, \mathcal{L}(B)) \leq n^{O(\log n)}$, but $\kappa(B) \geq \Omega(1.5^n)$.*

Recall that $\|B\|_\infty$ denotes the largest magnitude of an entry in B. It holds that $\|B\|_\infty \leq \|B\| \leq n\,\|B\|_\infty$.

▶ **Lemma 35.** *Let $B = B_n$ denote the $n \times n$ basis defined as*

$$b_{ij} = \begin{cases} 0 & \text{if } j < i, \\ 1 & \text{if } j = i, \\ -\frac{1}{2} & \text{if } j > i. \end{cases}$$

Then B is an HKZ basis and $\kappa(B) = \Omega(1.5^n)$.

Proof. For every basis A, it holds that $\min_{i\in[n]} \|\tilde{\mathbf{a}}_i\| \leq \lambda_1(\mathcal{L}(A))$ (see, e.g., [27]). Note that for $i \geq 0$ the i^{th} Gram-Schmidt vector of $(\pi_k^{(B)}(\mathbf{b}_k), \dots, \pi_k^{(B)}(\mathbf{b}_n))$ is simply $\tilde{b}_{i+k}$. Let $k \in [n]$. We then have that $1 = \min_{i\in[n]} \|\tilde{b}_i\| \leq \lambda_1(\pi_k(\mathcal{L}))$. On the other hand, $\lambda_1(\pi_k(\mathcal{L})) \leq \|\tilde{b}_k\| = \left\|\pi_k^{(B)}(\mathbf{b}_k)\right\| = 1$, implying that $\lambda_1(\pi_k(\mathcal{L}(B))) = 1$. It follows that B is an HKZ basis.

Because $\|B\|_\infty = 1$, it suffices to show that $\|B^{-1}\|_\infty \geq \Omega(1.5^n)$. Let $\mathbf{x}$ denote the nth column of B^{-1}. We must then have that $B\mathbf{x} = e_n$. Because B is upper triangular, we get the following formula by back substitution (see, e.g., [33]):

$$x_j = \begin{cases} 1 & \text{if } j = n, \\ \frac{1}{2}\sum_{k=j+1}^{n} x_k & \text{otherwise.} \end{cases} \tag{4}$$

We therefore have that $x_n = 1, x_{n-1} = \frac{1}{2}$. Using Eq. (4), we get that for $1 \leq m \leq n-2$,

$$x_m = \frac{1}{2}\sum_{k=m+1}^{n} x_k = \frac{1}{2}\cdot x_{m+1} + \frac{1}{2}\cdot\sum_{k=m+2}^{n} x_k = 1.5 \cdot x_{m+1} \; .$$

Applying this formula recursively, we get that $x_m = \frac{1}{2}\cdot 1.5^{n-m-1}$ for $1 \leq m \leq n-1$. ◀

The proof of Theorem 34 follows.

Proof of Theorem 34. Let $B' = B_n$ be an HKZ basis as specified in Lemma 35, and take I_n as the basis of $\mathbb{Z}_n$. Then $\kappa(B' \cdot I_n) = \Omega(1.5^n)$.

On the other hand, let $B = \text{SEYSEN}(B')$. Then, because $\eta(B') = 1$, $S(B) = n^{O(\log n)}$ by Theorem 20. Clearly, $\lambda_i(\mathbb{Z}_n) = 1$ for all $i \in [n]$. On the other hand, $1 \leq \lambda_i(\mathcal{L}(B)) \leq \sqrt{n}$ for all $i \in [n]$. The lower bound holds because $\min \|\tilde{b}_i\| = 1$, and the upper bound comes from the fact that $\|\mathbf{b}_i'\| \leq \sqrt{n}$ for all $i \in [n]$ and the linear independence of the $\mathbf{b}_i'$.[1] It follows that $M(\mathbb{Z}^n, \mathcal{L}(B)) \leq \sqrt{n}$ and $M(\mathcal{L}(B), \mathbb{Z}^n) \leq 1$. Applying Lemma 23 to B and B^{-1}, we then get that $\kappa(B \cdot I_n) \leq n^{O(\log n)}$. ◀

[1] In fact, $\lambda_n(\mathcal{L}(B)) = O(1)$.

Acknowledgements. We thank Oded Regev for pointing us to Seysen's paper and for many helpful conversations. The concise proof of Lemma 22 using the Transference Theorem is due to Michael Walter. We thank Paul Kirchner for suggesting the use of slide-reduced bases, and for identifying a minor bug in an earlier version of this paper.

References

1 Miklós Ajtai. Generating hard instances of lattice problems. In *STOC*, 1996.

2 Miklós Ajtai. The Shortest Vector Problem in L2 is NP-hard for randomized reductions. In *STOC*, 1998. doi:10.1145/276698.276705.

3 Miklós Ajtai, Ravi Kumar, and D. Sivakumar. A sieve algorithm for the Shortest Lattice Vector Problem. In *STOC*, pages 601–610, 2001. doi:10.1145/380752.380857.

4 Sanjeev Arora, Alan Frieze, and Haim Kaplan. A new rounding procedure for the assignment problem with applications to dense graph arrangement problems. *Mathematical Programming*, 2002. doi:10.1007/s101070100271.

5 Vikraman Arvind, Johannes Köbler, Sebastian Kuhnert, and Yadu Vasudev. Approximate Graph Isomorphism. In *Mathematical Foundations of Computer Science*, 2012.

6 L. Babai. Graph Isomorphism in quasipolynomial time, 2016. http://arxiv.org/abs/1512.03547.

7 W. Banaszczyk. New bounds in some transference theorems in the geometry of numbers. *Mathematische Annalen*, 296(1):625–635, 1993. doi:10.1007/BF01445125.

8 Zvika Brakerski and Vinod Vaikuntanathan. Lattice-based FHE as secure as PKE. In *ITCS*, 2014. doi:10.1145/2554797.2554799.

9 J. Conway and N.J.A. Sloane. *Sphere Packings, Lattices and Groups*. Springer New York, 1998.

10 Nicolas Gama and Phong Q. Nguyen. Finding short lattice vectors within Mordell's inequality. In *STOC*, 2008. doi:10.1145/1374376.1374408.

11 Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, 2009.

12 Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, 2008.

13 Oded Goldreich, Daniele Micciancio, Shmuel Safra, and Jean-Paul Seifert. Approximating shortest lattice vectors is not harder than approximating closest lattice vectors. *Information Processing Letters*, 71(2):55–61, 1999. doi:10.1016/S0020-0190(99)00083-6.

14 Ishay Haviv and Oded Regev. Tensor-based hardness of the Shortest Vector Problem to within almost polynomial factors. *Theory of Computing*, 8(23):513–531, 2012. Preliminary version in STOC'07.

15 Ishay Haviv and Oded Regev. On the Lattice Isomorphism Problem. In *SODA*, 2014. doi:10.1137/1.9781611973402.29.

16 Ravi Kannan. Minkowski's convex body theorem and Integer Programming. *Mathematics of Operations Research*, 12(3):pp. 415–440, 1987. URL: http://www.jstor.org/stable/3689974.

17 Subhash Khot. Hardness of approximating the Shortest Vector Problem in lattices. *Journal of the ACM*, 52(5):789–808, September 2005. Preliminary version in FOCS'04.

18 A. Korkine and G. Zolotareff. Sur les formes quadratiques. *Mathematische Annalen*, 6(3):366–389, 1873. doi:10.1007/BF01442795.

19 J. C. Lagarias, Hendrik W. Lenstra Jr., and Claus-Peter Schnorr. Korkin-Zolotarev bases and successive minima of a lattice and its reciprocal lattice. *Combinatorica*, 10(4):333–348, 1990. doi:10.1007/BF02128669.

20 A.K. Lenstra, H.W. Lenstra Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982. doi:10.1007/BF01457454.

21 Hendrik W. Lenstra Jr. and Alice Silverberg. Lattices with symmetry, 2014. `http://arxiv.org/abs/1501.00178`.

22 Daniele Micciancio. The Shortest Vector Problem is NP-hard to approximate to within some constant. *SIAM Journal on Computing*, 30(6):2008–2035, March 2001. Preliminary version in FOCS 1998.

23 Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on Gaussian measures. *SIAM Journal on Computing*, 37(1):267–302, 2007.

24 Daniele Micciancio and Panagiotis Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on Voronoi cell computations. *SIAM J. Comput.*, 42(3):1364–1391, 2013. `doi:10.1137/100811970`.

25 H. Minkowski. *Geometrie der Zahlen.* Number v. 1 in Geometrie der Zahlen. B.G. Teubner, 1910.

26 W. Plesken and B. Souvignier. Computing isometries of lattices. *J. Symbolic Comput.*, 24(3-4):327–334, 1997. Computational algebra and number theory (London, 1993).

27 Oded Regev. Lecture notes from course on "Lattices in Computer Science", 2009. URL: `http://www.cims.nyu.edu/~regev/teaching/lattices_fall_2009/index.html`.

28 Oded Regev. On lattices, Learning with Errors, random linear codes, and cryptography. *Journal of the ACM*, 56(6):Art. 34, 40, 2009. `doi:10.1145/1568318.1568324`.

29 C.P. Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical Computer Science*, 1987. `doi:10.1016/0304-3975(87)90064-8`.

30 Martin Seysen. Simultaneous reduction of a lattice basis and its reciprocal basis. *Combinatorica*, 13(3):363–376, 1993. `doi:10.1007/BF01202355`.

31 Carl Ludwig Siegel. A mean value theorem in geometry of numbers. *Annals of Mathematics*, 46(2):pp. 340–347, 1945. URL: `http://www.jstor.org/stable/1969027`.

32 Mathieu Dutour Sikiric, Achill Schürmann, and Frank Vallentin. Complexity and algorithms for computing Voronoi cells of lattices. *Math. Comput.*, 78(267):1713–1731, 2009. `doi:10.1090/S0025-5718-09-02224-8`.

33 Lloyd N. Trefethen and David Bau III. *Numerical Linear Algebra.* SIAM: Society for Industrial and Applied Mathematics, 1997. ISBN 0898713617.

Plurality Consensus in Arbitrary Graphs: Lessons Learned from Load Balancing

Petra Berenbrink[1], Tom Friedetzky[2], Peter Kling[*3], Frederik Mallmann-Trenn[4], and Chris Wastell[†5]

1 **Simon Fraser University, Burnaby, Canada**
2 **Durham University, Durham, U.K.**
3 **Simon Fraser University, Burnaby, Canada**
4 **Simon Fraser University, Burnaby, Canada; and École normale supérieure, Paris, France**
5 **Durham University, Durham, United Kingdom**

Abstract

We consider *plurality consensus* in networks of n nodes. Initially, each node has one of k opinions. The nodes execute a (randomized) distributed protocol to agree on the *plurality opinion* (the opinion initially supported by the most nodes). In certain types of networks the nodes can be quite cheap and simple, and hence one seeks protocols that are not only time efficient but also simple and space efficient. Typically, protocols depend heavily on the employed communication mechanism, which ranges from sequential (only one pair of nodes communicates at any time) to fully parallel (all nodes communicate with all their neighbors at once) and everything in-between.

We propose a framework to design protocols for a multitude of communication mechanisms. We introduce protocols that solve the plurality consensus problem and are, with probability $1 - o(1)$, both time and space efficient. Our protocols are based on an interesting relationship between plurality consensus and distributed load balancing. This relationship allows us to design protocols that generalize the state of the art for a large range of problem parameters.

1998 ACM Subject Classification C.2.2 Network Protocols

Keywords and phrases Plurality Consensus, Distributed Computing, Load Balancing

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.10

1 Introduction

The objective in the *plurality consensus* problem is to find the so-called *plurality opinion* (i.e., the opinion that is initially supported by the largest subset of nodes) in a network G where, initially, each of the n nodes has one of k opinions. Applications of this problem include distributed computing [20, 30, 31], social networks [29, 17, 28], as well as modeling of biological interactions [16, 15]. All these areas typically demand both very simple and space-efficient protocols. Communication models, however, can vary from anything between simple sequential communication with a single neighbor (often used in biological settings as a simple variant of asynchronous communication [5]) to fully parallel communication where all nodes communicate with all their neighbors simultaneously (like broadcasting models in distributed computing). This diversity turns out to be a major obstacle in algorithm

* Supported in part by the Pacific Institute for the Mathematical Sciences (PIMS).
† Supported in part by the Engineering and Physical Sciences Research Council (EPSRC).

design, since protocols (and their analyses) to a large degree depend upon the employed communication mechanism.

In this paper we present two simple plurality consensus protocols called Shuffle and Balance. Both protocols work in a very general discrete-time communication model. The communication partners are determined by a (possibly randomized) sequence $(\boldsymbol{M}_t)_{t \leq N}$ of *communication matrices*, where we assume[1] N to be some suitably large polynomial in n. That is, nodes u and v can communicate in round t if and only if $\boldsymbol{M}_t[u, v] = 1$. In that case, we call the edge $\{ u, v \}$ *active*; see [6, 32] for related graph models. Our results allow for a wide class of communication patterns (which can even vary over time) as long as the communication matrices have certain "smoothing" properties (cf. Section 2). These smoothing properties are inspired by similar smoothing properties used by Thomas Sauerwald and He Sun [32] for load balancing in the dimension exchange model. In fact, load balancing is the source of inspiration for our protocols. Initially, each node creates a suitably chosen number of tokens labeled with its own opinion. Our Balance protocol then performs discrete load balancing on these tokens, allowing each node to get an estimate on the total number of tokens for each opinion. The Shuffle protocol keeps the number of tokens on every node fixed, but shuffles tokens between communication partners. By keeping track of how many tokens of their own opinion (label) were exchanged in total, nodes gain an estimate on the total (global) number of such tokens. Together with a simple broadcast routine, all nodes can determine the plurality opinion.

The running time of our protocols is the smallest time t for which all nodes have stabilized on the plurality opinion. That is, all nodes have determined the plurality opinion and will not change. This time depends on the network G, the communication pattern $(\boldsymbol{M}_t)_{t \leq N}$, and the initial bias towards the plurality opinion (cf. Section 2). For both protocols we show a strong correlation between their running time, the mixing time of certain random walks and the (related) *smoothing time*, both of which are used in the analysis of recent load balancing results [32]. To give some more concrete examples of our results, let $T \coloneqq \mathrm{O}\left(\log n/(1 - \lambda_2)\right)$, where $1 - \lambda_2$ is the spectral gap of G. If the bias is sufficiently high, then both our protocols Shuffle and Balance determine the plurality opinion in time

- $n \cdot T$ in the *sequential model* (only one pair of nodes communicates per time step);
- $d \cdot T$ in the *balancing circuit model* (communication partners are chosen according to d (deterministic) perfect matchings in a round-robin fashion); and
- T in the *diffusion model* (all nodes communicate with all their neighbors at once).

To the best of our knowledge, these match the best known bounds in the corresponding models. For an arbitrary bias (in particular, *arbitrarily small* bias), the protocols differ in their time and space requirements. More details of our results can be found in Section 1.2.

1.1 Related Work

There is a diverse body of literature that analyzes consensus problems under various models and assumptions. Results differ in the considered topology (e.g., arbitrary or complete), the restrictions on model parameters (e.g., the number of opinions or the *initial bias*[2]), the time model (synchronous or asynchronous), or the required knowledge (e.g., n, maximal degree,

[1] For simplicity and without loss of generality; our protocols run in polynomial time in all considered models.

[2] The bias is $\alpha \coloneqq (n_1 - n_2)/n$, n_1 and n_2 being the support of the most and second most common opinions.

Table 1 Summary of plurality consensus results.

	Arbitrary Graph	Number of Opinions	Required Bias α O-notation	Time O-notation	Model	Space O-notation
Shuffle	✓	arbitrary	arbitrary	$T \cdot t_{\text{mix}}$ $T \cdot \log(n)/(1-\lambda_2)$ (d-reg graph)	sync & async	see Theorem 2
Balance	✓	arbitrary	arbitrary	τ $\log(n)/(1-\lambda_2)$ (d-reg graph)	sync & async	$k \cdot \log(n)$
[26]	✓	arbitrary	arbitrary	$D + \frac{F_2}{n_1^2} \cdot \log(k)$	broadcast	–
[27]	✓	2	arbitrary	n^5	async	1
[21]	✓	2	arbitrary	$\log n/\delta(G, n_1/n)$	async	1
[19]	expander	2	$vol(1) - vol(2) \geq 4\lambda_2^2 \cdot \lvert E \rvert$	$\log(n)$	sync	1
[18]	random d-reg	2	$\sqrt{1/d + d/n}$	$\log(n)$	sync	1
[9]	✗	$\leq n$	$\sqrt{\min\left\{k, \sqrt[3]{\frac{n}{\log(n)}}\right\} \cdot \frac{\log(n)}{n}}$	$\min\left\{k, \sqrt[3]{\frac{n}{\log(n)}}\right\} \cdot \log n$	sync	$\log(k)$
[8]	✗	$O((\frac{n}{\log(n)})^{1/3})$	$\epsilon \cdot n_2/n$	$md(c) \cdot \log(n)$	sync	$\log(k)$
[23]	✗	$O(n^\epsilon)$	$\sqrt{\log n/n}$	$k + \log(n)$	sync	$\log(k)$
[13]	✗	$o(\sqrt{n/\log(n)})$	$\gg \sqrt{\log n/n}$	$\log(n) \cdot \log\log(n)$	sync	$\log(k)$
[2]	✗	2	arbitrary	$\frac{\log^2(n)}{s\alpha} + \log^2(n)$	async	$s = O(n)$ states
[3]	✗	2	$\gg \log(n)/\sqrt{n}$	$\log(n)$	async	1

Shuffle assumes rough bounds on t_{mix} and n. Bounds on α can reduce the space requirements of our protocols. [26] requires a spanning tree and a common set of quasi-random hash functions. Time in the async model use parallel time. All results, except for [21], hold w.p. $1 - o(1)$. [2] also gives an expected time of $o(\log(n)/(s\alpha) + \log(n) \cdot \log(s))$.

or spanning tree). To capture this diverse spectrum, we classify[3] results into *population protocols*, *sensor networks*, and *pull voting*. A condensed form of this discussion is given in Table 1. We will not discuss work whose focus is too far away from this paper's, e.g., consensus on some arbitrary opinion, leader election, robustness concerns, or Byzantine models.

Population Protocols. The first line of work considers *population protocols* for consensus and models interactions between large populations of very simple entities (like molecules). Entities are modeled as finite state machines with a small state space and communicate asynchronously. In each step, an edge is chosen uniformly at random and only the two connected nodes communicate. We refer to this communication model as the *sequential model*. See [5, 4] for detailed introductions. Dana Angluin, James Aspnes, and David Eisenstat [3] propose a 3-state population protocol for majority voting (i.e., $k = 2$) on the clique. If the initial bias α is $\omega(\log n/\sqrt{n})$, their protocol agrees (w.h.p.) on the majority opinion in $O(n \cdot \log n)$ steps. George B. Mertzios, Sotiris E. Nikoletseas, Christoforos Raptopoulos, and Paul G. Spirakis [27] suggest a 4-state protocol for *exact* majority voting, which always returns the majority opinion (independent of α) in time $O(n^6)$ in arbitrary graphs and in time $O(n^2 \cdot \log(n)/\alpha)$ in the clique. This is optimal in that no population protocol for exact majority can have fewer than four states [27]. Dan Alistarh, Rati Gelashvili, and Milan Vojnovic [2] gives a protocol for $k = 2$ in the clique that allows for a speed-memory trade-off. It solves exact majority and has expected *parallel running time*[4] $O\left(\frac{\log n}{s \cdot \alpha} + \log n \cdot \log s\right)$ and (w.h.p.) $O\left(\frac{\log^2 n}{s \cdot \alpha} + \log^2 n\right)$.

Here, s is the number of states and must be in the range $s = O(n)$ and $s = \Omega(\log n \cdot \log\log n)$.

[3] This classification is neither unique nor injective but merely an attempt to make the overview more accessible.

[4] The number of steps divided by n. A typical measure for population protocols, based on the intuition that each node communicates roughly once in n steps.

In contrast to these population results, our protocols consider the case of arbitrary $k \geq 2$. Also, with the notable exception of [27], the above results are restricted to the complete graph. These restrictions are not surprising, given that these protocols operate on a very constrained state space. Our protocols work in arbitrary, even dynamic graphs. Balance can be seen as a slightly simplified and generalized version of [2], and Shuffle uses a similar idea for a speed-memory trade-off.

Sensor Networks. Another line of work has a background in sensor networks. *Quantized interval consensus* draws its motivation from signal processing. Initially, nodes measure quantized values (signals) and then communicate through a network to agree on the quantized values that enclose the average. This can be used to solve majority consensus ($k = 2$). The communication model is typically sequential. Florence Bénézit, Patrick Thiran, and Martin Vetterli [10] propose a protocol that is equivalent to the 4-state population protocol of [27] and prove that with probability 1 it converges in finite time, but without bounds on that convergence time. A more recent result by Moez Draief and Milan Vojnovic [21] shows that this protocol (and thus [27]) needs $\mathrm{O}\big(\frac{\log n}{\delta(Q_S,\alpha)}\big)$ steps in expectation. Here, $\delta(Q_S, \alpha)$ depends on the bias α and on the spectrum of a set of matrixes Q_S related to the underlying graph. The authors give concrete bounds for several specific graphs (e.g., in the complete graph the consensus time is of order[5] $\mathrm{O}(\log n/\alpha)$). The only related result for $k > 2$ we are aware of is [11] which again proves only convergence in finite time.

Another consensus variant is *mode computation*. For example, Fabian Kuhn, Thomas Locher, and Stefan Schmid [26] consider a graph of diameter D where each node has one or several of k distinct elements. The authors use a protocol based on a complex hashing scheme to compute the *mode* (the most frequent element) w.h.p. in time $\mathrm{O}\big(D + F_2/n_1^2 \cdot \log k\big)$. Here, $F_2 = \sum_i n_i^2$ is the second frequency moment and n_i the frequency of the i-th most common element. $F_2/n_1^2 \in [1, k]$ can be seen as an alternative bias measure. Nodes communicate via synchronous broadcasts and need a precomputed spanning tree and hash functions. [26] can also be used for aggregate computation as done by David Kempe, Alin Dobra, and Johannes Gehrke [25] (where the authors provide an elegant protocol to compute sums or averages in complete graphs).

Overall, [21] and [26] are probably most closely related to our work, as they consider arbitrary graphs. However, we cover more general communication models, including dynamic graphs. Similar to [21], our results for $k = 2$ rely on spectral properties of the underlying graph (and are asymptotically the same for their concrete examples). However, our bounds are related to well-studied load balancing bounds and mixing times of random walks (which we believe are easier to get a handle on than their $\delta(Q_S, \alpha)$).

Gossip Protocols. The third major research line on plurality consensus has its roots in gossiping and rumor spreading. Here, communication is typically restricted to synchronous pull requests (nodes query other nodes' opinions and use simple rules to update their own). See [30] for a slightly dated but thorough survey. Colin Cooper, Robert Elsässer, and Tomasz Radzik [18] consider a voting process for $k = 2$ opinions on d-regular graphs. They pull two random neighbors and, if they have the same opinion, adopt it. For random d-regular graphs and $\alpha = \Omega\big(\sqrt{1/d + d/n}\big)$, all nodes agree (w.h.p.) in $\mathrm{O}(\log n)$ rounds on the plurality opinion. For an arbitrary d-regular graph G, they need $\alpha = \Omega(\lambda_2)$ (where $1 - \lambda_2$ is the spectral gap of G). In the follow up paper Colin Cooper, Robert Elsässer, Tomasz Radzik,

[5] We state their bound in terms of our $\alpha = (n_1 - n_2)/n$; their definition of α differs slightly.

Nicolas Rivera, and Takeharu Shiraga [19] extend these results to expanders: The run time is $\mathrm{O}(\log n)$ for a bias of $\mathrm{vol}(1) - \mathrm{vol}(2) \geq 4\lambda_2^2 \cdot |E|$, where vol(1) and vol(2) denote the sum of degrees over nodes having Opinion 1 and 2, respectively. Luca Becchetti, Andrea E. F. Clementi, Emanuele Natale, Francesco Pasquale, Riccardo Silvestri, and Luca Trevisan [9] consider a similar update rule on the clique for k opinions. Here, each node pulls the opinion of three random neighbors and adopts the majority among those. They need $\mathrm{O}(\log k)$ memory bits and prove (w.h.p.) a tight running time of $\Theta(k \cdot \log n)$ (given a sufficiently high bias α). Luca Becchetti, Andrea Clementi, Emanuele Natale, Francesco Pasquale, and Riccardo Silvestri [8] build upon the idea of the 3-state population protocol from [3] (but in the gossip model) and generalize it to k opinions. Nodes pull the opinion of a random neighbor in each round. If $n_1 \geq (1+\varepsilon) \cdot n_2$ for a constant $\varepsilon > 0$ and if $k = \mathrm{O}\left((n/\log n)^{1/3}\right)$, they agree (w.h.p.) on the plurality opinion in time $\mathrm{O}(\mathrm{md} \cdot \log n)$ on the clique and need $\log k + 1$ bits. The *monochromatic distance* $\mathrm{md} \in [1, k]$ is an alternative bias measure (based on an idea similar to the frequency moment in [26]). Petra Berenbrink, Tom Friedetzky, George Giakkoupis, and Peter Kling [13] build upon [3] and design a protocol that reaches plurality consensus (w.h.p.) in time $\mathrm{O}(\log n \cdot \log\log n)$ and uses $\log k + 4$ bits.

The running times of gossip protocols are relatively good when compared to other protocols, like population protocols or those introduced here (cf. Table 1). In particular, these results do typically not show a linear dependency on the bias, as our Shuffle protocol or [2, 21, 27] do. This efficiency however comes at the expense of parameter constraints. In particular, results like [8, 13] do not seem to easily extend to arbitrary graphs and have inherent constraints on both k and α. Comparing these results seems to indicate that, at least for arbitrary graphs, there is a jump in complexity depending on whether or not one allows the protocol to fail for small absolute bias values.

1.2 Our Contribution

We introduce two protocols for plurality consensus, called Shuffle and Balance. Both solve plurality consensus under a diverse set of (randomized or adversarial) communication patterns in arbitrary graphs for any positive bias. We continue with a detailed description of our results.

Shuffle. Our main result is the Shuffle protocol. In the first time step each node generates γ tokens labeled with its initial opinion. During round t, any pair of nodes connected by an active edge (as specified by the communication pattern $(\boldsymbol{M}_t)_{t \leq N}$) exchanges tokens. We show that Shuffle solves plurality consensus and allows for a trade-off between running time and memory. More exactly, let the number of tokens be $\gamma = \mathrm{O}\left(\log n/(\alpha^2 \cdot T)\right)$, where T is a parameter to control the trade-off between memory and running time[6]. Moreover, let t_{mix} be such that any time interval $[t, t + t_{\mathrm{mix}}]$ is *ε-smoothing*[7] (cf. Section 2). Given knowledge of the *maximum number of communication partners* Δ and the *mixing time* t_{mix} of the underlying communication pattern[8], Shuffle lets all nodes agree on the plurality opinion in $\mathrm{O}(T \cdot t_{\mathrm{mix}})$ rounds (w.h.p.), using $\mathrm{O}\left(\log n/(\alpha^2 T) \cdot \log k + \log(T \cdot t_{\mathrm{mix}})\right)$ memory bits per

[6] The protocol works for any integral choice of γ (this fixes the trade-off parameter T).

[7] Intuitively, this means that the communication pattern has good load balancing properties during any time window of length t_{mix}. This coincides with the worst-case mixing time of a lazy random walk on active edges.

[8] For static graphs, Δ is the maximal degree which can be easily computed in a distributed way, see for example [14]. For t_{mix}, good bounds are known for many static graphs [1, Chapter 5].

node. This implies, for example, that plurality consensus on expanders in the sequential model is achieved in $\mathrm{O}\left(T \cdot n \log n\right)$ time steps and with $\mathrm{O}\left(\log n \cdot \log k/T + \log(Tn)\right)$ memory bits (assuming a constant initial bias). For arbitrary graphs, arbitrary bias, and many natural communication patterns (e.g., communicating with all neighbors in every round or communicating via random matchings), the time for plurality consensus is closely related to the spectral gap of the underlying communication network (cf. Corollary 3).

While our protocol is relatively simple, the analysis is quite involved. The idea is to observe that after t_{mix} time steps, each single token is on any node with (roughly) the same probability; the difficulty is that token movements are not independent. The main ingredients for our analysis are Lemmas 6 and 7, which generalize a result by Thomas Sauerwald and He Sun [32] (we believe that this generalization is interesting in its own right). These lemmas show that the joint distribution of token locations is negatively correlated, allowing us to derive a suitable Chernoff bound. Once this is proven, nodes can "count" tokens every t_{mix} time steps, building up over time an estimate of the total number of tokens labeled with their own opinion. By broadcasting these estimates, all nodes determine the plurality opinion.

Balance. The previous protocol, Shuffle, allows for a nice trade-off between running time and memory. If the number of opinions is relartively small, our much simpler Balance protocol gives better results. In Balance, each node u maintains a k-dimensional load vector. If j denotes u's initial opinion, the j-th dimension of this load vector is initialized with $\gamma \in \mathbb{N}$ (a sufficiently large value) and any other dimension is initialized with zero. In each time step, all nodes perform a simple, discrete load balancing on each dimension of these load vectors. Our results imply, for example, that plurality consensus on expanders in the sequential model is achieved in only $\mathrm{O}\left(n \cdot \log n\right)$ time steps with $\mathrm{O}\left(k\right)$ memory bits per node (assuming a constant initial bias).

Balance can be thought of as a (slightly simplified) version of [2] or [25] that generalizes naturally to $k \geq 2$ and arbitrary (even dynamic) graphs. In the setting of [2] (but as opposed to [2] for arbitrary k), it achieves plurality consensus with probability $1 - \mathrm{o}\left(1\right)$ in parallel time $\mathrm{O}\left(\log n\right)$ and uses $\mathrm{O}\left(k \cdot \log(1/\alpha)\right) = \mathrm{O}\left(k \cdot \log n\right)$ bits per node (Corollary 13), an improvement by a $\log(n)$ factor.

2 Model & General Definitions

We consider an undirected graph $G = (V, E)$ of $n \in \mathbb{N}$ nodes and let $1 - \lambda_2$ denote the eigenvalue (or spectral) gap of G. Each node u is assigned an *opinion* $o_u \in \{1, 2, \dots, k\}$. For $i \in \{1, 2, \dots, k\}$, we use $n_i \in \mathbb{N}$ to denote the number of nodes which have initially opinion i. Without loss of generality (w.l.o.g), we assume $n_1 > n_2 \geq \dots \geq n_k$, such that 1 is the opinion that is initially supported by the largest subset of nodes. We also say that 1 is the *plurality opinion*. The value $\alpha := \frac{n_1 - n_2}{n} \in [1/n, 1]$ denotes the *initial bias* towards the plurality opinion. In the *plurality consensus problem*, the goal is to design simple, distributed protocols that let all nodes agree on the plurality opinion. Time is measured in discrete rounds, such that the (randomized) running time of our protocols is the number of rounds it takes until all nodes are aware of the plurality opinion. As a second quality measure, we consider the total number of memory bits per node that are required by our protocols. All our statements and proofs assume n to be sufficiently large.

Communication Model. In any given round, two nodes u and v can communicate if and only if the edge between u and v is *active*. We use $\boldsymbol{M}_t$ to denote the symmetric *communication matrix* at time t, where $\boldsymbol{M}_t[u,v] = \boldsymbol{M}_t[v,u] = 1$ if $\{u,v\}$ is active and $\boldsymbol{M}_t[u,v] = \boldsymbol{M}_t[v,u] = 0$ otherwise. We assume (w.l.o.g) $\boldsymbol{M}_t[u,u] = 1$ (allowing nodes to "communicate" with themselves). Typically, the sequence $\boldsymbol{M} = (\boldsymbol{M}_t)_{t\in\mathbb{N}}$ of communication matrices (the *communication pattern*) is either randomized or adversarial, and our statements merely require that $\boldsymbol{M}$ satisfies certain smoothing properties (see below). For the ease of presentation, we restrict ourselves to polynomial number of time steps and consider only communication patterns $\boldsymbol{M} = (\boldsymbol{M}_t)_{t\leq N}$ where $N = N(n)$ is an arbitrarily large polynomial. Let us briefly mention some natural and common communication models covered by such patterns:

- *Diffusion Model:* All edges of the graph are permanently activated.
- *Random matching model:* In every round t, the active edges are given by a random matching. We require that random matchings from different rounds are mutually independent[9]. Results for the random matching model dependent on $p_{\min} := \min_{t\in\mathbb{N},\{u,v\}\in E} \Pr(\boldsymbol{M}_t[u,v] = 1)$.
- *Balancing Circuit Model:* There are d perfect matchings $\boldsymbol{M}_0, \boldsymbol{M}_1, \ldots, \boldsymbol{M}_{d-1}$ given. They are used in a round-robin fashion, such that for $t \geq d$ we have $\boldsymbol{M}_t = \boldsymbol{M}_{t \bmod d}$.
- *Sequential Model:* In each round t an uniformly random edge $\{u,v\} \in E$ is activated.

Notation. We use $\|\boldsymbol{x}\|_\ell$ to denote the ℓ-norm of vector $\boldsymbol{x}$, where the ∞-norm is the vector's maximum absolute entry. In general, bold font indicates vectors and matrices, and $x(i)$ refers to the i-th component of $\boldsymbol{x}$. The *discrepancy* of $\boldsymbol{x}$ is defined as $\operatorname{disc}(\boldsymbol{x}) := \max_i x(i) - \min_i x(i)$. For $i \in \mathbb{N}$ we define $[i] := \{1, 2, \ldots, i\}$ as the set of the first i integers. We use $\log x$ to denote the binary logarithm of $x \in \mathbb{R}_{>0}$. We write $a \mid b$ if a divides b. For any node $u \in V$, we use $d(u)$ to denote u's degree in G and $d_t(u) := \sum_v \boldsymbol{M}_t[u,v]$ to denote its *active degree* at time t (i.e., its degree when restricted to active edges). Similarly, $N(u)$ and $N_t(u)$ refer to u's (active) neighborhood. Moreover, $\Delta := \max_{t,u} d_t(u)$ is the maximum active degree of any node. We assume knowledge of Δ. On static graphs it can be computed efficiently in a distributed manner [14] and it is given by many dynamic graph models (e.g., 1 for the sequential model, d for balancing circuits). We say an event happens with high probability (w.h.p.) if its probability is at least $1 - 1/n^c$ for $c \in \mathbb{N}$.

Smoothing Property. The running time of our protocols is closely related to the runnig time ("smoothing time") of diffusion load balancing algorithms, which in turn is a function of the mixing time of a random walk on G (see also [6, 32]). More exactly, we consider a random walk on G that is restricted to the active edges in each time step. As indicated in Section 1.2, this random walk should converge towards the uniform distribution over the nodes of G. This leads to the following definition of the random walk's transition matrices $\boldsymbol{P}_t$ based on the communication matrices $\boldsymbol{M}_t$:

$$\boldsymbol{P}_t[u,v] := \begin{cases} \frac{1}{2\Delta} & \text{if } \boldsymbol{M}_t[u,v] = 1 \text{ and } u \neq v, \\ 1 - \frac{d_t(u)}{2\Delta} & \text{if } \boldsymbol{M}_t[u,v] = 1 \text{ and } u = v, \\ 0 & \text{if } \boldsymbol{M}_t[u,v] = 0. \end{cases} \tag{1}$$

Obviously, $\boldsymbol{P}_t$ is doubly stochastic for all $t \in \mathbb{N}$. Moreover, note that the random walk is trivial in any matching-based model, while we get $\boldsymbol{P}_t[u,v] = \frac{1}{2d}$ for every edge $\{u,v\} \in E$ in

[9] Note that there are several simple, distributed protocols to obtain such matchings [24, 14].

the diffusion model on a d-regular graph. We are now ready to define the required smoothing property.

▶ **Definition 1** (ε-smoothing)**.** Consider a fixed sequence $(\boldsymbol{M}_t)_{t\leq N}$ of communication matrices and a time interval $[t_1, t_2]$. We say $[t_1, t_2]$ is *ε-smoothing* (under $(\boldsymbol{M}_t)_{t\leq N}$) if for any non-negative vector $\boldsymbol{x}$ with $\|\boldsymbol{x}\|_\infty = 1$ it holds that $\operatorname{disc}(\boldsymbol{x} \cdot \prod_{t=t_1}^{t_2} \boldsymbol{P}_t) \leq \varepsilon$. Moreover, we define the *mixing time* $t_{\text{mix}}(\varepsilon)$ as the smallest number of steps such that any time window of length $t_{\text{mix}}(\varepsilon)$ is ε-smoothing. That is, $t_{\text{mix}}(\varepsilon) \coloneqq \min \{ t' \mid \forall t \in \mathbb{N}\colon [t, t+t'] \text{ is } \varepsilon\text{-smoothing} \}$.

The mixing time can be seen as the worst-case time required by a random walk to get "close" to the uniform distribution. If the parameter ε is not explicitly stated, we consider $t_{\text{mix}} \coloneqq t_{\text{mix}}(n^{-5})$. Note that Shuffle assumes knowledge of a bound on t_{mix} (cf. Section 1.2).

3 Protocol Shuffle

Our main result is the following theorem, stating the correctness as well as the time-/space-efficiency of Shuffle. The protocol is described in Section 3.1, followed by its analysis in Section 3.2.

▶ **Theorem 2.** *Let $\alpha = \frac{n_1 - n_2}{n} \in [1/n, 1]$ denotes the initial bias. Consider a fixed communication pattern $(\boldsymbol{M}_t)_{t\leq N}$ and an arbitrary parameter $T \in \mathbb{N}$. Protocol* Shuffle *ensures that all nodes know the plurality opinion after $\mathrm{O}(T \cdot t_{mix})$ rounds (w.h.p.) and requires $\left(12 \cdot \frac{\log(n)}{\alpha^2 \cdot T} + 4\right) \cdot \log(k) + 4\log\left(\frac{12 \cdot \log(n)}{\alpha^2}\right) + \log(T \cdot t_{mix})$ memory bits per node.*

The parameter T in the statement serves as a lever to trade running time for memory. Since t_{mix} depends on the graph and communication pattern, Theorem 2 might look a bit unwieldy. The following corollary gives a few concrete examples for common communication patterns on general graphs.

▶ **Corollary 3.** *Let G be an arbitrary d-regular graph.* Shuffle *ensures that all nodes agree on the plurality opinion (w.h.p.) using $\left(12 \cdot \frac{\log(n)}{\alpha^2 \cdot T} + 4\right) \cdot \log(k) + 4\log\left(\frac{12 \cdot \log(n)}{\alpha^2}\right) + \log(T \cdot t_{mix})$ bits of memory in time*

- $\mathrm{O}\left(T \cdot \frac{\log(n)}{1-\lambda_2}\right)$ *in the diffusion model,*
- $\mathrm{O}\left(\frac{T}{d \cdot p_{min}} \cdot \frac{\log(n)}{1-\lambda_2}\right)$ *in the random matching model,*
- $\mathrm{O}\left(T \cdot d \cdot \frac{\log(n)}{1-\lambda_2}\right)$ *in the balancing circuit model, and*
- $\mathrm{O}\left(T \cdot n \cdot \frac{\log(n)}{1-\lambda_2}\right)$ *in the sequential model.*

3.1 Protocol Description

We continue to explain the Shuffle protocol given in Listing 1. Our protocol consists of three parts that are executed in each time step: the *shuffle* part, the *broadcast* part, and the *update* part.

Every node u is initialized with $\gamma \in \mathbb{N}$ tokens labeled with u's opinion o_u. Our protocol sends 2Δ tokens chosen uniformly at random (without replacement) over each edge $\{ u, v \} \in E$. Here, $\gamma \geq 2\Delta^2$ is a parameter depending on T and α to be fixed during the analysis[10]. Shuffle maintains the invariant that, at any time, all nodes have exactly γ tokens. In addition to storing the tokens, each node maintains a set of auxiliary variables. The variable

[10] Shuffle needs not to know α, it works for any choice of γ; such a choice merely fixes the trade-off parameter T.

```
for $\{u, v\} \in E$ with $M_t[u, v] = 1$:                              {shuffle part}
  send $2\Delta$ tokens chosen u.a.r. (without replacement) to $v$

for $\{u, v\} \in E$ with $M_t[u, v] = 1$:                              {broadcast part}
  send    $(dom_u, e_u)$
  receive $(dom_v, e_v)$
$v := w$ with $e_w \geq e_{w'}$  $\forall w, w' \in N_t(u) \cup \{u\}$
$(dom_u, e_u) := (dom_v, e_v)$

if $t \equiv 0 \pmod{t_{\text{mix}}}$:                                  {update part}
  increase $c_u$ by the number of tokens labeled $o_u$ held by $u$
  $plu_u := dom_u$            {plurality guess: last broadcast's dom. op.}
  $(dom_u, e_u) := (o_u, c_u)$   {reset broadcast}
```

Listing 1 Protocol SHUFFLE as executed by node u at time t. At time zero, each node u creates γ tokens labeled o_u and sets $c_u := 0$ and $(dom_u, e_u) := (o_u, c_u)$.

c_u is increased during the update part and counts tokens labeled o_u. The variable pair (dom_u, e_u) is a temporary guess of the plurality opinion and its frequency. During the broadcast part, nodes broadcast these pairs, replacing their own pair whenever they observe a pair with higher frequency. Finally, the variable plu_u represents the opinion currently believed to be the plurality opinion. The shuffle and broadcast parts are executed in each time step, while the update part is executed only every t_{mix} time steps

Waiting t_{mix} time steps for each update gives the broadcast enough time to inform all nodes and ensures that the tokens of each opinion are well distributed. The latter implies that, if we consider a node u with opinion $o_u = i$ at time $T \cdot t_{\text{mix}}$, the value c_u is a good estimate of $T \cdot \gamma n_i / n$ (which is maximized for the plurality opinion). When we reset the broadcast (Line 13), the subsequent t_{mix} broadcast steps ensure that all nodes get to know the pair (o_u, c_u) for which c_u is maximal. Thus, if we can ensure that c_u is a good enough approximation of $T \cdot \gamma n_i / n$, all nodes get to know the plurality.

3.2 Analysis of Shuffle

Fix a communication pattern $(M_t)_{t \leq N}$ and an arbitrary parameter $T \in \mathbb{N}$. Remember that $t_{\text{mix}} = t_{\text{mix}}(n^{-5})$ denotes the smallest number such that any time window of length t_{mix} is n^{-5}-smoothing under $(M_t)_{t \leq N}$. We set the number of tokens stored in each node to $\gamma := \lceil c \cdot \frac{\log n}{\alpha^2 T} \rceil$, where c is a suitable constant. The analysis of SHUFFLE is largely based on Lemma 11, which states that, after $O(T \cdot t_{\text{mix}})$ time steps, the counter values c_u can be used to reliably separate the plurality opinion from any other opinion. The main technical difficulty is the huge dependency between the tokens' movements, rendering standard Chernoff-bounds inapplicable. Instead, we show that certain random variables satisfy the negative regression condition (Lemma 6), which allows us to majorize the token distribution by a random walk (Lemma 7) and to derive the Chernoff type bound in Lemma 10. This Chernoff type bound can be used to show that all counter values are concentrated which is the main pillar of the proof of Theorem 2.

Majorizing Shuffle by Random Walks

While our SHUFFLE protocol assumes that 2Δ divides γ, here we assume the slightly weaker requirement that $P_t[u, v] \cdot \gamma \in \mathbb{N}$ for any $u, v \in V$ and $t \in \mathbb{N}$. Let us first introduce some notation for the shuffle part at time t of our protocol. To ease the discussion, we consider u as a neighbor of itself and speak of $d_t(u)+1$ neighbors. For $i \in [d_t(u)+1]$, let $N_t(u,i) \in V$ denote the i-th neighbor of u (in an arbitrary order). Fix a node u and let u's tokens be numbered from 1 to γ. Our assumption on γ allows us to partition the tokens into $d_t(u)+1$ disjoint subsets (*slots*) $S_i \subseteq [\gamma]$ of size $\boldsymbol{P}_t[u,v] \cdot \gamma$ each, where $v = N_t(u,i)$. Let $\pi_{t,u} \colon [\gamma] \to [\gamma]$ be a random permutation. Token j with $\pi_{t,u}(j) \in S_i$ is sent to u's i-th neighbor. To ease notation, we drop the time index t and write π_u instead of $\pi_{t,u}$ (and, similarly for $d(u)$ and $N(u,i)$).

A *configuration* c describes the location of *all* γn tokens at a given point in time. For a token $j \in [\gamma n]$ we use $u_j \in V$ to denote its location in configuration c (which will always be clear from the context). For each such token j we define a random variable $X_j \in [d(u_j)+1]$ with $X_j = i$ if and only if $\pi_{u_j}(j) \in S_i$. In other words, X_j indicates to which of u_j's neighbors token j is sent. Our key technical lemma (Lemma 6) establishes the *negative regression condition* for these $(X_j)_{j\in[\gamma n]}$ variables. Negative regression is defined as follows:

▶ **Definition 4** (Neg. Regression [22, Def. 21]). A vector $(X_1, X_2, \ldots, X_n)$ of random variables is said to satisfy the *negative regression* condition if $\mathbb{E}\left[f(X_l, l \in \mathcal{L}) \mid X_r = x_r, r \in \mathcal{R}\right]$ is non-increasing in each x_r for any disjoint $\mathcal{L}, \mathcal{R} \subseteq [n]$ and for any non-decreasing function f.

▶ **Lemma 5** ([22, Lemma 26]). *Let $(X_1, X_2, \ldots, X_n)$ satisfy the negative regression condition and consider an arbitrary index set $I \subseteq [n]$ as well as any family of non-decreasing functions f_i $(i \in \{ I \})$. Then, we have*

$$\mathbb{E}\left[\prod_{i\in I} f_i(X_i)\right] \le \prod_{i\in I} \mathbb{E}\left[f_i(X_i)\right] \tag{2}$$

▶ **Lemma 6** (NRC). *Fix a configuration c and consider the random variables $(X_j)_{j\in[\gamma n]}$. Then $(X_j)_{j\in[\gamma n]}$ satisfies the negative regression condition (*NRC*).*

Proof. Remember that u_j is the location of token j in configuration c and that $X_j \in [d(u_j)+1]$ indicates where token j is sent in the next step. We show for any $u \in V$ that $(X_j)_{j\colon u_j=u}$ satisfies the NRC. The lemma's statement follows since the π_u are chosen independently (if two independent vectors (X_j) and (Y_j) satisfy the NRC, then so do both together).

Fix a node u and disjoint subsets $\mathcal{L}, \mathcal{R} \subseteq \{ j \in [\gamma n] \mid u_j = u \}$ of tokens on u. Define $d \coloneqq d(u)$ and let $f \colon [d+1]^{|\mathcal{L}|} \to \mathbb{R}$ be an arbitrary non-decreasing function. We have to show that $\mathbb{E}\left[f(X_l, l \in \mathcal{L}) \mid X_r = x_r, r \in \mathcal{R}\right]$ is non-increasing in each x_r (cf. Definition 4). That is, we need

$$\mathbb{E}\left[f(X_l, l \in \mathcal{L}) \mid X_r = x_r, r \in \mathcal{R}\right] \le \mathbb{E}\left[f(X_l, l \in \mathcal{L}) \mid X_r = \tilde{x}_r, r \in \mathcal{R}\right], \tag{3}$$

where $x_r = \tilde{x}_r$ holds for all $r \in \mathcal{R} \setminus \{ \hat{r} \}$ and $x_{\hat{r}} > \tilde{x}_{\hat{r}}$ for a fixed index $\hat{r} \in \mathcal{R}$. We prove Inequality (3) via a coupling of the processes on the left-hand side (LHS process) and right-hand side (RHS process) of that inequality. Since $x_{\hat{r}} \neq \tilde{x}_{\hat{r}}$, these processes involve two slightly different probability spaces Ω and $\tilde{\Omega}$, respectively. To couple these, we employ a common uniform random variable $U_i \in [0,1)$. By partitioning $[0,1)$ into $d+1$ suitable slots for each process (corresponding to the slots S_i mentioned above), we can use the outcome of U_i to set the X_j in both Ω and $\tilde{\Omega}$. We first explain how to handle the case $x_{\hat{r}} - \tilde{x}_{\hat{r}} = 1$. The case $x_{\hat{r}} - \tilde{x}_{\hat{r}} > 1$ follows from this by a simple reordering argument.

So assume $x_{\hat{r}} - \tilde{x}_{\hat{r}} = 1$. We reveal the yet unset random variables X_j (i.e., $j \in [\gamma n] \setminus \mathcal{R}$) one by one in order of increasing indices. To ease the description assume (w.l.o.g.) that the tokens from $\mathcal{R}$ are numbered from 1 to $|\mathcal{R}|$. When we reveal the j-th variable (which indicates

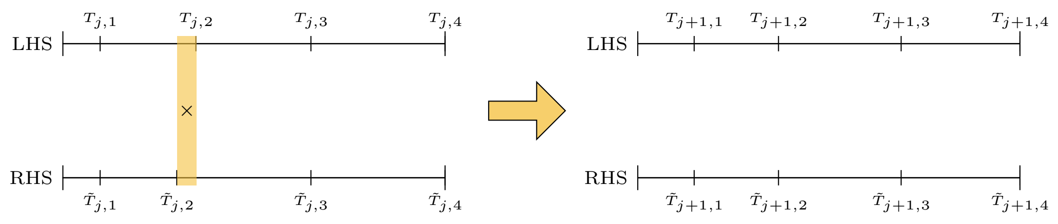


Figure 1 Illustration showing the $d+1=4$ different slots for the LHS and RHS process and how they change. In this example, $x_{\hat{r}}=3$ and $\tilde{x}_{\hat{r}}=2$. On the left, the uniform random variable U_j falls into slot $[T_1,T_2)$ for the LHS process (causing j to be sent to node $N(u,2)$) and into slot $[\tilde{T}_2,\tilde{T}_3)$ for the RHS process (causing j to be sent to node $N(u,3)$).

the new location of the j-th token), note that the probability $p_{j,i}$ that token j is assigned to $N(u,i)$ depends solely on the *number* of previous tokens $j'<j$ that were assigned to $N(u,i)$. Thus, we can consider $p_{j,i}\colon \mathbb{N}\to[0,1]$ as a function mapping $x\in\mathbb{N}$ to the probability that j is assigned to $N(u,i)$ conditioned on the event that exactly x previous tokens were assigned to $N(u,i)$. Note that $p_{j,i}$ is non-increasing. For a vector $\boldsymbol{x}\in\mathbb{N}^{d+1}$, we define a threshold function $T_{j,i}\colon\mathbb{N}^{d+1}\to[0,1]$ by $T_{j,i}(\boldsymbol{x}) \coloneqq \sum_{i'\leq i} p_{j,i'}(x_{i'})$ for each $i\in[d+1]$. To define our coupling, let $\beta_{j,i}\coloneqq |\{\, j'<j \mid X_{j'}=i\,\}|$ denote the number of already revealed variables with value i in the LHS process and define, similarly, $\tilde{\beta}_{j,i}\coloneqq|\{\, j'<j \mid \tilde{X}_{j'}=i\,\}|$ for the RHS process. We use $\boldsymbol{\beta_j},\boldsymbol{\tilde{\beta}_j}\in\mathbb{N}^{d+1}$ to denote the corresponding vectors. Now, to assign token j we consider a uniform random variable $U_j\in[0,1)$ and assign j in both processes using customized partitions of the unit interval. To this end, let $T_{j,i}\coloneqq T_{j,i}(\boldsymbol{\beta_j})$ and $\tilde{T}_{j,i}\coloneqq T_{j,i}(\boldsymbol{\tilde{\beta}_j})$ for each $i\in[d+1]$. We assign X_j in the LHS and RHS process as follows:

- **LHS Process:** $X_j=x_j=i$ if and only if $U_j\in[T_{j,i-1},T_{j,i})$,
- **RHS Process:** $X_j=\tilde{x}_j=i$ if and only if $U_j\in[\tilde{T}_{j,i-1},\tilde{T}_{j,i})$.

See Figure 1 for an illustration. Our construction guarantees that, considered in isolation, both the LHS and RHS process behave correctly.

At the beginning of this coupling, only the variables X_r corresponding to tokens $r\in\mathcal{R}$ are set, and these differ in the LHS and RHS process only for the index $\hat{r}\in\mathcal{R}$, for which we have $X_{\hat{r}}=x_{\hat{r}}$ (LHS) and $X_{\hat{r}}=\tilde{x}_{\hat{r}}=x_{\hat{r}}-1$ (RHS). Let $\iota\coloneqq x_{\hat{r}}$. For the first revealed token $j=\hat{r}+1$, this implies $\beta_{j,\iota}=\tilde{\beta}_{j,\iota}+1$, $\beta_{j,\iota-1}=\tilde{\beta}_{j,\iota-1}-1$, and $\beta_{j,i}=\tilde{\beta}_{j,i}$ for all $i\not\in\{\,\iota,\iota-1\,\}$. By the definitions of the slots for both processes, we get $T_{j,i}=\tilde{T}_{j,i}$ for all $i\neq\iota-1$ and $T_{j,\iota-1}>\tilde{T}_{j,\iota-1}$ (cf. Figure 1). Thus, the LHS and RHS process behave different if and only if $U_i\in[\tilde{T}_{j,\iota-1},T_{j,\iota-1})$. If this happens, we get $x_j<\tilde{x}_j$ (i.e., token j is assigned to a smaller neighbor in the LHS process). This implies $\boldsymbol{\beta_{j+1}}=\boldsymbol{\tilde{\beta}_{j+1}}$ and both processes behave identical from now on. Otherwise, if $U_i\not\in[\tilde{T}_{j,\iota-1},T_{j,\iota-1})$, we have $\boldsymbol{\beta_{j+1}}-\boldsymbol{\beta_{j+1}}=\boldsymbol{\beta_j}-\boldsymbol{\beta_j}$ and we can repeat the above argument. Thus, after all X_j are revealed, there is at most one $j\in\mathcal{L}$ for which $x_j\neq\tilde{x}_j$, and for this we have $x_j<\tilde{x}_j$. Since f is non-decreasing, this guarantees Inequality (3). To handle the case $x_{\hat{r}}-\tilde{x}_{\hat{r}}>1$, note that we can reorder the slots $[T_{j,i-1},T_{j,i})$ used for the assignment of the variables such that the slots for $x_{\hat{r}}$ and $\tilde{x}_{\hat{r}}$ are neighboring. Formally, this merely changes in which order we consider the neighbors in the definition of the functions $T_{j,i}$. With this change, the same arguments as above apply. ◀

Before proving the majorization of tokens with random walks (Lemma 7) we require further notation. Let $\mathcal{S}$ denote our random Shuffle process, and $\mathcal{W}$ the random walk process in which each of the γn tokens performs an independent random walk according to the sequence of random walk matrices $(\boldsymbol{P}_t)_{t\in\mathbb{N}}$ (i.e., a token on u uses $P_t(u,\cdot)$ for the transition probabilities). We use $w_j^{\mathcal{P}}(t)$ to denote the position of token j after t steps of a process $\mathcal{P}$. We assume (w.l.o.g.) $w_j^{\mathcal{S}}(0) = w_j^{\mathcal{W}}(0)$ for all j. While there are strong correlations between the tokens' movements in $\mathcal{S}$ (e.g., not all tokens can move to the same neighbor), Lemma 7 shows that these correlations are negative.

▶ **Lemma 7** (Majorizing RWs). *Consider a time $t \geq 0$, a token j, and node v. Let $B \subseteq [\gamma n]$ and $D \subseteq V$ be arbitrary subsets of tokens and nodes, respectively. The following holds:*

1. $\Pr\left(w_j^{\mathcal{S}}(t) = v\right) = \Pr\left(w_j^{\mathcal{W}}(t) = v\right)$ *and*

2. $\Pr\left(\bigcap_{j \in B}\left(w_j^{\mathcal{S}}(t) \in D\right)\right) \leq \Pr\left(\bigcap_{j \in B}\left(w_j^{\mathcal{W}}(t) \in D\right)\right) = \prod_{j \in B} \Pr\left(w_j^{\mathcal{W}}(t) \in D\right)$.

Proof. The first statement follows immediately from the definition of our process. For the second statement, note that the equality on the right-hand side holds trivially, since the tokens perform independent random walks in $\mathcal{W}$. To show the inequality, we define the intermediate process $\mathcal{SW}(t')$ ($t' \leq t$) that performs t' steps of $\mathcal{S}$ followed by $t - t'$ steps of $\mathcal{W}$. By this definition, $\mathcal{SW}(0)$ is identical to $\mathcal{W}$ restricted to t steps and, similar, $\mathcal{SW}(t)$ is identical to $\mathcal{S}$ restricted to t steps. Define

$$\mathcal{E}_{t'} \coloneqq \bigcap_{j \in B} \left(w_j^{\mathcal{SW}(t')}(t) \in D\right) \tag{4}$$

(the event that all tokens from B end up at nodes from D under process $\mathcal{SW}(t')$). The lemma's statement is equivalent to $\Pr(\mathcal{E}_t) \leq \Pr(\mathcal{E}_0)$. To prove this, we show $\Pr(\mathcal{E}_{t'+1}) \leq \Pr(\mathcal{E}_{t'})$ for all $t' \in \{0, 1, \ldots, t-1\}$. Combining these inequalities yields the desired result.

Fix an arbitrary $t' \in \{0, 1, \ldots, t-1\}$ and note that $\mathcal{SW}(t')$ and $\mathcal{SW}(t'+1)$ behave identical up to and including step t'. Hence, we can fix an arbitrary configuration (i.e., the location of each token) $c(t') = c$ immediately before time step $t'+1$. Remember that $u_j \in V$ denotes the location of j in configuration c. The auxiliary functions $h_j \colon [d(u_j)+1] \to [0,1]$ describe the probability that a random walk starting at time $t'+1$ from u_j's i-th neighbor ends up in a node from D. Formally,

$$h_j(i) \coloneqq \Pr\left(w_j^{\mathcal{W}}(t) \in D \;\middle|\; w_j^{\mathcal{W}}(t'+1) = N(u_j, i)\right). \tag{5}$$

We can assume (w.l.o.g.) that all h_j are non-decreasing (by reordering the neighborhood of u_j).

Now, by Lemma 6 the variables $(X_j)_{j \in B}$ satisfy the negative regression condition. Thus, we can apply Lemma 5 (a well-known characterization of negative regression) to the functions h_j. Using another simple auxiliary result (Claim 8) we can relate the (conditioned) probabilities of the events $\mathcal{E}_{t'}$ and $\mathcal{E}_{t'+1}$ to the expectations over the different $h_j(X_j)$. That is, for $p \coloneqq \Pr(\mathcal{E}_{t'+1} \mid c(t') = c)$ we compute

$$p \overset{Clm.\ 8(a)}{=} \mathbb{E}\left[\prod_{j \in B} h_j(X_j) \;\middle|\; c(t') = c\right] \overset{Lem.\ 5}{\leq} \prod_{j \in B} \mathbb{E}\left[h_j(X_j) \mid c(t') = c\right]$$
$$\overset{Clm.\ 8(b)}{=} \Pr(\mathcal{E}_{t'} \mid c(t') = c).$$

Using the law of total probability, we conclude $\Pr(\mathcal{E}_{t'+1}) \leq \Pr(\mathcal{E}_{t'})$, as required. ◀

▶ **Claim 8.** *Fix a time $t' \in \{0, 1, \ldots, t-1\}$ and consider an arbitrary configuration c. Then the following identities hold:*

(a) $\Pr(\mathcal{E}_{t'+1} \mid c(t') = c) = \mathbb{E}\left[\prod_{j \in B} h_j(X_j) \;\middle|\; c(t') = c\right]$, *and*

(b) $\Pr(\mathcal{E}_{t'} \mid c(t') = c) = \prod_{j \in B} \mathbb{E}\left[h_j(X_j) \mid c(t') = c\right]$.

Proof. Remember the definitions from Lemma 7 and its proof. We use the shorthand $d(u_j) = d_{t'+1}(u_j)$. Remember that each X_j indicates to which of the $d(u_j)+1$ neighbors of u_j (where u_j is considered a neighbor of itself) a token j moves during time step $t'+1$. Thus, given the configuration $c(t') = c$ immediately before time step $t'+1$, there is a bijection between any possible configuration $c(t'+1)$ and outcomes of the random variable vector $\boldsymbol{X} = (X_j)_{j\in[\gamma n]}$. Let $c_{\boldsymbol{x}}$ denote the configuration corresponding to a concrete outcome $\boldsymbol{X} = \boldsymbol{x} \in [d(u_j)+1]^{\gamma n}$. Thus, we have $\Pr(c(t'+1) = c_{\boldsymbol{x}} \mid c(t') = c) = \Pr(\boldsymbol{X} = \boldsymbol{x} \mid c(t') = c)$, and conditioning on $c(t'+1)$ is equivalent to conditioning on $\boldsymbol{X}$ and $c(t')$. For the claim's first statement, we calculate

$$\begin{aligned}\Pr\left(\mathcal{E}_{t'+1} \;\middle|\; c(t') = c\right) &= \sum_{c_{\boldsymbol{x}}} \Pr\left(\mathcal{E}_{t'+1} \;\middle|\; c(t'+1) = c_{\boldsymbol{x}}\right) \cdot \Pr\left(c(t'+1) = c_{\boldsymbol{x}} \;\middle|\; c(t') = c\right)\\ &= \sum_{c_{\boldsymbol{x}}} \prod_{j\in B} \Pr\left(w_j^{\mathcal{SW}(t'+1)}(t) \in D \;\middle|\; \boldsymbol{X} = \boldsymbol{x}, c(t') = c\right) \cdot \Pr\left(\boldsymbol{X} = \boldsymbol{x} \;\middle|\; c(t') = c\right)\\ &= \sum_{c_{\boldsymbol{x}}} \prod_{j\in B} h_j(x_j) \cdot \Pr\left(\boldsymbol{X} = \boldsymbol{x} \;\middle|\; c(t') = c\right)\\ &= \sum_{\boldsymbol{x}} \prod_{j\in B} h_j(x_j) \cdot \Pr\left(\boldsymbol{X} = \boldsymbol{x} \;\middle|\; c(t') = c\right) = \mathbb{E}\left[\prod_{j\in B} h_j(X_j) \;\middle|\; c(t') = c\right].\end{aligned}$$

Here, we first apply the law of total probability. Then, we use the bijection between $c(t'+1)$ and $\boldsymbol{X}$ (if $c(t')$ is given) and that the process $\mathcal{SW}(t'+1)$ consists of independent random walks if $c(t'+1)$ is fixed. Finally, we use the definition of the auxiliary functions $h_j(i)$, which equal the probability that a random walk starting at time $t'+1$ from u_j's i-th neighbor reaches a node from D.

For the claim's second statement, we do a similar calculation for the process $\mathcal{SW}(t')$. By definition, this process consists already from time t' onward of a collection of independent random walks. Thus, we can swap the expectation and the product in the last term of the above calculation, yielding the desired result. ◀

Separating the Plurality via Chernoff

▶ **Lemma 9** ([7, Lemma 3.1]). *Let $X_1, X_2, \ldots, X_n$ be a sequence of random variables with values in an arbitrary domain and let $Y_1, Y_2, \ldots, Y_n$ be a sequence of binary random variables with the property that $Y_i = Y_i(X_1, \ldots, X_i)$. If $\Pr(Y_i = 1 \mid X_1, \ldots, X_{i-1}) \leq p$, then*

$$\Pr\left(\sum Y_i \geq \ell\right) \leq \Pr(\mathrm{Bin}(n,p) \geq \ell) \tag{6}$$

and, similarly, if $\Pr(Y_i = 1 \mid X_1, \ldots, X_{i-1}) \geq p$, then

$$\Pr\left(\sum Y_i \leq \ell\right) \leq \Pr(\mathrm{Bin}(n,p) \leq \ell). \tag{7}$$

Here, $\mathrm{Bin}(n,p)$ denotes the binomial distribution with parameters n and p.

We are finally able to prove the following Chernoff-like bound.

▶ **Lemma 10** (Token Concentration). *Consider any subset B of tokens, a node $u \in V$, and an integer T. Let $X := \sum_{t\leq T} \sum_{j\in B} X_{j,t}$, where $X_{j,t}$ is 1 if token j is on node u at time $t \cdot t_{mix}$. With $\mu := (1/n + 1/n^5) \cdot |B| \cdot T$, we have $\Pr(X \geq (1+\delta)\cdot\mu) \leq e^{\delta^2\mu/3}$.*

Proof. Let $v_{j,t}$ denote the location of token j at time $(t-1)\cdot t_{\mathrm{mix}}$. For all $t \leq T$ and $\ell \in \mathbb{N}$ define the random indicator variable $Y_{j,t}$ to be 1 if and only if the random walk starting at

$v_{j,t}$ is at node u after t_{mix} time steps. By Lemma 7 we have for each $B' \subseteq B$ and $t \le T$ that

$$\Pr\left(\bigcap_{i\in B'} X_{j,t}=1\right) \le \prod_{j\in B'} \Pr\left(Y_{j,t}=1\right). \tag{8}$$

Hence for all $t \le T$ and $\ell \in \mathbb{N}$ we have $\Pr\left(\sum_{j\in B} X_{j,t} \ge \ell\right) \le \Pr\left(\sum_{j\in B} Y_{j,t} \ge \ell\right)$ and

$$\Pr\left(X \ge \ell\right) = \Pr\left(\sum_{t\le T}\sum_{j\in B} X_{j,t} \ge \ell\right) \le \Pr\left(\sum_{t\le T}\sum_{j\in B} Y_{j,t} \ge \ell\right). \tag{9}$$

Let us define $p := 1/n + 1/n^5$. By the definition of t_{mix}, we have for all $j \in B$ and $t \le T$ that

$$\Pr\left(Y_{j,t}=1 \mid Y_{1,1}, Y_{2,1}, \ldots, Y_{|B|,1}, Y_{1,2}, \ldots, Y_{j-1,t}\right) \le p. \tag{10}$$

Combining our observations with Lemma 9 (see above), we get $\Pr\left(X \ge \ell\right) \le \mathrm{Bin}(T \cdot |B|, p)$. Recall that $\mu = T \cdot |B| \cdot p$. Thus, by applying standard Chernoff bounds we get

$$\Pr\left(X \ge (1+\delta)\mu\right) \le \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu} \le e^{\delta^2\mu/3}, \tag{11}$$

which yields the desired statement. ◀

Together, these lemmas generalize a result given in [32] to a setting with considerably more dependencies. Equipped with this Chernoff bound, we prove concentration of the counter values.

▶ **Lemma 11** (Counter Seperation). *Let $c \ge 12$. For every time $t \ge c \cdot T \cdot t_{mix}$ there exist values $\ell_\top > \ell_\perp$ such that*
(a) *For all nodes w with $o_w \ge 2$ we have (w.h.p.) $c_w \le \ell_\perp$.*
(b) *For all nodes v with $o_v = 1$ we have (w.h.p.) $c_v \ge \ell_\top$.*

Proof. For two nodes v and w with $o_v = 1$ and $o_w \ge 2$, $\mu_i := (1/n + 1/n^5)c \cdot T \cdot \gamma \cdot n_k$ for all $i \in [k]$, and $\mu' := (1/n + 1/n^5)c \cdot T \cdot \gamma \cdot (n - n_1)$. For $i \in [k]$ define

$$\ell_\perp(i) := \mu_i + \sqrt{c^2 \cdot \log n \cdot T \cdot \gamma \frac{n_i}{n}} \qquad \text{and} \qquad \ell_\top := T\gamma - \mu' - \sqrt{c^2 \cdot \log n \cdot T \cdot \gamma \frac{n-n_1}{n}}.$$

We set $\ell_\perp := \ell_\perp(2)$. It is easy to show that $\ell_\perp < \ell_\top$. Now, let all γn tokens be labeled from 1 to γn. It remains to prove the lemma's statements:

- For the first statement, consider a node w with $o_w \ge 2$ and set $\lambda(o_w) := \ell_\perp(o_w) - \mu_{o_w} = \sqrt{c^2 \cdot \log n \cdot T \cdot \gamma \cdot n_{o_w}/n}$. Set the random indicator variable $X_{i,t}$ to be 1 if and only if i is on node w at time t and if i's label is o_w. Let $c_w = \sum_{i\in[\gamma n]}\sum_{j\le T} X_{i,j\cdot t_{\mathrm{mix}}}$. We compute

$$\begin{aligned}\Pr\left(c_w \ge \ell_\perp\right) \le \Pr\left(c_w \ge \mu_{o_w} + \lambda(o_w)\right) &= \Pr\left(c_w \ge \left(1 + \frac{\lambda(o_w)}{\mu_{o_w}}\right) \cdot \mu_{o_w}\right) \\ &\le \exp\left(-\frac{\lambda^2(o_w)}{3\mu_{o_w}}\right) \le \exp\left(-\frac{c}{6}\log n\right),\end{aligned} \tag{12}$$

  where the last line follows by Lemma 10 applied to $c_w = \sum_{i\in[\gamma n]}\sum_{j\le T} X_{i,j\cdot t_{\mathrm{mix}}}$ and setting B to the set of all tokens with label o_w. Hence, the claim follows for c large enough after taking the union bound over all $n - n_1 \le n$ nodes w with $o_w \ge 2$.

- For the lemma's second statement, consider a node v with $o_v = 1$ and set $\lambda' := \mu' - \ell_\top$. Define the random indicator variable $Y_{i,t}$ to be 1 if and only if token i is on node v at time t and if i's label is not 1. Set $Y = \sum_{j \le T} \sum_{i \in [\gamma n]} Y_{i,j \cdot t_{\text{mix}}}$ and note that $c_v = T\gamma - Y$. We compute

$$\Pr(c_v \le \ell_\top) = \Pr(T\gamma - Y \le \ell_\top) = \Pr(T\gamma - Y \le T\gamma - \mu' - \lambda') = \Pr(Y \ge \mu' + \lambda')$$
$$= \Pr\left(Y \ge \left(1 + \frac{\lambda'}{\mu'}\right) \cdot \mu'\right) \le \exp\left(-\frac{\lambda'^2}{3\mu'}\right) \le \exp\left(\frac{c}{6}\log n\right),$$

where the first inequality follows by Lemma 10 applied to Y and using B to denote the set of all tokens with a label other than 1. Hence, the claim follows for c large enough after taking the union bound over all $n_1 \le n$ nodes v with $o_u \ge 2$. ◀

We now give the proof of our main theorem.

Proof of Theorem 2. Fix an arbitrary time $t \in [c \cdot T \cdot t_{\text{mix}}, N]$ with $t_{\text{mix}} \mid t$, where c is the constant from the statement of Lemma 11. From Lemma 11 we have that (w.h.p.) the node u with the highest counter c_u has $o_u = 1$ (ties are broken arbitrarily). In the following we condition on $o_u = 1$. We claim that at time $t' = t + t_{\text{mix}}$ all nodes $v \in V$ have $plu_v = 1$. This is because the counters during the "broadcast part" (Lines 4 to 8) propagate the highest counter received after time t. The time τ until all nodes $v \in V$ have $plu_v = 1$ is bounded by the mixing by definition: In order for $[t, t']$ to be $1/n^5$-smoothing, the random walk starting at u at time t is with probability at least $1/n - 1/n^5$ on node v and, thus, there exists a path from u to v (with respect to the communication matrices). If there is such a path for every node v, the counter of u was also propagated to that v and we have $\tau \le t_{\text{mix}}$. Consequently, at time t' all nodes have the correct majority opinion. This implies the desired time bound. For the memory requirements, note that each node u stores γ tokens with a label from the set $[k]$ ($\gamma \cdot \mathrm{O}(\log k)$ bits), three opinions (its own, its plurality guess, and the dominating opinion; $\mathrm{O}(\log k)$ bits), the two counters c_u and e_u and the time step counter. The memory to store the counter c_u and e_u is $\mathrm{O}(\gamma T)$. Finally, the time step counter is bounded by $\mathrm{O}(\log(T \cdot t_{\text{mix}}))$ bits. This yields the claimed space bound. ◀

4 Protocol Balance

Protocol Description. The idea of our Balance protocol is quite simple: Every node u stores a k-dimensional vector $\boldsymbol{\ell_t(u)}$ with k integer entries, one for each opinion. Balance performs an entry-wise load balancing on $\boldsymbol{\ell_t(u)}$ according to the communication pattern $\boldsymbol{M} = (\boldsymbol{M}_t)_{t \le N}$ and the corresponding transition matrices $\boldsymbol{P}_t$ (cf. Section 2). Once the load is properly balanced, the nodes look at their largest entry and assume that this is the plurality opinion (stored in the variable plu_u).

In order to ensure a low memory footprint, we must not send fractional loads over active edges. To this end, we use a rounding scheme from [12, 32], which works as follows: Consider a dimension $i \in [k]$ and let $\ell_{i,t}(u) \in \mathbb{N}$ denote the current (integral) load at u in dimension i. Then u sends $\lfloor \ell_{i,t}(u) \cdot \boldsymbol{P}_t[u,v] \rfloor$ tokens to all neighbors v with $\boldsymbol{M}_t[u,v] = 1$. This results in at most $d_t(u)$ remaining *excess tokens* ($\ell_{i,t}(u)$ minus the total number of tokens sent out). These are then randomly distributed (without replacement), where neighbor v receives a token with probability $\boldsymbol{P}_t[u,v]$. In the following we call the resulting balancing algorithm *vertex-based balancing* algorithm. The formal description of protocol Balance is given in Listing 2.

```
for $i \in [k]$:
  for $\{u, v\} \in E$ with $\boldsymbol{M}_t[u,v] = 1$:
    send $\lfloor \ell_{i,t}(u) \cdot \boldsymbol{P}_t[u,v] \rfloor$ tokens from dimension $i$ to $v$
  $x := \ell_{i,t}(u) - \sum_{v:\, \boldsymbol{M}_t[u,v]=1} \lfloor \ell_{i,t}(u) \cdot \boldsymbol{P}_t[u,v] \rfloor$        {excess tokens}
  randomly distribute $x$ tokens such that:
    every $v \neq v$ with $\boldsymbol{M}_t[u,v] = 1$ receives 1 token w.p. $\boldsymbol{P}_t[u,v]$
    (and zero otherwise)
$plu_u := i$ with $\ell_{i,t}(u) \geq \ell_{j,t}(u) \quad \forall 1 \leq i,j \leq k$        {plurality guess}
```

Listing 2 Protocol Balance as executed by node u at time t. At time zero, each node initializes $\ell_{o_u,0}(u) := \gamma$ and $\ell_{j,0}(u) := 0$ for all $j \neq o_u$.

Analysis of Balance. Consider initial load vectors $\boldsymbol{\ell_0}$ with $\|\boldsymbol{\ell_0}\|_\infty \leq n^5$. Let $\tau := \tau(g, \boldsymbol{M})$ be the first time step when Vertex-Based Balancer under the (fixed) communication pattern $\boldsymbol{M} = (\boldsymbol{M}_t)_{t \leq N}$ is able to balance any such vector $\boldsymbol{\ell_0}$ up to a g-discrepancy. With this, we show:

▶ **Theorem 12.** *Let $\alpha = \frac{n_1 - n_2}{n} \in [1/n, 1]$ denote the initial bias. Consider a fixed communication pattern $\boldsymbol{M} = (\boldsymbol{M}_t)_{t \leq N}$ and an integer $\gamma \in [3 \cdot \frac{g}{\alpha}, n^5]$. Protocol* Balance *ensures that all nodes know the plurality opinion after $\tau(g, \boldsymbol{M})$ rounds and requires $k \cdot \log(\gamma)$ memory bits per node.*

Proof. Recall that $\gamma \geq 3\frac{g}{\alpha} = 3g \cdot \frac{n}{n_1 - n_2}$. For $i \in [k]$ let $\bar{\ell}_i := n_i \cdot \gamma / n$. The definition of $\tau(g, \boldsymbol{M})$ implies $\ell_{1,t}(u) \geq \bar{\ell}_1 - g$ and $\ell_{i,t}(u) \leq \bar{\ell}_i + g$ for all nodes u and $i \geq 2$. Consequently, we get

$$\ell_{1,t}(u) - \ell_{i,t}(u) \geq \bar{\ell}_1 - \bar{\ell}_i - 2g = 3g \cdot \frac{n_1 - n_i}{n_1 - n_2} - 2g > 0\,. \tag{13}$$

Thus, every node u has the correct plurality guess at time t. ◀

The memory usage of Balance depends on the number of opinions (k) and on the number of tokens generated on every node (γ). The algorithm is very efficient for small values of k but it becomes rather impractical if k is large. Note that if one chooses γ sufficiently large, it is easy to adjust the algorithm such that every node knows the frequency of *all* opinions in the network. The next corollary gives a few concrete examples for common communication patterns on general graphs.

▶ **Corollary 13.** *Let G be an arbitrary d-regular graph.* Balance *ensures that all nodes agree on the plurality opinion with probability $1 - e^{-(\log(n))^c}$ for some constant c*

(a) *using $\mathrm{O}(k \cdot \log n)$ bits of memory in time $\mathrm{O}\big(\frac{\log n}{1-\lambda_2}\big)$ in the diffusion model,*

(b) *using $\mathrm{O}(k \cdot \log n)$ bits of memory in time $\mathrm{O}\big(\frac{1}{d \cdot p_{min}} \cdot \frac{\log n}{1-\lambda_2}\big)$ in the random matching model,*

(c) *using $\mathrm{O}\left(k \cdot \log(\alpha^{-1})\right)$ bits of memory in time $\mathrm{O}\big(d \cdot \frac{\log n}{1-\lambda_2}\big)$ in the balancing circuit model, and*

(d) *using $\mathrm{O}\left(k \cdot \log(\alpha^{-1})\right)$ bits of memory in time $\mathrm{O}\big(n \cdot \frac{\log n}{1-\lambda_2}\big)$ in the sequential model.*

Proof. Part (a) follows directly from [33, Theorem 6.6] and Part (c) follows directly from [33, Theorem 1.1]. To show Part (b) and (d) we choose τ such that $\boldsymbol{M}_1, \boldsymbol{M}_2, \ldots, \boldsymbol{M}_\tau$ enable Vertex-Based Balancer to balance any vector $\boldsymbol{\ell_0}$ (with initial discrepancy of at most n^5) up to a *g-discrepancy*. The bound on τ then follows from [33, Theorem 1.1]. ◀

References

1 D. Aldous and J. Fill. Reversible markov chains and random walks on graphs, 2002. Unpublished. http://www.stat.berkeley.edu/~aldous/RWG/book.html.

2 Dan Alistarh, Rati Gelashvili, and Milan Vojnovic. Fast and exact majority in population protocols. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, (PODC)*, pages 47–56, 2015.

3 Dana Angluin, James Aspnes, and David Eisenstat. A simple population protocol for fast robust approximate majority. *Distributed Computing*, 21(2):87–102, 2008.

4 Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, 2007.

5 James Aspnes and Eric Ruppert. An introduction to population protocols. *Bulletin of the EATCS*, 93:98–117, 2007.

6 Chen Avin, Michal Koucký, and Zvi Lotker. How to explore a fast-changing world (cover time of a simple random walk on evolving graphs). In *Automata, Languages and Programming, 35th International Colloquium, ICALP*, pages 121–132, 2008.

7 Yossi Azar, Andrei Z. Broder, Anna R. Karlin, and Eli Upfal. Balanced allocations. *SIAM Journal of Computing*, 29(1):180–200, 1999.

8 Luca Becchetti, Andrea Clementi, Emanuele Natale, Francesco Pasquale, and Riccardo Silvestri. Plurality consensus in the gossip model. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 371–390, 2015. doi:10.1137/1.9781611973730.27.

9 Luca Becchetti, Andrea E. F. Clementi, Emanuele Natale, Francesco Pasquale, Riccardo Silvestri, and Luca Trevisan. Simple dynamics for plurality consensus. In *26th ACM Symposium on Parallelism in Algorithms and Architectures, (SPAA)*, pages 247–256, 2014. doi:10.1145/2612669.2612677.

10 Florence Bénézit, Patrick Thiran, and Martin Vetterli. Interval consensus: From quantized gossip to voting. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP*, pages 3661–3664. IEEE, 2009.

11 Florence Bénézit, Patrick Thiran, and Martin Vetterli. The distributed multiple voting problem. *J. Sel. Topics Signal Processing*, 5(4):791–804, 2011.

12 Petra Berenbrink, Colin Cooper, Tom Friedetzky, Tobias Friedrich, and Thomas Sauerwald. Randomized diffusion for indivisible loads. *J. Comput. Syst. Sci.*, 81(1):159–185, 2015.

13 Petra Berenbrink, Tom Friedetzky, George Giakkoupis, and Peter Kling. Efficient plurality consensus, or: The benefits of cleaning up from time to time. In *Proceedings of the 43rd International Colloquium on Automata, Languages and Programming (ICALP)*, 2016. to appear.

14 Stephen P. Boyd, Arpita Ghosh, Balaji Prabhakar, and Devavrat Shah. Randomized gossip algorithms. *IEEE Transactions on Information Theory*, 52(6):2508–2530, 2006.

15 Luca Cardelli and Attila Csikász-Nagy. The cell cycle switch computes approximate majority. *Scientific reports*, 2, 2012.

16 Yuan-Jyue Chen, Neil Dalchau, Niranjan Srinivas, Andrew Phillips, Luca Cardelli, David Soloveichik, and Georg Seelig. Programmable chemical controllers made from dna. *Nature nanotechnology*, 8(10):755–762, 2013.

17 Andrea E. F. Clementi, Miriam Di Ianni, Giorgio Gambosi, Emanuele Natale, and Riccardo Silvestri. Distributed community detection in dynamic graphs. *Theor. Comput. Sci.*, 584:19–41, 2015.

18 Colin Cooper, Robert Elsässer, and Tomasz Radzik. The power of two choices in distributed voting. In *Automata, Languages, and Programming - 41st International Colloquium, (ICALP)*, pages 435–446, 2014.

19 Colin Cooper, Robert Elsässer, Tomasz Radzik, Nicolas Rivera, and Takeharu Shiraga. Fast consensus for voting on general expander graphs. In *Proceedings of the 29th International Symposium on Distributed Computing (DISC)*, pages 248–262, 2015. `doi:10.1007/978-3-662-48653-5_17`.

20 Benjamin Doerr, Leslie Ann Goldberg, Lorenz Minder, Thomas Sauerwald, and Christian Scheideler. Stabilizing consensus with the power of two choices. In *Proceedings of the 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures, (SPAA)*, pages 149–158, 2011.

21 Moez Draief and Milan Vojnovic. Convergence speed of binary interval consensus. *SIAM J. Control and Optimization*, 50(3):1087–1109, 2012.

22 Devdatt P. Dubhashi and Desh Ranjan. Balls and bins: A study in negative dependence. *Random Struct. Algorithms*, 13(2):99–124, 1998.

23 Robert Elsässer, Tom Friedetzky, Dominik Kaaser, Frederik Mallmann-Trenn, and Horst Trinker. Efficient k-party voting with two choices. *CoRR*, abs/1602.04667, 2016. URL: `http://arxiv.org/abs/1602.04667`.

24 Bhaskar Ghosh and S. Muthukrishnan. Dynamic load balancing by random matchings. *J. Comput. Syst. Sci.*, 53(3):357–370, 1996. `doi:10.1006/jcss.1996.0075`.

25 David Kempe, Alin Dobra, and Johannes Gehrke. Gossip-based computation of aggregate information. In *Proceedings 44th Symposium on Foundations of Computer Science (FOCS)*, pages 482–491, 2003.

26 Fabian Kuhn, Thomas Locher, and Stefan Schmid. Distributed computation of the mode. In *Proceedings of the 26th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 15–24, 2008.

27 George B. Mertzios, Sotiris E. Nikoletseas, Christoforos Raptopoulos, and Paul G. Spirakis. Determining majority in networks with local interactions and very small local memory. In *Automata, Languages, and Programming - 41st International Colloquium, (ICALP)*, pages 871–882, 2014.

28 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. `doi:10.1007/s10458-013-9230-4`.

29 Elchanan Mossel and Grant Schoenebeck. Reaching consensus on social networks. In *Innovations in Computer Science - (ICS)*, pages 214–229, 2010.

30 David Peleg. Local majorities, coalitions and monopolies in graphs: a review. *Theor. Comput. Sci.*, 282(2):231–257, 2002. `doi:10.1016/S0304-3975(01)00055-X`.

31 Etienne Perron, Dinkar Vasudevan, and Milan Vojnovic. Using three states for binary consensus on complete graphs. In *28th IEEE International Conference on Computer Communications, (INFOCOM)*, pages 2527–2535, 2009.

32 Thomas Sauerwald and He Sun. Tight bounds for randomized load balancing on arbitrary network topologies. In *53rd Annual IEEE Symposium on Foundations of Computer Science, (FOCS)*, pages 341–350, 2012.

33 Thomas Sauerwald and He Sun. Tight bounds for randomized load balancing on arbitrary network topologies. *CoRR*, abs/1201.2715, 2012. full version of FOCS'12. `arXiv:1201.2715`.

On the Hardness of Learning Sparse Parities

Arnab Bhattacharyya[*1], **Ameet Gadekar**[2], **Suprovat Ghoshal**[3], **and Rishi Saket**[4]

1 **Department of Computer Science and Automation, Indian Institute of Science, Bangalore, India**
arnabb@csa.iisc.ernet.in
2 **Department of Computer Science and Automation, Indian Institute of Science, Bangalore, India**
ameet.gadekar@csa.iisc.ernet.in
3 **Department of Computer Science and Automation, Indian Institute of Science, Bangalore, India**
suprovat.ghoshal@csa.iisc.ernet.in
4 **IBM Research, Bangalore, India**
rissaket@in.ibm.com

Abstract

This work investigates the hardness of computing sparse solutions to systems of linear equations over $\mathbb{F}_2$. Consider the k-EvenSet problem: given a homogeneous system of linear equations over $\mathbb{F}_2$ on n variables, decide if there exists a nonzero solution of Hamming weight at most k (i.e. a k-sparse solution). While there is a simple $O(n^{k/2})$-time algorithm for it, establishing fixed parameter intractability for k-EvenSet has been a notorious open problem. Towards this goal, we show that unless k-Clique can be solved in $n^{o(k)}$ time, k-EvenSet has no polynomial time algorithm when $k = \omega(\log^2 n)$.

Our work also shows that the non-homogeneous generalization of the problem – which we call k-VectorSum – is W[1]-hard on instances where the number of equations is $O(k \log n)$, improving on previous reductions which produced $\Omega(n)$ equations. We use the hardness of k-VectorSum as a starting point to prove the result for k-EvenSet, and additionally strengthen the former to show the hardness of *approximately learning k-juntas*. In particular, we prove that given a system of $O(\exp(O(k)) \cdot \log n)$ linear equations, it is W[1]-hard to decide if there is a k-sparse linear form satisfying all the equations or any function on at most k-variables (a k-junta) satisfies at most $(1/2 + \varepsilon)$-fraction of the equations, for any constant $\varepsilon > 0$. In the setting of computational learning, this shows hardness of approximate *non-proper* learning of k-parities. In a similar vein, we use the hardness of k-EvenSet to show that that for any constant d, unless k-Clique can be solved in $n^{o(k)}$ time, there is no $\mathrm{poly}(m, n) \cdot 2^{o(\sqrt{k})}$ time algorithm to decide whether a given set of m points in $\mathbb{F}_2^n$ satisfies: (i) there exists a non-trivial k-sparse homogeneous linear form evaluating to 0 on all the points, or (ii) any non-trivial degree d polynomial P supported on at most k variables evaluates to zero on $\approx \mathbf{Pr}_{\mathbb{F}_2^n}[P(\mathbf{z}) = 0]$ fraction of the points i.e., P is *fooled* by the set of points.

Lastly, we study the approximation in the sparsity of the solution. Let the Gap-k-VectorSum problem be: given an instance of k-VectorSum of size n, decide if there exist a k-sparse solution, or every solution is of sparsity at least $k' = (1 + \delta_0)k$. Assuming the Exponential Time Hypothesis, we show that for some constants c_0, $\delta_0 > 0$ there is no $\mathrm{poly}(n)$ time algorithm for Gap-k-VectorSum when $k = \omega((\log \log n)^{c_0})$.

1998 ACM Subject Classification F.2.1 Analysis of Algorithms and Problem Complexity: Numerical Algorithms and Problems – Computations in finite fields

[*] AB was supported in part by DSTO1358 Ramanujan Fellowship.

Keywords and phrases Fixed Parameter Tractable, Juntas, Minimum Distance of Code, Psuedorandom Generators

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.11

1 Introduction

Given a system of linear equations over $\mathbb{F}_2$, does there exist a sparse non-trivial solution? This question is studied in different guises in several areas of mathematics and computer science. For instance, in coding theory, if the system of linear equations is $\mathbf{Mx} = \mathbf{0}$ where $\mathbf{M}$ is the parity check matrix of a binary code, then the minimum (Hamming) weight of a nonzero solution is the distance of the code. This also captures the problem of determining whether a binary matroid has a short cycle, as the latter reduces to deciding whether there is a sparse nonzero $\mathbf{x}$ such that $\mathbf{Mx} = \mathbf{0}$. In learning theory, the well known sparse parity problem is: given a binary matrix $\mathbf{M}$ and a vector $\mathbf{b}$ decide whether there is a small weight nonzero vector $\mathbf{x}$ satisfying $\mathbf{Mx} = \mathbf{b}$. The version where $\mathbf{Mx}$ is required to equal $\mathbf{b}$ in most coordinates, but not necessarily all, is also well studied as the problem of learning noisy parities.

Let a vector $\mathbf{x} \in \mathbb{F}_2^n$ be called *k-sparse* if it is nonzero in at most k positions, i.e. it has Hamming weight at most k. In this work, we show that learning a k-sparse solution to a system of linear equations is fixed parameter intractable, even when (i) the number of equations is only *logarithmic* in the number of variables, (ii) the learning is allowed to be *approximate*, i.e. satisfy only 51% of the equations and, (iii) is allowed to output as hypothesis any function (junta) supported on at most k variables. We also prove variants of these results for the case when the system of equations is homogeneous, which correspond to hardness of the well known k-EvenSet problem. Note that it is always possible to recover a k-sparse solution in $O(n^k)$ time simply by enumerating over all k-sparse vectors. Our results show that for many settings of k, no substantially faster algorithm is possible for k-EvenSet unless widely believed conjectures are false. Assuming similar conjectures, we also rule out fast algorithms for learning γk-sparse solutions to a linear system promising the existence of a k sparse solutions, for some $\gamma > 1$.

In the next few paragraphs we recall previous related work and place our results in their context. Let us first formally define the basic objects of our study:

▶ **Definition 1.** k-VectorSum: Given a matrix $\mathbf{M} \in \mathbb{F}_2^{m\times n}$ and a vector $\mathbf{b} \in \mathbb{F}_2^m$, and a positive integer k as parameter, decide if there exists a k-sparse vector $\mathbf{x}$ such that $\mathbf{Mx} = \mathbf{b}$.

▶ **Definition 2.** k-EvenSet: Given a matrix $\mathbf{M} \in \mathbb{F}_2^{m\times n}$, and a positive integer k as parameter, decide if there exists a nonzero k-sparse vector $\mathbf{x}$ such that $\mathbf{Mx} = \mathbf{0}$.

▶ Remark. In the language of coding theory, k-VectorSum is also known as the Maximum-LikelihoodDecoding problem and k-EvenSet as the MinimumDistance problem.

Clearly, k-VectorSum is as hard as k-EvenSet[1]. The k-VectorSum problem was shown to be W[1]-hard[2] by Downey, Fellows, Vardy and Whittle [15], even in the special

[1] The name k-EvenSet is from the following interpretation of the problem: given a set system $\mathcal{F}$ over a universe U and a parameter k, find a nonempty subset $S \subseteq U$ of size at most k such that the intersection of S with every set in $\mathcal{F}$ has even size.

[2] Standard definitions in parameterized complexity appear in Section 2.

case of the vector $\mathbf{b}$ consisting of all 1's. More recently, Bhattacharyya, Indyk, Woodruff, and Xie [6] showed that the time complexity of k-VectorSum is at least $\min(2^{\Theta(m)}, n^{\Theta(k)})$, assuming the Exponential Time Hypothesis (i.e., 3-SAT has no $2^{o(n)}$ time algorithm) [24].

In contrast, the complexity of k-EvenSet remains unresolved, other than its containment in W[2] shown in [15]. Proving W[1]-hardness for k-EvenSet was listed as an open problem in Downey and Fellows' 1999 monograph [17] and has been reiterated more recently in lists of open problems [21, 19]. Note that if we ask for a vector $\mathbf{x}$ whose weight is *exactly* k instead of at most k, the problem is known to be W[1]-hard [15]. Our work gives evidence ruling out efficient algorithms for k-EvenSet for a wide range of settings of k.

In the non-parameterized setting, where k is part of the input, these problems are very well-studied. Vardy showed that EvenSet (or MinimumDistance) is NP-hard [33]. The question of approximating k, the minimum distance of the associated code, has also received attention. Dumer, Micciancio, and Sudan [18] showed that if $\mathsf{RP} \neq \mathsf{NP}$, then k is hard to approximate within some constant factor $\gamma > 1$. Cheng and Wan [11, 12] proved the same assuming $\mathsf{P} \neq \mathsf{NP}$, and subsequently Austrin and Khot [4] gave a simpler deterministic reduction for this problem. The results of [11, 12] and [4] were further strengthened by Micciancio [30].

From a computational learning perspective, the k-VectorSum problem can be restated as: given an m-sized set of n-dimensional point and value pairs (i.e. elements of $\mathbb{F}_2^n \times \mathbb{F}_2$), decide if there exists a parity (i.e. a homogeneous linear form) supported on at most k variables (i.e. a k-parity) that is consistent with all the pairs. This has been extensively studied as a promise problem when the points are generated uniformly at random. Note that in this case, if $m = \Omega(n)$, there is a unique solution w.h.p and can be found efficiently by Gaussian elimination. On the other hand, for $m = O(k \log n)$, the best known running time of $O(n^{k/2})$ is given in [28] (credited to Dan Spielman). Obtaining a polynomial time algorithm for $m = \text{poly}(k \log n)$ would imply *attribute-efficient learning* of k-parities and is a long-standing open problem in the area [7].

A natural question studied in this work is whether one can do better if the learning algorithm is allowed to be *non-proper* (i.e., output a hypothesis that is not a k-parity) and is allowed to not satisfy all the point-value pairs. To further motivate this problem, let us look at the case when k is not fixed along with a promise that there exists a parity consistent with $1 - \delta$ (for some constant $\delta > 0$) fraction of the point-value pairs, i.e., the agnostic setting. When the points are adversarially drawn, there is no non-trivial proper algorithm known but there is a non-proper algorithm due to Kalai, Mansour, and Verbin [25] that runs in time $2^{O(n/\log n)}$ and outputs a circuit C consistent with at least $\left(1 - \delta - 2^{-n^{0.99}}\right)$ of the point-value pairs. On the hardness side, Håstad's inapproximability for Max-3LIN [23] implies that properly learning a noisy parity in the agnostic setting is NP-hard, even for $1/2+\varepsilon$ accuracy, for any constant $\varepsilon > 0$. Nearly a decade later, Gopalan, Khot, and Saket [22] showed that achieving an accuracy of $1-1/2^d+\varepsilon$ using degree-d polynomials as hypotheses is NP-hard and subsequently, Khot [26] proved NP-hardness for learning with accuracy $1/2+\varepsilon$ using constant degree polynomials[3]. Our work studies the intractability of approximate non-proper learning of k-parity and extends the hardness result for k-VectorSum to learning by juntas of k variables and for k-EvenSet to learning using constant degree polynomials on k variables.

[3] As far as we know, this result is unpublished although it was communicated to the fourth author of this paper. The full version of this paper [5] includes a proof of Khot's result with his permission to illustrate some of the techniques which inspire part of this work.

Another interesting question in the parameterized setting is related to a gap in the sparsity parameter k, i.e. how tractable it is to learn a γk-sparse solution when the existence of a k-sparse solution is guaranteed, for some constant $\gamma > 1$. Previously, Bonnet et al. [8] and Khot and Shinkar [27] studied the approximation problem corresponding to k-Clique, and both these works show conditional hardness results. More generally, there have been several previous works studying approximation algorithms with the optimum value being a parameter; see references in Marx's survey [29]. In our work, we prove a "gap in k" hardness result for k-VectorSum similar to that obtained in [8] for k-Clique.

In the rest of this section we formally describe our results for k-VectorSum and k-EvenSet, and give a brief description of the techniques used to obtain them.

1.1 Our Results

All the reductions given in this section run in time polynomial in the size of the output instances. We do not make this explicit for ease of notation.

Hardness of exact problems

The main result of this paper is the following hardness reduction from k-VectorSum to the k-EvenSet problem.

▶ **Theorem 3** (Hardness of k-EvenSet). *There is an* FPT *reduction from an instance* $(\mathbf{M}, \mathbf{t})$ *of* k-VectorSum, *where* $\mathbf{M} \in \mathbb{F}_2^{m \times n}$ *and* $\mathbf{t} \in \mathbb{F}_2^m$, *to an instance* $\mathbf{M}'$ *of* $O((k \log n)^2)$-EvenSet, *where* $\mathbf{M}' \in \mathbb{F}_2^{m' \times n'}$ *such that both* m' *and* n' *are bounded by fixed polynomials in* m *and* n.

Combined with the W[1]-hardness of k-VectorSum ([15, 13] or Theorem 5 below), the above yields the following corollary.

▶ **Corollary 4.** *There does not exist a* $\text{poly}(n)$ *time algorithm for* k-EvenSet *when* $k = \omega(\log^2 n)$, *assuming that* k-Clique *does not have a polynomial time algorithm for any* $k = \omega_n(1)$. *More generally, under the same assumption,* k-EvenSet *does not admit a* $\text{poly}(n) \cdot 2^{o(\sqrt{k})}$ *time algorithm for unrestricted* k.

Proof. Suppose there is a $T(n, k)$ algorithm for k-EvenSet. Chaining the W[1]-hardness of k-VectorSum from Theorem 5 with the reduction in Theorem 3, we obtain a $T\left(\text{poly}(n), O\left((k^2 \log n)^2\right)\right)$ algorithm for k-Clique. Choosing $k = \omega_n(1)$ implies the first part of the corollary. For the second part, observe that if $f(x) = 2^{o(\sqrt{x})}$, then we have $f\left((k^2 \log n)^2\right) = n^{o(1)}$ for some $k = \omega_n(1)$. ◀

To the best of our knowledge, Corollary 4 gives the first nontrivial hardness results for parameterized k-EvenSet. Theorem 3 is obtained by adapting the hardness reduction for the inapproximability of MinimumDistance by Austrin and Khot [4] to the parameterized setting.

We also give a reduction from k-Clique showing the W[1]-hardness of k-VectorSum on instances which have a small number of rows.

▶ **Theorem 5** (W[1]-hardness of k-VectorSum). *The* k-VectorSum *problem is* W[1]*-hard on instances* $(\mathbf{M}, \mathbf{b})$ *where* $\mathbf{M} \in \mathbb{F}_2^{m \times n}$ *and* $\mathbf{b} \in \mathbb{F}_2^m$ *such that* $m = O(k \log n)$. *This is obtained by an* FPT *reduction from an* r*-vertex instance of* ℓ-Clique *to* $(\mathbf{M}, \mathbf{b})$ *such that* $m = O(\ell^2 \log r)$, $n = O((\ell r)^2)$ *and* $k = \Theta(\ell^2)$. *Our reduction implies, in particular, that* k-VectorSum *does not admit an* $n^{o(\sqrt{k})}$ *time algorithm on such instances, unless* k-Clique *on* r*-vertex graphs has an* $r^{o(k)}$ *time algorithm.*

As far as we know, in previous proofs of the W[1]-hardness of k-VectorSum [15, 13], the number of rows in the matrix output by the reduction was linear in n. Our proof is inspired by a recent proof of the W[1]-hardness of k-Sum [1]. Additionally, in Section 7 , we give a simple $O(n \cdot 2^m)$ time algorithm for k-VectorSum, which suggests that m cannot be made sublogarithmic in n for hard instances. The logarithmic upper bound on the number of equations in our hardness reduction to k-VectorSum also leads to a similarly efficient W[1]-hardness of approximate non-proper learning of k-parities (Theorem 7 below) which uses Theorem 5 as the starting point.

Hardness of non-proper and approximately learning sparse parities

Theorem 5 can be restated in terms of W[1]-hardness of learning a k-parity[4].

▶ **Theorem 6** (Theorem 5 restated). *The following is* W[1]*-hard: given $m = O(k \log n)$ point-value pairs $\{(\mathbf{y}_i, a_i)\}_{i=1}^m \subseteq \mathbb{F}_2^n \times \mathbb{F}_2$, decide whether there exists a k-parity L which satisfies all the point-value pairs, i.e., $L(\mathbf{y}_i) = a_i$ for all $i = 1, \ldots, m$.*

Next, we strengthen the above theorem in two ways. We show that the W[1]-hardness holds for learning a k-parity using a k-junta, and additionally for any desired accuracy exceeding 50 Here, a k-junta is any function depending on at most k variables.

▶ **Theorem 7.** *The following is W[1]-hard: for any constant $\delta > 0$, given $m = O(k \cdot 2^{3k} \cdot (\log n)/\delta^3)$ point-value pairs $\{(\mathbf{z}_i, b_i)\}_{i=1}^m \subseteq \mathbb{F}_2^n \times \mathbb{F}_2$, decide whether:*

YES Case. *There exists a k-parity which satisfies all the point-value pairs.*

NO Case. *Any function $f : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ depending on at most k variables satisfies at most $1/2 + \delta$ fraction of the point-value pairs.*

Theorem 7 also implies hardness for *approximately* learning k-juntas as stated in the following corollary:

▶ **Corollary 8.** *There exists no $n^{o(k)}$ time algorithm which given $m = O(k \cdot 2^{3k} \cdot (\log n)/\delta^3)$ point-value pairs $\{(\mathbf{z}_i, b_i)\}_{i=1}^m$, computes a k-junta $f : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ which satisfies at least $1/2 + \delta$ fraction of the point-value pairs, unless k-Clique on n vertices can be solved in $n^{o(k)}$ time.*

In comparison, the problem of *exactly* learning k-juntas previously shown to be W[2]-hard by Arvind, Köbler, and Lindner [3]. Note that the current best algorithm for learning k-juntas, even over the uniform distribution, takes $n^{\Omega(k)}$ time [32, 31].

We similarly strengthen Theorem 3 to rule out efficient algorithms for approximately learning a k-sparse solution to a homogeneous linear system using constant degree polynomials supported on at most k variables.

▶ **Theorem 9.** *For any constants $\delta > 0$ and positive integer d, given an instance $(\mathbf{A}, \mathbf{b})$ of k'-VectorSum, where $\mathbf{A} \in \mathbb{F}_2^{m' \times n'}$ and $\mathbf{b} \in \mathbb{F}_2^{m'}$, there is an* FPT *reduction to a set of m points $\{\mathbf{z}_i\}_{i=1}^m \subseteq \mathbb{F}_2^n$ such that for some $k = O((k' \log n')^2)$,*

YES Case. *There exists a non-trivial k-parity L such that $L(\mathbf{z}_i) = 0$ for all $i = 1, \ldots, m$.*

[4] Note that Theorem 5 as stated shows hardness of learning homogeneous k-sparse linear forms (without the constant term). The result can easily be made to hold for learning by general k-sparse linear forms by adding a point-value pair which is $(\mathbf{0}, 0)$.

NO Case. *Any degree d polynomial $P : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ depending on at most k variables satisfies $P(\mathbf{z}_i) = 0$ for at most $\left(\mathbf{Pr}_{\mathbf{z}\in_U \mathbb{F}_2^n}[P(\mathbf{z}) = 0] + \delta\right)$ fraction of the points*[5]*, where $\mathbf{z} \in_U \mathbb{F}_2^n$ is sampled u.a.r.*

In the above m and n are bounded by polynomials in m' and n'.

In particular, if we assume that k-Clique does not have a poly(n) time algorithm for any $k = \omega(1)$, then for any constant $\delta > 0$ and positive integer d there is no poly$(m, n) \cdot 2^{o(\sqrt{k})}$ time algorithm to decide whether a given set of points $\{\mathbf{z}_i\}_{i=1}^m \subseteq \mathbb{F}_2^n$ satisfies the YES or the NO case. The proof of Theorem 9 relies on an application of Viola's [34] pseudorandom generator for constant degree polynomials, and is inspired by Khot's [26] NP-hardness of learning linear forms using constant degree polynomials.

Gap in sparsity parameter

Using techniques similar to those employed in [8], we prove the following *gap in k* hardness for k-VectorSum, i.e., hardness of Gap-k-VectorSum.

▶ **Theorem 10.** *Assuming the Exponential Time Hypothesis, there are universal constants $\delta_0 > 0$ and c_0 such that there is no* poly(N) *time algorithm to determine whether an instance of* Gap-k-VectorSum *of size N admits a solution of sparsity k or all solutions are of sparsity at least $(1 + \delta_0)k$, for any $k = \omega((\log\log N)^{c_0})$. More generally, under the same assumption, this problem does not admit an $N^{O(k/\omega((\log\log N)^{c_0}))}$ time algorithm for unrestricted k.*

1.2 Our Techniques

Our reduction for proving Theorem 3 proceeds by homogenizing a W[1]-hard instance of k-VectorSum by including $\mathbf{b}$ as a column of $\mathbf{M}$. To force the solution to always choose $\mathbf{b}$, we use the approach of Austrin and Khot [4] who face the same issue when reducing to the MinimumDistance problem. But since we need to retain the bound on the sparsity of the solution, we cannot use their techniques directly. Instead, for a purported sparse solution $\mathbf{x}$, we construct a small length sketch $\mathbf{y}$ that also belongs to an ε-balanced code $\mathcal{C}$. Now, consider $\mathbf{Y}$ that supposedly equals $\mathbf{y}\mathbf{y}^{\mathbf{T}}$. Note that we can check through a system of linear constraints that $\mathbf{Y}$ belongs to the tensor product code $\mathcal{C} \otimes \mathcal{C}$. We then proceed as in [4] to ensure that in the soundness analysis, $\mathbf{Y}$ has non-trivially large weight whenever $\mathbf{x}$ is set to $\mathbf{0}$, implying that the derived k-EvenSet instance is unsatisfiable. Our construction inflates the parameter k to $O((k \log n)^2)$.

The proof of Theorem 5 is based on a gadget reduction from an n-vertex instance of k-Clique creating columns of $\mathbf{M}$ corresponding to the vertices and edges of the graph along with a target vector $\mathbf{b}$. Unlike previous reductions in which the number of coordinates (rows of $\mathbf{M}$) are linear in the number of vertices, we reuse the same set of coordinates for the vertices and edges by assigning unique logarithmic length patterns to each vertex. In total we create k columns for each vertex and $\binom{k}{2}$ columns for each edge, using $O(k^2 \log n)$ coordinates. The target vector $\mathbf{b}$ ensures that a solution always has at least $k + \binom{k}{2}$ columns, which suffices in the YES case while the NO case requires strictly more columns to sum to $\mathbf{b}$.

The hardness of approximately learning k-parities with k-juntas given in Theorem 7 is obtained by transforming the instance of Theorem 5 using an ε-balanced code, along with an analysis of the Fourier spectrum of any k-junta on the resulting distribution. In contrast,

[5] Note that $\mathbf{Pr}_{\mathbf{z}\in_U \mathbb{F}_2^n}[P(\mathbf{z}) = 0] \leqslant 1 - 2^{-d}$, for any non-trivial degree d polynomial P.

Theorem 9 is obtained by taking the uniform distribution over the equations in the hard instance of Theorem 3 (appropriately transformed using an ε-balanced code) as an input to Viola's construction [34] of pseudorandom generators for degree d polynomials. Note that the $\exp(k)$ blowup in the reduction for Theorem 7 rules out its use for proving Theorem 9 due to the presence of a $(\log^2 n)$ factor in the sparsity parameter of the instance obtained in Theorem 3. On the other hand, the non-homogeneity of the k-VectorSum problem hinders the use of Viola's pseudorandom generator for proving a version (for degree d polynomials on k variables instead of k-juntas) of Theorem 7 which avoids the $\exp(k)$ blowup.

For Theorem 10, we use the improved *sparsification lemma* of Calabro, Impagliazzo, and Paturi [10] followed by Dinur's almost linear PCP construction [14] to reduce an n-variable 3-SAT instance to $2^{\varepsilon n}$ Gap-3-SAT instances with almost linear in n clauses and variables. For each instance, a corresponding k-VectorSum instance is created by partitioning the clauses into k blocks and adding $\mathbb{F}_2$-valued variables for partial assignments to each block along with non-triviality and consistency equations. In the YES case setting one variable from each block to 1 (i.e. a k-sparse solution) suffices, whereas in the NO case at least γk variables need to be set to 1, for some constant $\gamma > 1$. The parameters are such that an efficient algorithm to decide the YES and NO cases would violate the Exponential Time Hypothesis for 3-SAT.

Organization of the paper. Reducing from a W[1]-hard instance of k-VectorSum, Theorem 3 is proved in Section 3. This is extended in Section 4 to prove Theorem 9. The reduction proving Theorem 7 is given in Section 5, and starts with a hard instance from Theorem 6 (restatement of Theorem 5). Lastly, we given an efficient reduction from k-Clique to k-VectorSum in in Section 6. Due to lack of space, we do not include the proof of Theorem 10, which can be found in the full version [5] of the paper instead.

In the next section we give some definitions and results which shall prove useful for the subsequent proofs.

2 Preliminaries

2.1 Parameterized Complexity

A *parameterization* of a problem is a poly(n)-time computable function that assigns an integer $k \geqslant 0$ to each problem instance x of length n (bits). The pair (x, k) is an instance of the corresponding parameterized problem. The parameterized problem is said to be *fixed parameter tractable (FPT)* if it admits an algorithm that runs in time $f(k) \cdot \text{poly}(n)$ where k is the parameter of the input, n is the size of the input, and f is an arbitrary computable function. The W-*hierarchy*, introduced by Downey and Fellows [16, 17], is a sequence of parameterized complexity classes with $\text{FPT} = \text{W}[0] \subseteq \text{W}[1] \subseteq \text{W}[2] \subseteq \cdots$. It is widely believed that $\text{FPT} \neq \text{W}[1]$.

These hierarchical classes admit notions of completeness and hardness under FPT reductions i.e., $f(k) \cdot \text{poly}(n)$-time transformations from a problem A instance (x, k) where $|x| = n$, to an instance (x', k') of problem B where $|x'| \leqslant f(k) \cdot \text{poly}(n)$ and k' is bounded by $f(k)$. For example, consider the k-Clique problem: given a graph G on n vertices and an integer parameter k, decide if G has a clique of size k. The k-Clique problem is W[1]-complete, and serves as a canonical hard problem for many W[1]-hardness reductions including those in this work.

For a precise definition of the W-hierarchy, and a general background on parameterized algorithms and complexity, see [17, 20, 13].

2.2 Coding Theoretic Tools

Our hardness reductions use some basic results from coding theory. For our purposes, we shall be restricting our attention to *linear* codes over $\mathbb{F}_2$ i.e., those which form linear subspaces. A code $\mathcal{C} \subseteq \mathbb{F}_2^n$ is said to be a $[n, k, d]$-binary linear code if $\mathcal{C}$ forms a k-dimensional subspace of $\mathbb{F}_2^n$ such that all nonzero elements (codewords) in $\mathcal{C}$ are of Hamming weight at least d. We use weight $\mathrm{wt}(\mathbf{x})$ of a codeword $\mathbf{x}$ to denote its Hamming weight, *distance* of a code to denote the minimum weight of any nonzero codeword, and *rate* to denote the fraction k/n. A *generator* matrix $\mathbf{G} \in \mathbb{F}_2^{n\times k}$ for $\mathcal{C}$ is such that $\mathcal{C} = \{\mathbf{Gx} \mid \mathbf{x} \in \mathbb{F}_2^k\}$. Also associated with $\mathcal{C}$ is a *parity check matrix* $\mathbf{G}^\perp \in \mathbb{F}_2^{(n-k)\times n}$ satisfying: $\mathbf{G}^\perp\mathbf{y} = \mathbf{0}$ *iff* $\mathbf{y} \in \mathcal{C}$. We shall use the generator and parity check matrices of well studied code constructions whose properties we state below.

▶ **Theorem 11** (BCH Codes, Theorem 3 [9])**.** *The dimension of the BCH code of block length $n = (2^m - 1)$ and distance d, is at least $\left(n - \lceil \frac{d-1}{2} \rceil m\right)$. Further, the corresponding parity check matrix is constructible in time* $\mathrm{poly}(n)$*.*

While the above theorem restricts the block length to be of the form $(2^m - 1)$, for general n we can use as the parity check matrix any n columns of the parity check matrix of a BCH code of the minimum length $(2^m - 1)$ greater than or equal to n. In particular, we have the following corollary tailored for our purpose.

▶ **Corollary 12.** *For all lengths n and positive integers $k < n$, there exists a parity check matrix $\mathbf{R} \in \mathbb{F}_2^{20k\log n \times n}$ such that $\mathbf{Rx} \neq 0$ whenever $0 < \mathrm{wt}(\mathbf{x}) < 18k$. Moreover, this matrix can be computed in time* $\mathrm{poly}(n)$*.*

The following explicit family of ε-balanced binary linear codes of constant rate was given by Alon et al. [2].

▶ **Theorem 13** (ε-balanced codes [2])**.** *There exists an explicit family of codes $\mathcal{C} \subseteq \mathbb{F}_2^n$ such that every codeword in $\mathcal{C}$ has normalized weight in the range $[1/2 - \varepsilon, 1/2 + \varepsilon]$, and rate $\Omega(\varepsilon^3)$, which can be constructed in time $\mathrm{poly}(n, \frac{1}{\varepsilon})$, where $\varepsilon > 0$ is any arbitrarily small constant.*

Given a linear code $\mathcal{C} \subseteq \mathbb{F}_2^n$, the product code $\mathcal{C}^{\otimes 2}$ consists of $n \times n$ matrices where each row and each column belongs to $\mathcal{C}$; equivalently, $\mathcal{C}^{\otimes 2} = \{\mathbf{GXG}^{\mathrm{T}} : \mathbf{X} \in \mathbb{F}_2^{k\times k}\}$ where $\mathbf{G} \in \mathbb{F}_2^{n\times k}$ is the generator matrix for the code $\mathcal{C}$. If the distance $d(\mathcal{C}) = d$, then it is easy to verify that $d(\mathcal{C}^{\otimes 2}) \geqslant d^2$. However, we shall use the following lemma from [4] for a tighter lower bound on the Hamming weight when the code word satisfies certain properties.

▶ **Lemma 14** (Density of Product Codes [4])**.** *Let $\mathcal{C} \subseteq \mathbb{F}_2^n$ be a binary linear code of distance $d = d(\mathcal{C})$, and let $\mathbf{Y} \in \mathcal{C}^{\otimes 2}$ be a nonzero codeword with the additional properties that $\mathrm{diag}(\mathbf{Y}) = \mathbf{0}$, and $\mathbf{Y} = \mathbf{Y}^{\mathsf{T}}$. Then, the Hamming weight of $\mathbf{Y}$ is at least $\frac{3}{2}d^2$.*

2.3 Viola's Pseudorandom Generator

The proof of Theorem 9 in Section 4 uses Viola's [34] construction of pseudorandom generators which we describe below.

▶ **Definition 15.** A distribution $\mathcal{D}$ over $\mathbb{F}_2^n$ is said to *ε-fool* degree d polynomials in n-variables over $\mathbb{F}_2$ if for any degree d polynomial P:

$$\left| \mathop{\mathbf{E}}_{\mathbf{z}\leftarrow\mathcal{D}} [e(P(\mathbf{z}))] - \mathop{\mathbf{E}}_{\mathbf{z}\leftarrow\mathcal{U}} [e(P(\mathbf{z}))] \right| \leqslant \varepsilon,$$

where $\mathcal{U}$ is the uniform distribution over $\mathbb{F}_2^n$ and $e(x) := (-1)^x$ for $x \in \{0, 1\}$.

▶ **Theorem 16.** *Let $\mathcal{Y}_1, \ldots, \mathcal{Y}_d$ be d independent distributions on $\mathbb{F}_2^n$ that each ε-fool linear polynomials. Then the distribution $\mathcal{W} = \mathcal{Y}_1 + \cdots + \mathcal{Y}_d$ ε_d-fools degree-d polynomials where $\varepsilon_d := 16 \cdot \varepsilon^{1/2^{d-1}}$.*

3 Parameterized Reduction for the k-EvenSet problem

This section is devoted to proving the Theorem 3. The next few paragraphs give an informal description of the reduction. We then define the variables and equations of the k-EvenSet instance, and analyze the completeness and soundness of the reduction.

3.1 Reduction Overview

Let $\mathbf{Mx} = \mathbf{t}$ be a hard instance of k-VectorSum i.e., in the YES case there exists a k-sparse solution, whereas in the NO case all solutions have Hamming weight at least $(k+1)$. We homogenize this affine system by replacing the target vector $\mathbf{t}$ by $a_0\mathbf{t}$ for some $\mathbb{F}_2$-variable a_0, where $a_0\mathbf{t}$ is a coordinate-wise multiplication of $\mathbf{t}$ with the scalar a_0. Clearly, if all $(k+1)$-sparse (including a_0 as a variable) solutions to $\mathbf{Mx} = a_0\mathbf{t}$ have $a_0 = 1$ then the hardness of k-VectorSum implies the desired hardness result for k-EvenSet. However, this may not be true in general: there could exist a k-sparse $\mathbf{x}$ such that $\mathbf{Mx} = \mathbf{0}$. The objective of our reduction therefore, is to ensure that any solution to $\mathbf{Mx} = a_0\mathbf{t}$ that has $a_0 = 0$ with a k-sparse $\mathbf{x}$, must have significantly large weight in other auxiliary variables which we shall add in the construction.

Towards this end, we borrow some techniques from the proof of the inapproximability of MinimumDistance by Austrin and Khot [4]. Using transformations by suitable codes we first obtain a $K = O(k \log n)$-length *sketch* $\mathbf{y} = (y_1, \ldots, y_K)$ of $\mathbf{x}$, such that $\mathbf{y}$ is of normalized weight nearly $1/2$ when $\mathbf{x}$ is k-sparse but nonzero. We then construct a codeword $\mathbf{Y} \in \mathbb{F}_2^{K \times K}$, which is intended to be the product codeword $\mathbf{yy}^T$. However, this relationship cannot be expressed explicitly in terms of linear equations. Instead, for each pair of coordinates $(i,j) \in [K] \times [K]$, we introduce functions $Z_{ij} : \mathbb{F}_2 \times \mathbb{F}_2 \mapsto \mathbb{F}_2$ indicating the value taken by the pair (y_i, y_j) along with constraints that relate the Z_{ij} variables to codewords $\mathbf{y}$ and $\mathbf{Y}$. In fact, the explicit variables $\{Z_{ij}\}$ determine both $\mathbf{y}$ and $\mathbf{Y}$ which are implicit. The constraints also satisfy the key property: if $\mathbf{x}$ is k-sparse, then the number of nonzero Z_{ij} variables is significantly larger when $a_0 = 0$ than when $a_0 = 1$. This forces all sparse solutions to set $a_0 = 1$, which gives us the desired separation in sparsities between the YES and NO cases.

3.2 Constraints

Let $\mathbf{Mx} = \mathbf{t}$ be the instance of k-VectorSum over $\mathbb{F}_2$, in n variables and m equations. We homogenize this system by introducing a new $\mathbb{F}_2$-variable a_0 so that the new system of equations is then given by

$$\mathbf{Mx} = a_0\mathbf{t}, \tag{1}$$

where the $a_0\mathbf{t}$ is the coordinate wise product of $\mathbf{t}$ with the scalar a_0. We also add the following additional constraints and variables.

Linear Sketch Constraints: Let $\mathbf{R} \in \mathbb{F}^{k' \times n}$ be the parity check matrix of a $[n, n-k', 18k]$ linear code, where $k' = 20k \log n$, as defined in Corollary 12. Define $\boldsymbol{\eta}$ to be a k'-length sketch of $\mathbf{x}$ using $\mathbf{R}$ as,

$$\boldsymbol{\eta} = \mathbf{Rx}. \tag{2}$$

Mixing Constraints: Let $\mathbf{C} \in \mathbb{F}_2^{K \times k'}$ be the generator matrix of a linear code $\mathcal{C} \subseteq \mathbb{F}_2^K$ as defined in Theorem 13 where $\mathcal{C}$ has relative distance $\frac{1}{2} - \varepsilon$ and rate $\Omega(\varepsilon^3)$ for some small $\varepsilon > 0$ and $K = \frac{k'}{\Omega(\varepsilon^3)} \leqslant \frac{20k \log n}{c\varepsilon^3}$, for some constant $c > 0$. We add the constraint

$$\mathbf{y} = \mathbf{C}\boldsymbol{\eta} = \mathbf{C}\mathbf{R}\mathbf{x}. \tag{3}$$

Product Code Constraints: Let $\mathcal{C}^{\otimes 2} := \mathcal{C} \bigotimes \mathcal{C}$ be the product code with relative distance $\left(\frac{1}{2} - \varepsilon\right)^2$, constructed from $\mathcal{C}$. Let $\mathbf{Y} = \{Y_{ij}\}_{1 \leqslant i,j \leqslant K} \in \mathbb{F}_2^{K \times K}$ be such that $\mathbf{Y} = \mathbf{y}\mathbf{y}^\mathsf{T}$. To represent this relation linearly, we introduce variables $\{Z_{ij}(a,b)\}_{a,b \in \mathbb{F}_2}$ for each $1 \leqslant i,j \leqslant K$, which are intended to indicate the value assigned to the pair (y_i, y_j) i.e., $Z_{ij}(a,b) = \mathbb{1}\{y_i = a, y_j = b\}$. For each $(i,j) \in [K] \times [K]$ we add the following equations,

$$\begin{aligned} Z_{ij}(0,0) + Z_{ij}(0,1) + Z_{ij}(1,0) + Z_{ij}(1,1) &= a_0 & (4)\\ Z_{ij}(1,0) + Z_{ij}(1,1) &= y_i & (5)\\ Z_{ij}(0,1) + Z_{ij}(1,1) &= y_j & (6)\\ Z_{ij}(1,1) &= Y_{ij}. & (7) \end{aligned}$$

Furthermore, we add the constraints

$$\mathbf{Q}\mathbf{Y} = \mathbf{0}, \tag{8}$$

where $\mathbf{Q}$ is the parity check matrix for the product code $\mathcal{C}^{\otimes 2}$, and

$$\begin{aligned} Y_{ij} &= Y_{ji} \qquad \forall i \neq j, & (9)\\ Y_{ii} &= y_i \qquad \forall i \in [K], & (10) \end{aligned}$$

so that $\mathbf{Y}$ preserves the diagonal entries and symmetry of $\mathbf{y}\mathbf{y}^T$. Finally, we introduce $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^{r-1}$ and constraints

$$\mathbf{x}^i = \mathbf{x} \qquad \forall i \in [r-1], \tag{11}$$

where $r = \frac{K^2}{16k} \leqslant \frac{25k(\log n)^2}{c^2\varepsilon^6}$. These $r-1$ explicit copies of the vector $\mathbf{x}$ are used to balance the Hamming weight of the final solution. Observe that all the variables described above are linear combinations of $a_0, \{Z_{ij}(\cdot,\cdot)\}_{i,j \in [k]}$ and the coordinates of the vectors $\mathbf{x}$ and $\{\mathbf{x}^i\}_{i \in [r-1]}$. Hence, we analyze the sparsity of the solution restricted to these explicit variables. The total number of variables considered is $4K^2 + r \cdot n + 1$.

▶ Remark. The key difference between [4] and our reduction is in Equation (2) which constructs a small $(O(k \log n))$-length sketch of the n-length vector $\mathbf{x}$. This helps us contain the blowup in the sparsity of the solution to $O(k^2 \log^2 n)$ instead of $O(n)$.

3.3 Completeness

In the YES case, setting $a_0 = 1$ we obtain a k-sparse $\mathbf{x}$ such that $\mathbf{M}\mathbf{x} = a_0\mathbf{t} = \mathbf{t}$. Furthermore, for each $i,j \in [K]$, exactly one of the Z_{ij} variables would be nonzero. Hence, we have a solution of weight $K^2 + rk + 1$.

3.4 Soundness

Since the solution has to be non-trivial, at least one of $a_0, \mathbf{x}, \mathbf{y}, \mathbf{Y}$ must be nonzero. Note that when $\mathbf{x} = \mathbf{0}$, $\mathbf{y} = \mathbf{0}$ since $\mathbf{y}$ is a homogeneous linear transformation of $\mathbf{x}$. Moreover, we may assume that the weight of $\mathbf{x}$ is at most $\frac{K^2+1}{r} + k + 1 < 18k$ by our setting of r, otherwise the total weight of the solution would be at least $r \cdot \left(\frac{K^2+1}{r} + k + 1\right) \geqslant K^2 + r(k+1) + 1$ due to the copies of $\mathbf{x}$ and we would be done. The construction of $\mathbf{y}$ along with the upper bound of $18k$ on the weight of $\mathbf{x}$ constrains $\mathbf{y}$ to be nonzero when $\mathbf{x}$ is nonzero. Thus, the only three cases we need to consider are:

Case (i): $a_0 = 1$. In this case, any solution $\mathbf{x}$ to $\mathbf{Mx} = a_0\mathbf{t} = \mathbf{t}$ has weight at least $k+1$. Furthermore, for each $i, j \in [K]$, at least one of the four Z_{ij} variables must be nonzero since $a_0 = 1$. Hence, the total Hamming weight of the solution is at least $K^2 + r(k+1) + 1$.

Case (ii): $a_0 = 0, \mathbf{x} \neq \mathbf{0}, \mathbf{y} \neq \mathbf{0}$. By construction, since $\mathbf{y}$ is nonzero it has weight $\geqslant \left(\frac{1}{2} - \varepsilon\right) K$. Therefore, for at least $1 - \left(\frac{1}{2} + \varepsilon\right)^2 \geqslant \frac{3}{4} - 2\varepsilon$ fraction of the pairs $(i,j) \in [K] \times [K]$, either $y_i = 1$ or $y_j = 1$. Observe that for each such pair, at least two Z_{ij} variables are set to 1. Thus, the weight of any solution in this case is at least $2\left(\frac{3}{4} - 2\varepsilon\right)K^2 = \left(\frac{3}{2} - 4\varepsilon\right)K^2$.

Case (iii): $a_0 = 0, \mathbf{x} = \mathbf{0}, \mathbf{y} = \mathbf{0}, \mathbf{Y} \neq \mathbf{0}$. We have that $\text{diag}(\mathbf{Y}) = \mathbf{y} = \mathbf{0}$, $\mathbf{Y}$ is symmetric and it belongs to the product code $\mathcal{C}^{\otimes 2}$ (as enforced by Equations (8) and (9)). Then by Lemma 14, the weight of $\mathbf{Y}$ is at least $\left(\frac{3}{8} - 3\varepsilon\right) K^2$. Observe that for each $i, j \in [K]$ such that $Y_{ij} = 1$, Equations (4)-(7) force all four Z_{ij} variables to be set to 1. Hence, the number of nonzero Z_{ij}'s are at least $\left(\frac{3}{2} - 12\varepsilon\right) K^2$.

The above analysis yields that in contrast to the YES case which admits a $(K^2 + rk + 1)$-sparse solution, in the NO case all solutions are of weight at least

$$\min\left\{\left(K^2 + r(k+1) + 1\right), \left(\left(\frac{3}{2} - 12\varepsilon\right) K^2\right)\right\} \geqslant K^2 + r(k+1) + 1$$

by choice of the parameter r. Thus, solving the d-EvenSet problem with $d = K^2 + rk + 1 = O(k^2(\log n)^2)$ solves the k-VectorSum instance $\mathbf{Mx} = \mathbf{t}$.

4 Proof of Theorem 9

We first prove the following strengthening of Theorem 3.

▶ **Theorem 17** (Hardness of approximate k-EvenSet). *For any constant $\varepsilon > 0$, given an instance $(\mathbf{A}, \mathbf{b})$ of k'-VectorSum, where $\mathbf{A} \in \mathbb{F}_2^{m' \times n'}$ and $\mathbf{b} \in \mathbb{F}_2^{m'}$, there is an* FPT *reduction to an instance $\mathbf{B} \in \mathbb{F}_2^{m \times n}$ of k-EvenSet for some $k = O((k' \log n')^2)$, such that*

YES Case. *There is a nonzero k-sparse vector $\mathbf{x}$ which satisfies $\mathbf{Bx} = 0$.*

NO Case. *For any nonzero k-sparse vector $\mathbf{x}$ the weight of $\mathbf{Bx}$ is in the range $[1/2 - \varepsilon, 1/2 + \varepsilon]$.*

Here both m and n are bounded by fixed polynomials in m' and n'.

Proof. Let $\mathbf{M} \in \mathbb{F}_2^{r \times n}$ be the instance of k-EvenSet obtained by applying Theorem 3 to the instance $(\mathbf{A}, \mathbf{b})$ of k'-VectorSum we start with. Let $\mathbf{W} \in \mathbb{F}_2^{m \times r}$ be the generator matrix of an ε-balanced linear code given by Theorem 13, where $m = O(r/\varepsilon^3)$. Taking $\mathbf{B} := \mathbf{WM}$ completes the proof. ◀

It is easy to see that in the NO case the uniform distribution on the rows of the matrix $\mathbf{B}$ fools all linear forms (with error ε) over k variables.

Viola's result [34] (Theorem 16) implies that for any constant d, taking d-wise sums of the rows of $\mathbf{B}$ yields a distribution on which the YES case solution evaluates to 0, while in the NO case it fools all degree d polynomials supported on at most k variables with error $16 \cdot \varepsilon^{1/2^{d-1}}$. Taking ε to be a small enough constant completes the proof of Theorem 9.

5 Hardness of Learning k-Parities using k-Juntas

This section gives the proof of Theorem 7. Combining Theorem 6 with a small bias linear code we first induce an approximation gap for learning k-parities along with extending the result to non-homogeneous linear forms. In particular, let $\mathbf{W} = \{W_{ij}\} \in \mathbb{F}_2^{t\times m}$ be the generator matrix of an ε-balanced linear code given by Theorem 13, where $t = O(m/\varepsilon^3)$. Here, we choose $\varepsilon := \delta \cdot 2^{-k}$, where δ is as given in Theorem 7. Given an instance $\{(\mathbf{y}_j, a_j)\}_{j=1}^m$ from Theorem 6, let $\mathbf{z}_i = \sum_{j=1}^m W_{ij}\mathbf{y}_j$, and $b_i = \sum_{j=1}^m W_{ij}a_j$, for $i = 1,\ldots,t$. In the YES case, there is a homogeneous linear form L^* supported on at most k variables that satisfies all $\{(\mathbf{y}_j, a_j)\}_{j=1}^m$ and thus satisfies linear combinations of these point-value pairs, in particular $\{(\mathbf{z}_i, b_i)\}_{i=1}^t$.

For the NO case, we begin with the following lemma.

▶ **Lemma 18.** *If $\{(\mathbf{y}_j, a_j)\}_{j=1}^m$ is a* NO *instance, then any linear form $L(\mathbf{x}) + c$ supported on at most k variables satisfies a fraction in the range $[1/2 - \varepsilon, 1/2 + \varepsilon]$ of the point-value pairs $\{(\mathbf{z}_i, b_i)\}_{i=1}^t$.*

Proof. Since the homogeneous part L does not satisfy all pairs $\{(\mathbf{y}_j, a_j)\}_{j=1}^m$, it will satisfy a fraction in the range $[1/2 - \varepsilon, 1/2 + \varepsilon]$ of the pairs $\{(\mathbf{z}_i, b_i)\}_{i=1}^t$, due the lower and upper bounds bound on the weight of the nonzero codewords in the column space of $\mathbf{W}$. This also holds for $L + c$ for any constant c. ◀

We now extend the NO case to k-juntas. Let $f : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ be a function depending only a subset $S \subseteq [n]$ of coordinates where $|S| \leqslant k$. Define an extension $g : \mathbb{F}_2^{n+1} \mapsto \mathbb{F}_2$ as $g(x_1,\ldots,x_n,x_{n+1}) := f(x_1,\ldots,x_n) + x_{n+1}$. For convenience we shall abuse notation to denote $(\mathbf{z}, b) = (z_1,\ldots,z_n,b)$ where $\mathbf{z} = (z_1,\ldots,z_n) \in \mathbb{F}_2^n$ and $b \in \mathbb{F}_2$. To complete the proof we need to show that,

$$\left| \underset{(\mathbf{z},b)\in\mathcal{Z}}{\mathbf{E}} [e(g(\mathbf{z},b))] \right| \leqslant 2\delta, \tag{12}$$

where $e(x) := (-1)^x$. For some real values C_α ($\alpha \subseteq [n+1]$), the Fourier expansion of $e(g)$ is given by,

$$e(g) = \sum_{\alpha\subseteq[n+1]} C_\alpha \chi_\alpha.$$

Since $e(g(x_1,\ldots,x_{n+1})) = e(f(x_1,\ldots,x_n) + x_{n+1})$ and f depends only on coordinates in S, it is easy to see that the Fourier spectrum of $e(g)$ is supported only on characters χ_α such that $\alpha \subseteq S \cup \{n+1\}$. Further, since $e(g(x_1,\ldots,x_{n+1}))$ changes sign on flipping x_{n+1}, $C_\alpha \neq 0 \Rightarrow (n+1) \in \alpha$. Thus,

$$e(g) = \sum_{\substack{\alpha\subseteq S\cup\{n+1\}\\ (n+1)\in\alpha}} C_\alpha \chi_\alpha. \tag{13}$$

Observe that for any α in the sum above, $\chi_\alpha(x_1,\ldots,x_n,b) = e(L(x_1,\ldots,x_n) + b)$ where L is a homogeneous linear form supported on at most k variables. For any such α, Lemma 18 implies

$$\left| \underset{(\mathbf{z},b)\in\mathcal{Z}}{\mathbf{E}} [\chi_\alpha(\mathbf{z},b)] \right| \leqslant 2\varepsilon. \tag{14}$$

Using the above along with Equation (13) yields,

$$\left|\underset{(\mathbf{z},b)\in\mathcal{Z}}{\mathbf{E}}[e(g(\mathbf{z},b))]\right| \leqslant (2\varepsilon)\cdot\sum_{\substack{\alpha\subseteq S\cup\{n+1\}\\(n+1)\in\alpha}}|C_\alpha| \leqslant (2\varepsilon)\cdot 2^k = 2\delta,$$

where the last inequality is because there are at most 2^k subsets α in the sum on the RHS of Equation (13) and each $|C_\alpha|\leqslant 1$ since $e(g)$ is a $\{-1,1\}$-valued function.

6 W[1]-hardness of k-VectorSum on $O(k\log n)$ Equations

The following theorem implies Theorem 5.

▶ **Theorem 19.** *There is an* FPT *reduction from an instance $G(V,E)$ of k-*Clique*, over n vertices and m edges, to an instance $(\mathbf{M},\mathbf{b})$ of k'-*VectorSum*, where $\mathbf{M}\in\mathbb{F}_2^{d\times n'}$ such that $k'=\Theta(k^2)$, $d=O(k^2\log n)$ and $n'=O(nk+mk^2)$.*

The rest of this section is devoted to proving the above theorem. We start by observing that a k-clique in a graph $G(V,E)$ can be certified by the pair of mappings $f:[k]\mapsto V$ and $g:\binom{[k]}{2}\mapsto E$, such that $g(i,j)=(f(i),f(j))\in E \quad \forall i,j\in[k], i<j$. Here, we use $\binom{[k]}{2}$ to represent $\{(i,j)\mid 1\leqslant i<j\leqslant k\}$. The underlying idea behind the reduction is to construct $\mathbf{M}$ and $\mathbf{b}$ such that f and g exist *iff* there is a sparse set of columns of $\mathbf{M}$ that sums up to $\mathbf{b}$.

Construction of M and b. Let $G(V,E)$ be a k-Clique instance on $n=|V|$ vertices and $m=|E|$ edges, where $V=\{v_1,v_2,\ldots,v_n\}$. For each vertex $v_i\in V$, assign a distinct $N=\lceil\log(n+1)\rceil$ bit nonzero binary pattern denoted by $\mathbf{q}_i\in\mathbb{F}_2^N$. We first construct a set of vectors – which shall be the columns of $\mathbf{M}$ – corresponding to the vertices and edges. The dimension over which the vectors are defined is partitioned into three sets of coordinates:

Edge-Vertex Incidence Coordinates: These consist of k slots, where each slot consists of $(k-1)$ subslots, and each subslot in turn consists of N coordinates. In any column of $\mathbf{M}$, a subslot may either contain the N-length pattern of a vertex, or it might be all zeros.

Edge Indicator Coordinates: These are a set of $\binom{k}{2}$ coordinates corresponding to $\{(i,j)\mid 1\leqslant i<j\leqslant k\}$, indicating whether the vector represents an edge mapped from (i,j). Any column of $\mathbf{M}$ may have at most one of these coordinates set to 1.

Vertex Indicator Coordinates: These are a set of k coordinates corresponding to indices $i\in\{1,\ldots,k\}$, which indicate whether the vector represents a vertex mapped from i. Any column of $\mathbf{M}$ may have at most one of these coordinates set to 1.

Thus, each vector is a concatenation of $k(k-1)N$ edge-vertex incidence bits, followed by $\binom{k}{2}$ edge indicator bits and k vertex indicator bits, so that $d=k(k-1)N+\binom{k}{2}+k=O(k^2\log n)$. For ease of notation, let S_l^j represent the N-sized subset of coordinates belonging to the subslot l of slot j where $j\in[k]$ and $l\in[k-1]$. We define $\mathbf{q}_i(S_l^j)\in\mathbb{F}_2^d$ to be the vector which contains the pattern of vertex v_i in S_l^j, and is zero everywhere else. For $1\leqslant i<j\leqslant k$, let $\boldsymbol{\delta}_{i,j}\in\mathbb{F}_2^d$ be the vector which has a 1 at the edge indicator coordinate corresponding to (i,j), and is 0 everywhere else. Similarly, $\boldsymbol{\delta}_i\in\mathbb{F}_2^d$ is the indicator vector which has its ith vertex indicator coordinate set to 1, everything else being 0. Using these components we construct the vertex and edge vectors as follows.

Vertex Vectors: For each vertex $v_i \in V$ and $j \in [k]$, we introduce a vector $\boldsymbol{\eta}(v_i, j) \in \mathbb{F}_2^d$ which indicates that vertex v_i is mapped from index (slot) j i.e., $f(j) = v_i$. The vector is constructed as follows: populate each of the $(k-1)$ subslots of the jth slot with the pattern $\mathbf{q}_i$ of vertex v_i, and set its jth vertex indicator coordinate to 1. Formally, $\boldsymbol{\eta}(v_i, j) := \sum_{l=1}^{k-1} \mathbf{q}_i(S_l^j) + \boldsymbol{\delta}_j$. For each vertex there are k vertex vectors resulting in a total of nk vertex vectors.

Edge Vectors: For each edge $e = (v_{i_1}, v_{i_2}) \in E$ where $i_1 < i_2$, and $1 \leqslant j_1 < j_2 \leqslant k$, we introduce a vector that indicates that the pair of indices (slots) (j_1, j_2) is mapped to (v_{i_1}, v_{i_2}) i.e., $g(j_1, j_2) = (v_{i_1}, v_{i_2})$. We construct the vector $\boldsymbol{\eta}(e, j_1, j_2) \in \mathbb{F}_2^d$ as follows: populate $S_{j_2-1}^{j_1}$ with the pattern of vertex v_{i_1}, and $S_{j_1}^{j_2}$ with the pattern of vertex v_{i_2}. Additionally, we set the edge indicator coordinate corresponding to (j_1, j_2) to 1. The vector is formally expressed as, $\boldsymbol{\eta}(e, j_1, j_2) := \mathbf{q}_{i_1}(S_{j_2-1}^{j_1}) + \mathbf{q}_{i_2}(S_{j_1}^{j_2}) + \boldsymbol{\delta}_{j_1,j_2}$. Intuitively, for the lower ordered vertex v_{i_1}, $\boldsymbol{\eta}(e, j_1, j_2)$ cancels out the (j_2-1)th subslot of slot j_1, and for the higher ordered vertex v_{i_2}, it cancels out the j_1th subslot of its j_2th slot. Note that we are treating (v_{i_1}, v_{i_2}) as an *unordered* pair since $i_1 < i_2$. Therefore, for each edge $e \in E$, and for each choice of $1 \leqslant j_1 < j_2 \leqslant k$, we introduce one edge vector. Hence, there are a total of $m \cdot \binom{k}{2}$ edge vectors in the set.

The vertex and edge vectors constructed above constitute the columns of $\mathbf{M}$. The target vector $\mathbf{b}$ ensures that (i) every solution must have at least k vertex vectors, and $\binom{k}{2}$ edge vectors and (ii) the vectors must cancel each other out in the Edge-Vertex Incidence coordinates. Formally, $\mathbf{b} = \sum_{i \in [k]} \boldsymbol{\delta}_i + \sum_{1 \leqslant i < j < k} \boldsymbol{\delta}_{i,j}$. In other words, all the edge and vertex indicator coordinates of $\mathbf{b}$ are set to 1, and everything else to 0.

6.1 YES case

We show that if $G(V, E)$ has a k-Clique, then there exists a set of $k + \binom{k}{2}$ columns of $\mathbf{M}$ that sum to $\mathbf{b}$. Assume that $v_{i_1}, v_{i_2}, \ldots, v_{i_k}$ form a k-clique where $i_1 < i_2 < \cdots < i_k$. We select k vertex vectors $\{\boldsymbol{\eta}(v_{i_j}, j)\}_{j \in [k]}$, and $\binom{k}{2}$ edge vectors $\{\boldsymbol{\eta}(e, j_1, j_2) \mid e = (v_{i_{j_1}}, v_{i_{j_2}}), 1 \leqslant j_1 < j_2 \leqslant k\}$. Since the k vertices form a clique, these vectors always exists. Observe that for any fixed $j \in [k]$, (i) for $\ell = 1, \ldots, j-1$, $\boldsymbol{\eta}(v_{i_j}, j)$ and $\boldsymbol{\eta}(e, \ell, j)$ have the same pattern $\mathbf{q}_{i_j}$ in subslot ℓ of slot j, where $e = (v_{i_\ell}, v_{i_j})$, and (ii) for $\ell = j+1, \ldots, k$, $\boldsymbol{\eta}(v_{i_j}, j)$ and $\boldsymbol{\eta}(e, j, \ell)$ have the same pattern $\mathbf{q}_{i_j}$ in subslot $(\ell - 1)$ of slot j, where $e = (v_{i_j}, v_{i_\ell})$. Thus, the $k + \binom{k}{2}$ selected vectors sum to zero on all but the vertex and edge indicator coordinates and thus sum up to $\mathbf{b}$.

6.2 NO Case

Suppose for a contradiction that $\mathcal{S}$ is a subset of columns of $\mathbf{M}$ that sum to $\mathbf{b}$ and that $|\mathcal{S}| \leqslant k + \binom{k}{2}$.

► **Proposition 20.** *There are exactly k vertex vectors corresponding to indices (slots) $i \in [k]$ in $\mathcal{S}$. Also, there are exactly $\binom{k}{2}$ edge vectors, one for each pair (i, j) $(1 \leqslant i < j \leqslant k)$ of slots, in $\mathcal{S}$.*

Proof. This follows from the observation that there are $k + \binom{k}{2}$ nonzero indicator coordinates in the target $\mathbf{b}$, and each (edge or vertex) vector contributes exactly one nonzero (edge or vertex) indicator coordinate. Therefore, by a counting argument, k vertex vectors, one each for the indices (slots) $i \in [k]$, must contribute to the k vertex indicator bits. Similarly, $\binom{k}{2}$

edge vectors, one for each pair of slots (i, j) $(1 \leqslant i < j \leqslant k)$, must contribute to the $\binom{k}{2}$ edge indicator bits. ◀

The above proposition implies that for each pair of vertex vectors there is exactly one edge vector which has a common populated subslot with each of them. So there are exactly $(k-1)$ edge vectors which share a populated subslot with any given vertex vector in $\mathcal{S}$.

Since the k vertex vectors in $\mathcal{S}$ populate distinct slots, in total $k(k-1)$ subslots are populated by the sum of the k vertex vectors. Note that any edge vector populates exactly 2 subslots. Thus, for the $\binom{k}{2} = k(k-1)/2$ edge vectors in $\mathcal{S}$ to sum up to the values in $k(k-1)$ subslots, it must be that the edge vectors populate distinct subslots. In other words, no two edge vectors are both nonzero in the same slot-subslot combination.

Thus, for each vertex vector there are exactly $(k-1)$ edge vectors which share distinct populated subslots with it, and these edge vectors must cancel out the corresponding subslots i.e., have the same pattern in the shared subslot as that of the vertex vector. In other words, for any two vertex vectors corresponding to slots i and j respectively $(i < j)$, the edge vector corresponding to the pair (i, j) must cancel one subslot from each one of the two vertex vectors. This is possible only if (i) the k vertex vectors correspond to distinct vertices in G, and (ii) each pair of these vertices have an edge between them for the corresponding edge vector to exist. This implies that G has a k-clique which is a contradiction.

7 A simple $O(n \cdot 2^m)$ -time algorithm for k-VectorSum

Let $(\mathbf{M}, \mathbf{b})$ be an instance of k-VectorSum where $\mathbf{M} \in \mathbb{F}_2^{m \times n}$ and $\mathbf{b} \in \mathbb{F}_2^m$. Construct a graph G on vertex set $V = \mathbb{F}_2^m$ and edge set given by,

$$E = \left\{ \{\mathbf{u}, \mathbf{v}\} \in \binom{V}{2} \mid \mathbf{u} + \mathbf{v} \text{ is a column of } \mathbf{M} \right\}.$$

We say that an edge $\{\mathbf{u}, \mathbf{v}\} \in E$ is labeled by the column $\mathbf{u} + \mathbf{v}$ of $\mathbf{M}$. Clearly, if there is a vector $\mathbf{x}$ of Hamming weight at most k such that $\mathbf{M}\mathbf{x} = \mathbf{b}$ then there is a path of length at most k in G from $\mathbf{0}$ to $\mathbf{b}$ given by choosing the edges labeled by the columns corresponding to the non-zero entries of $\mathbf{x}$ in any sequence. On the other hand, if there is a path in G from $\mathbf{0}$ to $\mathbf{b}$ of length at most k, then there is a sequence of at most k columns (with possible repetitions) of $\mathbf{M}$ which sum up to $\mathbf{b}$. Cancelling out even number of repetitions of any column yields a subset of at most k distinct columns of $\mathbf{M}$ that sum up to $\mathbf{b}$. Thus, deciding k-VectorSum reduces to determining whether there is a path of length at most k from $\mathbf{0}$ to $\mathbf{b}$.

The size of V is 2^r and of E is at most $n \cdot 2^m$, and the graph can be constructed in time $O(n \cdot 2^m)$. Doing a Breadth First Search yields a running time of $O(n \cdot 2^m)$.

Acknowledgements. We thank Subhash Khot for his permission to include (in the full version of this paper [5]) his proof for the hardness of learning parities using polynomials. Also, thanks to Ryan Williams for encouraging and stimulating conversations and to Rocco Servedio for bringing [3] to our attention.

References

1 Amir Abboud, Kevin Lewi, and Ryan Williams. Losing weight by gaining edges. In *Proc. 22nd Annual European Symposium on Algorithms*, pages 1–12. Springer, 2014.

2 Noga Alon, Jehoshua Bruck, Joseph Naor, Moni Naor, and Ron M Roth. Construction of asymptotically good low-rate error-correcting codes through pseudo-random graphs. *IEEE Trans. Inform. Theory*, 38(2):509–516, 1992.

3 Vikraman Arvind, Johannes Köbler, and Wolfgang Lindner. Parameterized learnability of juntas. *Theor. Comp. Sci.*, 410(47-49):4928–4936, 2009.

4 Per Austrin and Subhash Khot. A simple deterministic reduction for the gap minimum distance of code problem. *IEEE Trans. Inform. Theory*, 60(10):6636–6645, 2014.

5 Arnab Bhattacharyya, Ameet Gadekar, Suprovat Ghoshal, and Rishi Saket. On the hardness of learning sparse parities. *CoRR*, abs/1511.08270, 2015. URL: `http://arxiv.org/abs/1511.08270`.

6 Arnab Bhattacharyya, Piotr Indyk, David P Woodruff, and Ning Xie. The complexity of linear dependence problems in vector spaces. In *Proc. 2nd Innovations in Computer Science*, pages 496–508, 2011.

7 Avrim Blum. On-line algorithms in machine learning. In *Workshop on on-line algorithms, Dagstuhl*, pages 305–325. Springer, 1996.

8 Edouard Bonnet, Bruno Escoffier, Eun Jung Kim, and Vangelis Th. Paschos. On subexponential and fpt-time inapproximability. *Algorithmica*, 71(3):541–565, 2015.

9 R. C. Bose and Dwijendra K. Ray-Chaudhuri. On A class of error correcting binary group codes. *Information and Control*, 3(1):68–79, 1960.

10 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. A duality between clause width and clause density for SAT. In *21st Annual IEEE Conference on Computational Complexity*, pages 252–260, 2006.

11 Qi Cheng and Daqing Wan. Complexity of decoding positive-rate reed-solomon codes. In *Proc. 35th Annual International Conference on Automata, Languages, and Programming*, pages 283–293. Springer, 2008.

12 Qi Cheng and Daqing Wan. A deterministic reduction for the gap minimum distance problem. In *Proc. 41st Annual ACM Symposium on the Theory of Computing*, pages 33–38. ACM, 2009.

13 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer International Publishing, 2015.

14 Irit Dinur. The PCP theorem by gap amplification. *J. ACM*, 54(3), 2007.

15 Rod G Downey, Michael R Fellows, Alexander Vardy, and Geoff Whittle. The parametrized complexity of some fundamental problems in coding theory. *SIAM J. on Comput.*, 29(2):545–570, 1999.

16 Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness I: basic results. *SIAM J. on Comput.*, 24(4):873–921, 1995.

17 Rodney G Downey and Michael Ralph Fellows. *Parameterized complexity*. Springer Science & Business Media, 1999.

18 Ilya Dumer, Daniele Micciancio, and Madhu Sudan. Hardness of approximating the minimum distance of a linear code. *IEEE Trans. Inform. Theory*, 49(1):22–37, 2003.

19 Michael R Fellows, Jiong Guo, Dániel Marx, and Saket Saurabh. Data reduction and problem kernels (Dagstuhl Seminar 12241). *Dagstuhl Reports*, 2(6):26–50, 2012.

20 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer Verlag, 2006.

21 Fedor V Fomin and Dániel Marx. FPT suspects and tough customers: Open problems of Downey and Fellows. In *The Multivariate Algorithmic Revolution and Beyond*, pages 457–468. Springer, 2012.

22 Parikshit Gopalan, Subhash Khot, and Rishi Saket. Hardness of reconstructing multivariate polynomials over finite fields. *SIAM J. on Comput.*, 39(6):2598–2621, 2010.

23 Johan Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001.

24 Russell Impagliazzo and Ramamohan Paturi. Complexity of k-SAT. In *Proc. 14th Annual IEEE Conference on Computational Complexity*, pages 237–240. IEEE, 1999.

25 Adam Tauman Kalai, Yishay Mansour, and Elad Verbin. On agnostic boosting and parity learning. In *Proc. 40th Annual ACM Symposium on the Theory of Computing*, pages 629–638. ACM, 2008.

26 Subhash Khot. personal communication, 2009.

27 Subhash Khot and Igor Shinkar. On hardness of approximating the parameterized clique problem. In *Proc. 7th Innovations in Theoretical Computer Science*, pages 37–45. ACM, 2016.

28 Adam R Klivans and Rocco A Servedio. Toward attribute efficient learning of decision lists and parities. *J. Mach. Learn. Res.*, 7:587–602, 2006.

29 Dániel Marx. Parameterized complexity and approximation algorithms. *The Computer Journal*, 51(1):60–78, 2008.

30 Daniele Micciancio. Locally dense codes. In *Proc. 29th Annual IEEE Conference on Computational Complexity*, pages 90–97. IEEE, 2014.

31 Elchanan Mossel, Ryan O'Donnell, and Rocco A. Servedio. Learning functions of k relevant variables. *J. Comp. Sys. Sci.*, 69(3):421–434, November 2004.

32 Gregory Valiant. Finding correlations in subquadratic time, with applications to learning parities and juntas. In *Proc. 53rd Annual IEEE Symposium on Foundations of Computer Science*, pages 11–20. IEEE, 2012.

33 Alexander Vardy. The intractability of computing the minimum distance of a code. *IEEE Trans. Inform. Theory*, 43(6):1757–1766, 1997.

34 Emanuele Viola. The sum of D small-bias generators fools polynomials of degree D. *Computational Complexity*, 18(2):209–217, 2009.

Online Algorithms for Multi-Level Aggregation[*]

Marcin Bienkowski[1], Martin Böhm[2], Jaroslaw Byrka[3], Marek Chrobak[4], Christoph Dürr[5], Lukáš Folwarczný[6], Łukasz Jeż[7], Jiří Sgall[8], Nguyen Kim Thang[9], and Pavel Veselý[10]

1 Institute of Computer Science, University of Wrocław, Wrocław, Poland
2 Computer Science Institute, Charles University, Prague, Czech Republic
3 Institute of Computer Science, University of Wrocław, Wrocław, Poland
4 Department of Computer Science, University of California, Riverside, USA
5 Sorbonne Universités, UPMC Univ Paris 06, CNRS, LIP6, Paris, France
6 Computer Science Institute, Charles University, Prague, Czech Republic
7 Institute of Computer Science, University of Wrocław, Wrocław, Poland
8 Computer Science Institute, Charles University, Prague, Czech Republic
9 IBISC, Université d'Evry Val d'Essonne, Evry, France
10 Computer Science Institute, Charles University, Prague, Czech Republic

Abstract

In the *Multi-Level Aggregation Problem* (MLAP), requests arrive at the nodes of an edge-weighted tree $\mathcal{T}$, and have to be served eventually. A *service* is defined as a subtree X of $\mathcal{T}$ that contains its root. This subtree X serves all requests that are pending in the nodes of X, and the cost of this service is equal to the total weight of X. Each request also incurs waiting cost between its arrival and service times. The objective is to minimize the total waiting cost of all requests plus the total cost of all service subtrees. MLAP is a generalization of some well-studied optimization problems; for example, for trees of depth 1, MLAP is equivalent to the TCP Acknowledgment Problem, while for trees of depth 2, it is equivalent to the Joint Replenishment Problem. Aggregation problem for trees of arbitrary depth arise in multicasting, sensor networks, communication in organization hierarchies, and in supply-chain management. The instances of MLAP associated with these applications are naturally online, in the sense that aggregation decisions need to be made without information about future requests.

Constant-competitive online algorithms are known for MLAP with one or two levels. However, it has been open whether there exist constant competitive online algorithms for trees of depth more than 2. Addressing this open problem, we give the first constant competitive online algorithm for networks of arbitrary (fixed) number of levels. The competitive ratio is $O(D^4 2^D)$, where D is the depth of $\mathcal{T}$. The algorithm works for arbitrary waiting cost functions, including the variant with deadlines. We include several additional results in the paper. We show that a standard lower-bound technique for MLAP, based on so-called *Single-Phase* instances, cannot give super-constant lower bounds (as a function of the tree depth). This result is established by giving an online algorithm with optimal competitive ratio 4 for such instances on arbitrary trees. We also study the MLAP variant when the tree is a path, for which we give a lower bound of 4 on the competitive ratio, improving the lower bound known for general MLAP. We complement this with a matching upper bound for the deadline setting.

1998 ACM Subject Classification F.1.2 Modes of Computation – Online computation, F.2.2 Nonnumerical Algorithms and Problems – Sequencing and scheduling

[*] Research partially supported by NSF grants CCF-1536026, CCF-1217314 and OISE-1157129, Polish NCN grants DEC-2013/09/B/ST6/01538, 2015/18/E/ST6/00456, project 14-10003S of GA ČR and GAUK project 548214.

Keywords and phrases algorithmic aspects of networks, online algorithms, scheduling and resource allocation

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.12

1 Introduction

Certain optimization problems can be formulated as aggregation problems. They typically arise when expensive resources can be shared by multiple agents, who incur additional expenses for accessing a resource. For example, costs may be associated with waiting until the resource is accessible, or, if the resource is not in the desired state, a costly setup or retooling may be required.

1-level aggregation

A simple example of an aggregation problem is the *TCP Acknowledgment Problem (*TCP-AP*)*, where control messages ("agents") waiting for transmission across a network link can be aggregated and transmitted in a single packet ("resource"). Such aggregation can reduce network traffic, but it also results in undesirable delays. A reasonable compromise is to balance the two costs, namely the number of transmitted packets and the total delay, by minimizing their weighted sum [15]. Interestingly, TCP-AP is equivalent to the classical Lot Sizing Problem studied in the operations research literature since the 1950s. (See, for example, [30].) In the offline variant of TCP-AP, that is when all arrival times of control messages are known beforehand, an optimal schedule for aggregated packets can be computed with dynamic programming in time $O(n \log n)$ [1]. In practice, however, packet aggregation decisions must be done on the fly, without any information about future message releases. This scenario is captured by the online variant of TCP-AP that has also been well studied; it is known that the optimal competitive ratio is 2 in the deterministic case [15] and $\mathrm{e}/(\mathrm{e}-1) \approx 1.582$ in the randomized case [17, 11, 28].

2-level aggregation

Another optimization problem involving aggregation is the *Joint Replenishment Problem (JRP)*, well-studied in operations research. JRP models tradeoffs that arise in supply-chain management. One such scenario involves optimizing shipments of goods from a supplier to retailers, through a shared warehouse, in response to their demands. In JRP, aggregation takes place at two levels: items addressed to different retailers can be shipped together to the warehouse, at a fixed cost, and then multiple items destined to the same retailer can be shipped from the warehouse to this retailer together, also at a fixed cost, which can be different for different retailers. Pending demands accrue waiting cost until they are satisfied by a shipment. The objective is to minimize the sum of all shipment costs and all waiting costs.

JRP is known to be $\mathbb{NP}$-hard [2], and even $\mathbb{APX}$-hard [25, 5]. The currently best approximation, due to Bienkowski et al. [6], achieves a factor of 1.791, improving on earlier work by Levi et al. [21, 23, 24]. In the deadline variant of JRP, denoted JRP-D, there is no cost for waiting, but each demand needs to be satisfied before its deadline. As shown in [5], JRP-D can be approximated with ratio 1.574.

For the online variant of JRP, Buchbinder et al. [10] gave a 3-competitive algorithm using a primal-dual scheme (improving an earlier bound of 5 in [9]) and proved a lower bound of

2.64, that was subsequently improved to 2.754 [6]. The optimal competitive ratio for JRP-D is 2 [6].

Multiple-level aggregation

TCP-AP and JRP can be thought of as aggregation problems on edge-weighted trees of depth 1 and 2, respectively. In TCP-AP, this tree is just a single edge between the sender and the recipient. In JRP, this tree consists of the root (supplier), with one child (warehouse), and any number of grandchildren (retailers). A shipment can be represented by a subtree of this tree and edge weights represent shipping costs. These trees capture the general problem on trees of depth 1 and 2, as the children of the root can be considered separately (see Section 2).

This naturally extends to trees of any depth D, where aggregation is allowed at each level. Multi-level message aggregation has been, in fact, studied in communication networks in several contexts. In multicasting, protocols for aggregating control messages (see [8, 3], for example) can be used to reduce the so-called *ack-implosion*, the proliferation of control messages routed to the source. A similar problem arises in energy-efficient data aggregation and fusion in sensor networks [16, 31]. Outside of networking, tradeoffs between the cost of communication and delay arise in message aggregation in organizational hierarchies [26]. In supply-chain management, multi-level variants of lot sizing have been studied [14, 19]. The need to consider more tree-like (in a broad sense) supply hierarchies has also been advocated in [20].

These applications have inspired research on offline and online approximation algorithms for multi-level aggregation problems. Becchetti et al. [4] gave a 2-approximation algorithm for the deadline case. (See also [9].) Pedrosa [27] showed, adapting an algorithm of Levi et al. [22] for the multi-stage assembly problem, that there is a $(2 + \varepsilon)$-approximation algorithm for general waiting cost functions, where ε can be made arbitrarily small.

In the online case, Khanna et al. [18] gave a rent-or-buy solution (that serves a group of requests once their waiting cost reaches the cost of their service) and showed that their algorithm is $O(\log \alpha)$-competitive, where α is defined as the sum of all edge weights. However, they assumed that each request has to wait at least one time unit. This assumption is crucial for their proof, as demonstrated by Brito et al. [9], who showed that the competitive ratio of a rent-or-buy strategy is $\Omega(D)$, even for paths with D edges. The same assumption of a minimal cost for a request and a ratio dependent on the edge-weights is also essential for Vaya [29], who studies a variant of the problem with bounded bandwidth (the number of packets that can be served by a single edge in a single service).

The existence of a primal-dual $(2 + \varepsilon)$-approximation algorithm [27, 22] for the offline problem suggests the possibility of constructing an online algorithm along the lines of [11]. Nevertheless, despite substantial effort of many researchers, the online multi-level setting remains wide open. This is perhaps partly due to impossibility of direct emulation of the cleanup phase in the primal-dual offline algorithms in the online setting, as this cleanup is performed in the "reverse time" order.

The case when the tree is just a path has also been studied. An offline polynomial-time algorithm that computes an optimal schedule was given in [7]. For the online variant, Brito et al. [9] gave an 8-competitive algorithm. This result was improved by Bienkowski et al. [7] who showed that the competitive ratio of this problem is between $2 + \phi \approx 3.618$ and 5.

Table 1 Previous and current bounds on the competitive ratios for MLAP for trees of various depths. Ratios written in bold are shown in this paper. Unreferenced results are either immediate consequences of other entries in the table or trivial observations. Asterisked ratios represent results for MLAP with arbitrary waiting cost functions, which, though not explicitly stated in the respective papers, are straightforward extensions of the corresponding results for MLAP-L.

	MLAP and MLAP-L		MLAP-D	
	upper	lower	upper	lower
depth 1	2^* [15]	2 [15]	1	1
rand. alg. for depth 1	1.582^* [17]	1.582 [28]	1	1
depth 2	3 [10]	2.754 [6]	2 [6]	2 [6]
fixed depth $D \geq 2$	$\mathbf{O(D^4 2^D)}$	2.754	$\mathbf{D^2 2^D}$	2
paths of arbitrary depth	5^* [7]	3.618 [7], **4**	**4**	**4**

1.1 Our Contributions

We study online competitive algorithms for multi-level aggregation. Minor technical differences notwithstanding, our model is equivalent to those studied in [9, 18], also extending the deadline variant in [4] and the assembly problem in [22]. We have decided to choose a more generic terminology to emphasize general applicability of our model and techniques.

Formally, our model consists of a tree $\mathcal{T}$ with positive weights assigned to edges, and a set of requests $\mathcal{R}$ that arrive in the nodes of $\mathcal{T}$ over time. These requests are served by subtrees rooted at the root of $\mathcal{T}$. Such a subtree X serves all requests pending at the nodes of X at cost equal to the total weight of X. Each request incurs a waiting cost, defined by a non-negative and non-decreasing function of time, which may be different for each request. The objective is to minimize the sum of the total service and waiting costs. We call this the *Multi-Level Aggregation Problem* (MLAP).

In most earlier papers on aggregation problems, the waiting cost function is linear, that is, it is assumed to be simply the delay between the times when a request arrives and when it is served. We denote this version by MLAP-L. However, most of the algorithms for this model extend naturally to arbitrary cost functions. Another variant is MLAP-D, where each request is given a certain deadline, has to be served before or at its deadline, and there is no penalty associated with waiting. This can be modeled by the waiting cost function that is 0 up to the deadline and $+\infty$ afterwards.

In this paper, we mostly focus on the online version of MLAP, where an algorithm needs to produce a schedule in response to requests that arrive over time. When a request appears, its waiting cost function is also revealed. At each time t, the online algorithm needs to decide whether to generate a service tree at this time, and if so, which nodes should be included in this tree.

The main result of our paper is an $O(D^4 2^D)$-competitive algorithm for MLAP for trees of depth D, presented in Section 4. A simpler $D^2 2^D$-competitive algorithm for MLAP-D is presented in Section 3. No competitive algorithms have been known so far for online MLAP for arbitrary depth trees, even for the special case of MLAP-D on trees of depth 3.

For both results we use a reduction of the general problem to the special case of trees with fast decreasing weights. For such trees we then provide an explicit competitive algorithm. While our algorithm is compact and elegant, it is not a straightforward extension of the 2-level algorithm. (In fact, we have been able to show that naïve extensions of the latter algorithm are not competitive.) It is based on carefully constructing a sufficiently large service tree whenever it appears that an urgent request must be served. The specific structure of the service tree is then heavily exploited in an amortization argument that constructs a mapping from the algorithm's cost to the cost of the optimal schedule. We believe that these three new techniques: the reduction to trees with fast decreasing weights, the construction of the service trees, and our charging scheme, will be useful in further studies of online aggregation problems.

In Section 5 we study a version of MLAP, that we refer to as *Single-Phase* MLAP (or 1P-MLAP), in which all requests arrive at the beginning, but they also have a common *expiration time* that we denote by θ. Any request not served by time θ pays waiting cost at time θ and does not need to be served anymore. In spite of the expiration-date feature, it can be shown that 1P-MLAP can be represented as a special case of MLAP. 1P-MLAP is a crucial tool in all the lower bound proofs in the literature for competitive ratios of MLAP, including those in [10, 7], as well as in our lower bounds in Section 6. It also has a natural interpretation in the context of JRP (2-level MLAP), if we allow all orders to be canceled, say, due to changed market circumstances. In the online variant of 1P-MLAP all requests are known at the beginning, but the expiration time θ is unknown. For this version we give an online algorithm with competitive ratio 4, matching the lower bound. Since 1P-MLAP can be expressed as a special case of MLAP, our result implies that the techniques from [10, 7] cannot be used to prove a lower bound on the competitive ratio for MLAP larger than 4, and any study of the dependence of the competitive ratio on the depth D will require new insights and techniques.

In Section 6 we consider MLAP on paths. For this case, we give a 4-competitive algorithm for MLAP-D and we provide a matching lower bound. We show that the same lower bound of 4 applies to MLAP-L as well, improving the previous lower bound of 3.618 from [7].

Due to the page limit, most of the proofs will be given in the full version of the paper.

2 Preliminaries

Let $\mathcal{T}$ be a tree with root r. For any set of nodes $Z \subseteq \mathcal{T}$ and a node x, Z_x denotes the set of all descendants of x in Z; in particular, $\mathcal{T}_x$ is the subtree of $\mathcal{T}$ rooted at x. The parent of a node x is denoted $\mathit{parent}(x)$. The *depth of* x, denoted $\mathit{depth}(x)$, is the number of edges on the simple path from r to x. In particular, r is at depth 0. The depth D of $\mathcal{T}$ is the maximum depth of a node of $\mathcal{T}$.

We will deal with weighted trees in this paper. For $x \neq r$, by ℓ_x or $\ell(x)$ we denote the weight of the edge connecting node x to its parent. We assume that all these weights are positive. We extend this notation to r by setting $\ell_r = 0$. If Z is any set of nodes of $\mathcal{T}$, then the weight of Z is $\ell(Z) = \sum_{x \in Z} \ell_x$.

Definition of MLAP

A *request* ρ is specified by a triple $\rho = (\sigma_\rho, a_\rho, \omega_\rho)$, where σ_ρ is the node of $\mathcal{T}$ in which ρ is issued, a_ρ is the *arrival time* of ρ, and ω_ρ is the waiting cost function of ρ. We assume that $\omega_\rho(t) = 0$ for $t \leq a_\rho$ and $\omega_\rho(t)$ is non-decreasing for $t \geq a_\rho$. MLAP-L is the variant of MLAP with linear waiting costs; that is, for each request ρ we have $\omega_\rho(t) = t - a_\rho$, for $t \geq a_\rho$. In MLAP-D, the variant with deadlines, we have $\omega_\rho(t) = 0$ for $t \leq d_\rho$ and $\omega_\rho(t) = \infty$ for $t > d_\rho$, where d_ρ is called the *deadline* of request ρ.

In our algorithms for MLAP with general costs we will be assuming that all waiting cost functions are continuous. This is only for technical convenience and we discuss more general waiting cost functions in the full version of the paper; we also show there that MLAP-D can be considered a special case of MLAP, and that our algorithms can be extended to the discrete-time model.

A *service* is a pair (X, t), where X is a subtree of $\mathcal{T}$ rooted at r and t is the time of this service. We will occasionally refer to X as the service tree (or just service) at time t, or even omit t altogether if it is understood from context.

An instance $\mathcal{J} = \langle \mathcal{T}, \mathcal{R} \rangle$ of the *Multi-Level Aggregation Problem* (MLAP) consists of a weighted tree $\mathcal{T}$ with root r and a set $\mathcal{R}$ of requests arriving at the nodes of $\mathcal{T}$. A *schedule* is a set S of services. For a request ρ, let (X, t) be the service in S with minimal t such that $\sigma_\rho \in X$ and $t \geq a_\rho$. We then say that (X, t) *serves* ρ and the *waiting cost* of ρ in S is defined as $\mathsf{wcost}(\rho, \mathsf{S}) = \omega_\rho(t)$. Furthermore, the request ρ is called *pending* at all times in the interval $[a_\rho, t]$. Schedule S is called *feasible* if all requests in $\mathcal{R}$ are served by S.

The cost of a feasible schedule S, denoted $\mathsf{cost}(\mathsf{S})$, is defined by

$$\mathsf{cost}(\mathsf{S}) = \mathsf{scost}(\mathsf{S}) + \mathsf{wcost}(\mathsf{S}),$$

where $\mathsf{scost}(\mathsf{S})$ is the total service cost and $\mathsf{wcost}(\mathsf{S})$ is the total waiting cost, that is

$$\mathsf{scost}(\mathsf{S}) = \sum_{(X,t) \in \mathsf{S}} \ell(X) \quad \text{and} \quad \mathsf{wcost}(\mathsf{S}) = \sum_{\rho \in \mathcal{R}} \mathsf{wcost}(\rho, \mathsf{S}).$$

The objective of MLAP is to compute a feasible schedule S for $\mathcal{J}$ with minimum $\mathsf{cost}(\mathsf{S})$.

Online algorithms

We use the standard and natural definition of online algorithms and the competitive ratio. We assume the continuous time model. The computation starts at time 0 and from then on the time gradually progresses. At any time t new requests can arrive. If the current time is t, the algorithm has complete information about the requests that arrived up until time t, but has no information about any requests whose arrival times are after time t. The instance includes a time horizon H that is not known to the online algorithm, which is revealed only at time $t = H$. At time H, all requests that are still pending must be served. (In the offline case, H can be assumed to be equal to the maximum request arrival time.)

If $\mathcal{A}$ is an online algorithm and $R \geq 1$, we say that $\mathcal{A}$ is *R-competitive* if $\mathsf{cost}(\mathsf{S}) \leq R \cdot \mathsf{opt}(\mathcal{J})$ for any instance $\mathcal{J}$ of MLAP, where S is the schedule computed by $\mathcal{A}$ on $\mathcal{J}$ and $\mathsf{opt}(\mathcal{J})$ is the optimum cost for $\mathcal{J}$.

Quasi-root assumption

Throughout the paper we will assume that r, the root of $\mathcal{T}$, has only one child. This is without loss of generality, because if we have an algorithm (online or offline) for MLAP on such trees, we can apply it independently to each child of r and its subtree. This will give us an algorithm for MLAP on arbitrary trees with the same performance. From now on, let us call the single child the *quasi-root* of $\mathcal{T}$ and denote it by q. Note that q is included in every (non-trivial) service.

Reduction to L-Decreasing Trees

One basic intuition that emerges from earlier works on trees of depth 2 (see [10, 9, 6]) is that the hardest case of the problem is when ℓ_q, the weight of the quasi-root, is much larger than the weights of leaves. For arbitrary depth trees, the hard case is when the weights of nodes quickly decrease with their depth. We show that this is indeed the case, by defining the notion of L-decreasing trees that captures this intuition and showing that MLAP reduces to the special case of MLAP for such L-decreasing trees, increasing the competitive ratio by a factor of at most DL. This is a general result, not limited only to algorithms in our paper.

Formally, for $L \geq 1$, we say that $\mathcal{T}$ is *L-decreasing* if for each node $u \neq r$ and each child v of u we have $\ell_u \geq L \cdot \ell_v$. (The value of L used in our algorithms will be fixed later.)

▶ **Theorem 2.1.** *Assume that there exists an R-competitive algorithm $\mathcal{A}$ for MLAP (resp. MLAP-D) on L-decreasing trees (where R can be a function of D, the tree depth). Then there exists a (DLR)-competitive algorithm $\mathcal{B}$ for MLAP (resp. MLAP-D) on arbitrary trees.*

Proof. Fix the underlying instance $\mathcal{J} = (\mathcal{T}, \mathcal{R})$, where $\mathcal{T}$ is a tree and $\mathcal{R}$ is a sequence of requests in $\mathcal{T}$.

We start by constructing an L-decreasing tree $\mathcal{T}'$ on the same set of nodes. For any node $u \in \mathcal{T} - \{r\}$, the parent of u in $\mathcal{T}'$ will be the lowest (closest to u) ancestor w of u in $\mathcal{T}$ such that $\ell_w \geq L \cdot \ell_u$; if no such w exists, we take $w = r$. Note that $\mathcal{T}'$ may violate the quasi-root assumption, which does not change the validity of the reduction, as we may use independent instances of the algorithm for each child of r in $\mathcal{T}'$. Since in $\mathcal{T}'$ each node u is connected to one of its ancestors from $\mathcal{T}$, it follows that $\mathcal{T}'$ is a tree rooted at r with depth at most D. Obviously, $\mathcal{T}'$ is L-decreasing.

It follows that if a set of vertices X is a service subtree of $\mathcal{T}$, then it is also a service subtree for $\mathcal{T}'$. (Note that the actual topology of the trees induced by X in $\mathcal{T}$ and $\mathcal{T}'$ may be very different.) Thus, also any schedule for $\mathcal{J}$ is also a schedule for $\mathcal{J}' = (\mathcal{T}', \mathcal{R})$, which gives us that $\mathsf{opt}(\mathcal{J}') \leq \mathsf{opt}(\mathcal{J})$.

The algorithm $\mathcal{B}$ for $\mathcal{T}$ is defined as follows: On a request sequence $\mathcal{R}$, we simulate $\mathcal{A}$ for $\mathcal{R}$ in $\mathcal{T}'$, and whenever $\mathcal{A}$ contains a service X, $\mathcal{B}$ issues the service $X' \supseteq X$, created from X as follows: Start with $X' = X$. Then, for each $u \in X - \{r\}$, if w is the parent of u in $\mathcal{T}'$, then add to X' all inner nodes on the path from u to w in $\mathcal{T}$. By the construction of $\mathcal{T}'$, for each u we add at most $D - 1$ nodes, each of cost less than $L \cdot \ell_u$. It follows that $\ell(X') \leq ((D-1)L + 1)\ell(X) \leq DL \cdot \ell(X)$.

In total, the service cost of $\mathcal{B}$ is at most DL times the service cost of $\mathcal{A}$. Any request served by $\mathcal{A}$ is served by $\mathcal{B}$ at the same time or earlier, thus the waiting cost of $\mathcal{B}$ is at most the waiting cost of $\mathcal{A}$ (resp. for MLAP-D, $\mathcal{B}$ produces a valid schedule for $\mathcal{J}$). Since $\mathcal{A}$ is R-competitive, we obtain

$$\mathsf{cost}(\mathcal{B}, \mathcal{J}) \leq DL \cdot \mathsf{cost}(\mathcal{A}, \mathcal{J}') \leq DLR \cdot \mathsf{opt}(\mathcal{J}') \leq DLR \cdot \mathsf{opt}(\mathcal{J}),$$

and thus $\mathcal{B}$ is DLR-competitive. ◀

3 A Competitive Algorithm for MLAP-D

In this section we present our online algorithm for MLAP-D with competitive ratio at most $D^2 2^D$. To this end, we will give an online algorithm that achieves competitive ratio $R_L = (2 + 1/L)^{D-1}$ for L-decreasing trees. Taking $L = D/2$ and using the reduction to L-decreasing trees from Theorem 2.1 then leads to a $D^2 2^D$-competitive algorithm for arbitrary trees.

3.1 Intuitions

Consider the optimal 2-competitive algorithm for MLAP-D for trees of depth 2 [6]. Assume that the tree is L-decreasing, for some large L. (Thus $\ell_q \gg \ell_v$, for each leaf v.) Whenever a pending request reaches its deadline, this algorithm serves a subtree X consisting of r, q and the set of leaves with the earliest deadlines and total weight of about ℓ_q. This is a natural strategy: We have to pay at least ℓ_q to serve the expiring request, so including an additional set of leaves of total weight ℓ_q can at most double our overall cost. But, assuming that no new requests arrive, serving this X can significantly reduce the cost in the future, since servicing these leaves individually is expensive: it would cost $\ell_v + \ell_q$ per each leaf v, compared to the incremental cost of ℓ_v to include v in X.

For L-decreasing trees with three levels (that is, for $D = 3$), we may try to iterate this idea. When constructing a service tree X, we start by adding to X the set of most urgent children of q whose total weight is roughly ℓ_q. Now, when choosing nodes of depth 3, we have two possibilities: (1) for each $v \in X - \{r, q\}$ we can add to X its most urgent children of combined weight ℓ_v (note that their total weight will add up to roughly ℓ_q, because of the L-decreasing property), or (2) from the set of *all* children of the nodes in $X - \{r, q\}$, add to X the set of total weight roughly ℓ_q consisting of (globally) most urgent children.

It is not hard to show that the option (1) does not lead to a constant-competitive algorithm: The counter-example involves an instance with one node w of depth 2 having many children with requests with early deadlines and all other leaves having requests with very distant deadlines. Assume that $\ell_q = L^2$, $\ell_w = L$, and that each leaf has weight 1. The example forces the algorithm to serve the children of w in small batches of size L with cost more than L^2 per batch or L per each child of w, while the optimum can serve all the requests in the children of w at once with cost $O(1)$ per request, giving a lower bound $\Omega(L)$ on the competitive ratio. (The requests at other vertices can be ignored in the optimal solution, as we can keep repeating the above strategy, similar to the lower-bound technique for 1P-MLAP that will be described in the full version of the paper. Reissuing requests at the vertices other than w will not increase the cost of the optimum.) A more intricate example shows that option (2) by itself is not sufficient to guarantee constant competitiveness either.

The idea behind our algorithm, for trees of depth $D = 3$, is to do *both* (1) and (2) to obtain X. This increases the cost of each service by a constant factor, but it protects the algorithm against both bad instances. The extension of our algorithm to depths $D > 3$ carefully iterates the process of constructing the service tree X, to ensure that for each node $v \in X$ and for each level i below v we add to X sufficiently many urgent descendants of v at that level.

3.2 Notations

To give a formal description, we need some more notations. For any set of nodes $Z \subseteq \mathcal{T}$, let Z^i denote the set of nodes in Z of depth i in tree $\mathcal{T}$ (recall that r has depth 0, q has depth 1, and leaves have depth at most D). Let also $Z^{<i} = \bigcup_{j=0}^{i-1} Z^j$ and $Z^{\leq i} = Z^{<i} \cup Z^i$. These notations can be combined with the notation Z_x, so, e.g., $Z_x^{<i}$ is the set of all descendants of x in Z whose depth in $\mathcal{T}$ is smaller than i.

Let $f : \mathcal{T} \to \mathbb{R} \cup \{+\infty\}$ be some function that measures urgency of nodes, so that the nodes with smaller values of f are more urgent. (For MLAP-D, urgency is measured by the deadlines, in which case the function f is set to d^t, while for MLAP we need to use a different measure.) For any set A of nodes in $\mathcal{T}$ and a real number β, let $\mathsf{Urgent}(A, \beta, f)$ be a set of nodes obtained by choosing the nodes from A in order of urgency, until either their total weight exceeds β or we run out of nodes. More precisely, we define $\mathsf{Urgent}(A, \beta, f)$ as a smallest set of nodes in A such that (i) for all $u \in \mathsf{Urgent}(A, \beta, f)$, and $v \in A - \mathsf{Urgent}(A, \beta, f)$ we have $f(u) \leq f(v)$, and (ii) either $\ell(\mathsf{Urgent}(A, \beta, f)) \geq \beta$ or $\mathsf{Urgent}(A, \beta, f) = A$.

We assume that all the deadlines in the given instance are distinct. This may be done without loss of generality, as in case of ties we can modify the deadlines by infinitesimally small perturbations and obtain an algorithm for the general case.

At any given time t during the computation of the algorithm, for each node v, let $d^t(v)$ denote the earliest deadline among all requests in $\mathcal{T}_v$ (i.e., among all descendants of v) that are pending for the algorithm; if there is no pending request in $\mathcal{T}_v$, we set $d^t(v) = +\infty$. We will use the function d^t as the urgency of vertices at time t, i.e., a node u will be considered more urgent than a node v if $d^t(u) < d^t(v)$.

3.3 Algorithm OnlTreeD

At any time t when some request expires, that is when $t = d^t(r)$, the algorithm serves a subtree X constructed by first initializing $X = \{r, q\}$, and then incrementally augmenting X according to the following pseudo-code:

```
for each depth $i = 2, \dots, D$
    $Z^i \leftarrow$ set of all children of nodes in $X^{i-1}$
    for each $v \in X^{<i}$
        $U(v, i, t) \leftarrow \mathsf{Urgent}(Z^i_v, \ell_v, d^t)$
        $X \leftarrow X \cup U(v, i, t)$
```

In other words, at depth i, we restrict our attention to Z^i, the children of all the nodes in X^{i-1}, i.e., of the nodes that we have previously selected to X at level $i-1$. (We start with $i = 2$ and $X^1 = \{q\}$.) Then we iterate over all $v \in X^{<i}$ and we add to X the set $U(v, i, t)$ of vertices from $\mathcal{T}^i_v$ (descendants of v at depth i) whose parents are in X, one by one, in the order of increasing deadlines, stopping when either their total weight exceeds ℓ_v or when we run out of such vertices. Note that these sets do not need to be disjoint.

The constructed set X is a service tree, as we are adding to it only nodes that are children of the nodes already in X.

Let ρ be the request triggering the service at time t, i.e., satisfying $d_\rho = t$. (By the assumption about different deadlines, ρ is unique.) Naturally, all the nodes u on the path from r to σ_ρ have $d^t(u) = t$ and qualify as the most urgent, thus the node σ_ρ is included in X. Therefore every request is served before its deadline.

Intuitively, it should be clear that the described algorithm cannot have a better competitive ratio than $\ell(X)/\ell_q$: If all requests are in q, the optimum will serve only q, while our algorithm uses a set X with many nodes that turn out to be useless. As we will show, via an iterative charging argument, the ratio $\ell(X)/\ell_q$ is actually achieved by the algorithm.

Recall that $R_L = (2 + 1/L)^{D-1}$. We now prove a bound on the cost of the service tree.

▶ **Lemma 3.1.** *Let X be the service tree produced by Algorithm* OnlTreeD *at time t. Then $\ell(X) \le R_L \cdot \ell_q$.*

Proof. We prove by induction that $\ell(X^{\le i}) \le (2 + 1/L)^{i-1}\ell_q$ for all $i \le D$.

The base case of $i = 1$ is trivial, as $X^{\le 1} = \{r, q\}$ and $\ell_r = 0$. For $i \ge 2$, X^i is a union of the sets $U(v, i, t)$ over all nodes $v \in X^{<i}$. Since $\mathcal{T}$ is L-decreasing, each node in the set $U(v, i, t)$ has weight at most ℓ_v/L. Thus the total weight of $U(v, i, t)$ is at most $\ell(U(v, i, t)) \le \ell_v + \ell_v/L = (1 + 1/L)\ell_v$. Therefore, by the inductive assumption, we get that

$$\begin{aligned}\ell(X^{\le i}) &\le (1 + (1 + 1/L)) \cdot \ell(X^{<i}) \\ &\le (2 + 1/L) \cdot (2 + 1/L)^{i-2}\ell_q = (2 + 1/L)^{i-1}\ell_q\,,\end{aligned}$$

proving the induction step and completing the proof that $\ell(X) \le R_L \cdot \ell_q$. ◀

3.4 Analysis

The competitive analysis uses a charging scheme. Fix some optimal schedule S^*. Consider a service (X, t) of Algorithm OnlTreeD. We will identify in X a subset of "critically overdue" nodes (to be defined shortly) of total weight at least $\ell_q \geq \ell(X)/R_L$, and we will show that for each such critically overdue node v we can charge the portion ℓ_v of the service cost of X to an earlier service in S^* that contains v. Further, any node in service of S^* will be charged at most once. This implies that the total cost of our algorithm is at most R_L times the optimal cost, giving us an upper bound of R_L on the competitive ratio for L-decreasing trees.

In the proof, by opt_v^t we denote the time of the first service in S^* that includes v and is strictly after time t; we also let $\mathsf{opt}_v^t = +\infty$ if no such service exists. For a service (X, t) of the algorithm, we say that a node $v \in X$ is *overdue* at time t if $d^t(v) < \mathsf{opt}_v^t$. Servicing of such v is delayed in comparison to S^*, because S^* must have served v before (or at) time t. Note also that r and q are overdue at time t, as $d^t(r) = d^t(q) = t$ by the choice of the service time. We define $v \in X$ to be *critically overdue* at time t if (i) v is overdue at t, and (ii) there is no other service of the algorithm in the time interval (t, opt_v^t) in which v is overdue.

We are now ready to define the charging for a service (X, t). For each $v \in X$ that is critically overdue, we charge its weight ℓ_v to the last service of v in S^* before or at time t. This charging is well-defined as, for each overdue v, there must exist a previous service of v in S^*. The charging is obviously one-to-one because between any two services in S^* that involve v there may be at most one service of the algorithm in which v is critically overdue. The following lemma shows that the total charge from X is large enough.

► **Lemma 3.2.** *Let (X, t) be a service of Algorithm* OnlTreeD *and suppose that $v \in X$ is overdue at time t. Then the total weight of critically overdue nodes in X_v at time t is at least ℓ_v.*

Proof. The proof is by induction on the depth of $\mathcal{T}_v$, the subtree rooted at v.

The base case is when $\mathcal{T}_v$ has depth 0, that is when v is a leaf. We show that in this case v must be critically overdue, which implies the conclusion of the lemma. Towards contradiction, suppose that there is some other service at time $t' \in (t, \mathsf{opt}_v^t)$ in which v is overdue. Since v is a leaf, after the service at time t there are no pending requests in $\mathcal{T}_v = \{v\}$. This would imply that there is a request ρ with $\sigma_\rho = v$ such that $t < a_\rho \leq d_\rho < \mathsf{opt}_v^t$. But this is not possible, because S^* does not serve v in the time interval (t, opt_v^t). Thus v is critically overdue and the base case holds.

Assume now that v is not a leaf, and that the lemma holds for all descendants of v. If v is critically overdue, the conclusion of the lemma holds.

Thus we can now assume that v is not critically overdue. This means that there is a service (Y, t') of Algorithm OnlTreeD with $t < t' < \mathsf{opt}_v^t$ which contains v and such that v is overdue at t'. Thus $\mathsf{opt}_v^t = \mathsf{opt}_v^{t'}$.

Let ρ be the request with $d_\rho = d^{t'}(v)$, i.e., the most urgent request in $\mathcal{T}_v$ at time t'.

We claim that $a_\rho \leq t$, i.e., ρ arrived no later than at time t. Indeed, since v is overdue at time t', it follows that $d_\rho < \mathsf{opt}_v^{t'} = \mathsf{opt}_v^t$. The optimal schedule S^* cannot serve ρ after time t, as S^* has no service from v in the interval $(t, d_\rho]$. Thus S^* must have served ρ before or at t, and hence $a_\rho \leq t$, as claimed.

Now consider the path from σ_ρ to v in Y. (See Figure 1.) As ρ is pending for the algorithm at time t and ρ is not served by (X, t), it follows that $\sigma_\rho \notin X$. Let w be the last node on this path in $Y - X$. Then w is well-defined and $w \neq v$, as $v \in X$. Let i be the depth of w. Note that the parent of w is in $X_v^{<i}$, so $w \in Z^i$ in the algorithm when X is constructed.

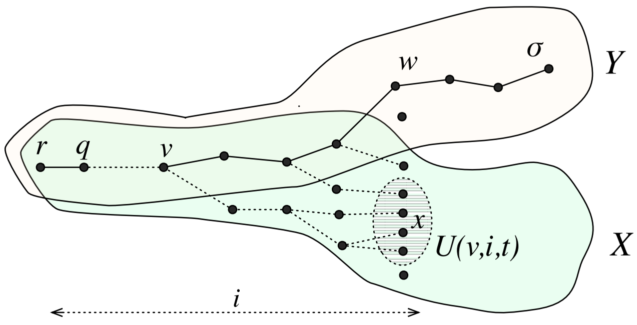


Figure 1 Illustration of the proof of Lemma 3.2.

The node σ_ρ is in $\mathcal{T}_w$ and ρ is pending at t, thus we have $d^t(w) \leq d_\rho$. Since $w \in Z^i$ but w was not added to X at time t, we have that $\ell(U(v,i,t)) \geq \ell_v$ and each $x \in U(v,i,t)$ is at least as urgent as w. This implies that such x satisfies

$$d^t(x) \leq d^t(w) \leq d_\rho < \mathsf{opt}_v^{t'} = \mathsf{opt}_v^t \leq \mathsf{opt}_x^t,$$

thus x is overdue at time t. By the inductive assumption, the total weight of critically overdue nodes in each subtree X_x is at least ℓ_x. Adding these weights over all $x \in U(v,i,t)$, we obtain that the total weight of critically overdue nodes in X_v is least $\ell(U(v,i,t)) \geq \ell_v$, completing the proof. ◀

Now consider a service (X,t) of the algorithm. The quasi-root q is overdue at time t, so Lemmas 3.2 and 3.1 imply that the charge from (X,t) is at least $\ell_q \geq \ell(X)/R_L$. Since each node in any service in S^* is charged at most once, we conclude that Algorithm OnlTreeD is R_L-competitive for any L-decreasing $\mathcal{T}$.

From the previous paragraph, using Theorem 2.1, we now obtain that there exists a $DLR_L = DL(2+1/L)^{D-1}$-competitive algorithm for general trees. For $D \geq 2$, choosing $L = D/2$ yields a competitive ratio bounded by $\frac{1}{2}D^2 2^{D-1} \cdot (1+1/D)^D \leq \frac{1}{4}D^2 2^D \cdot \mathrm{e} \leq D^2 2^D$. (For $D = 1$ there is a trivial 1-competitive algorithm for MLAP-D.) Summarizing, we obtain the following result.

▶ **Theorem 3.3.** *There exists a $D^2 2^D$-competitive online algorithm for MLAP-D.*

4 A Competitive Algorithm for MLAP

In this section we give an online algorithm for MLAP that achieves a constant competitive ratio for trees of bounded depth. Specifically, for trees of depth D the competitive ratio of this algorithm is at most $O(D^4 2^D)$. As before, Theorem 2.1 allows us to assume that the tree in the instance is L-decreasing.

4.1 Preliminaries and notations

We use some of the notation introduced in Section 3 and introduce some more.

Recall that $\omega_\rho(t)$ is the waiting cost function of a request ρ which is assumed to be continuous. We will overload this notation, so that we can talk about the waiting cost function for a set of requests or a set of vertices. Specifically, for a set P of requests and a set Z of nodes, let

$$\omega_P(Z,t) = \sum_{\rho \in P : \sigma_\rho \in Z} \omega_\rho(t).$$

Thus $\omega_P(Z,t)$ is the total waiting cost of the requests from P that are issued in Z.

Maturity time

In the algorithm for MLAP-D, the times of services and the urgency of nodes are both naturally determined by the deadlines. For MLAP with continuous waiting costs there are no hard deadlines; however, we can still introduce a useful notion of *maturity time* of a node, which is, roughly speaking, the time when some subtree rooted at this node has its waiting cost equal to its service cost; this subtree is then called *mature*. This turns out to be natural both for the quasi-root as the time to service it and for the other vertices as a measure of their urgency in the algorithm. We proceed to define these notions.

Consider some time t and any set $P \subseteq \mathcal{R}$ of requests. A subtree Z of $\mathcal{T}$ (not necessarily rooted at r) is called *P-mature* at time t if $\omega_P(Z,t) \geq \ell(Z)$. Also, let $\mu_P(Z)$ denote the minimal time τ such that $\omega_P(Z,\tau) = \ell(Z)$; we let $\mu_P(Z) = \infty$ if such τ does not exist. In other words, $\mu_P(Z)$ is the earliest τ at which Z is P-mature. Since $\omega_P(Z,0) = 0$ and $\omega_P(Z,t)$ is non-decreasing and continuous function of t, $\mu_P(Z)$ is well-defined.

For a node v, let the *P-maturity time of v*, denoted $M_P(v)$, be the minimum of values $\mu_P(Z)$ over all subtrees Z of $\mathcal{T}$ rooted at v. The tree Z that achieves this minimum will be denoted $C_P(v)$ and called the *P-critical tree rooted at v*; if there are more such trees, choose one arbitrarily.

The following simple lemma guarantees that the maturity time of any node in the P-critical tree $C_P(v)$ is at most the maturity time of v.

▶ **Lemma 4.1.** *Let $u \in C_P(v)$ and let $Y = (C_P(v))_u$ be the subtree of the $C_P(v)$ rooted at u. Then $M_P(u) \leq \mu_P(Y) \leq M_P(v)$.*

Proof. The first inequality is trivial. To show the second inequality, we proceed by contradiction. If the second inequality does not hold, then $u \neq v$ and $\omega_P(Y, M_P(v)) < \ell(Y)$. Take $Y' = C_P(v) \setminus Y$, which is a tree rooted at v. Since $\omega_P(C_P(v), M_P(v)) = \ell(C_P(v))$, we have that $\omega_P(Y', M_P(v)) = \omega_P(C_P(v), M_P(v)) - \omega_P(Y, M_P(v)) > \ell(C_P(v)) - \ell(Y) = \ell(Y')$. This in turn implies that $\mu_P(Y') < M_P(v)$, which is a contradiction with the definition of $M_P(v)$. ◀

Most of the references to maturity of a node or to its critical set will be made with respect to the set of requests pending for our algorithm at a given time. (In particular, if the algorithm schedules a service at some time t, the maturity times are computed with respect to the requests that are pending at time t *before* the service is executed.) Thus, for any time t, we will use notation $M^t(v)$ and $C^t(v)$ to denote the time $M_P(v)$ and the P-critical tree $C_P(v)$, where P is the set of requests pending for the algorithm at time t. Note that in general it is possible that $M^t(v) < t$. However, our algorithm will maintain the invariant that for the quasi-root q we will have $M^t(q) \geq t$ at each time t.

4.2 Algorithm

We now describe our algorithm. A service will occur at any maturity time of the quasi-root q, that is at a time t for which $t = M^t(q)$. At such a time, the algorithm chooses a service that contains the critical tree C of q and an extra set E, whose service cost is not much more expensive than that of C. The extra set is constructed similarly as in OnlTreeD, where the urgency of nodes is now measured by their maturity time. As before, this extra set will be a union of a system of sets $U(v,i,t)$ for $i = 2,\ldots,D$, and $v \in C^{<i} \cup E^{<i}$, except that now the sets $U(v,i,t)$ will be mutually disjoint and also disjoint from C.

Algorithm OnlTree

At any time t such that $t = M^t(q)$, serve the set $X = C \cup E$ constructed according to the following pseudo-code:

```
$C \leftarrow C^t(q) \cup \{r\}$
$E \leftarrow \emptyset$
for each depth $i = 2,\ldots,D$
    $Z^i \leftarrow$ set of all nodes in $\mathcal{T}^i \setminus C$ whose parent is in $C \cup E$
    for each $v \in (C \cup E)^{<i}$
        $U(v,i,t) \leftarrow \mathsf{Urgent}(Z^i_v, \ell_v, M^t)$
        $E \leftarrow E \cup U(v,i,t)$
        $Z^i \leftarrow Z^i \setminus U(v,i,t)$
```

At the end of the instance (when $t = H$, the time horizon) issue the last service that contains all vertices v with a pending request in $\mathcal{T}_v$.

The analysis of the algorithm will be given in the full version of the paper. We obtain:

▶ **Theorem 4.2.** *There exists an $O(D^4 2^D)$-competitive algorithm for MLAP on trees of depth D.*

5 Single-Phase MLAP

We now consider a restricted variant of MLAP that we refer to as *Single-Phase* MLAP, or 1P-MLAP. In 1P-MLAP all requests arrive at the beginning, at time 0. The instance also includes a parameter θ representing the common *expiration time* for all requests. We do not require that all requests are served. Any unserved request pays only the cost of waiting until the expiration time θ.

In the online variant of 1P-MLAP, all requests, including their waiting cost functions, are known to the online algorithm at time 0. The only unknown is the expiration time θ.

Although not explicitly named, variants of 1P-MLAP have been considered in [10, 7], where they were used to show lower bounds on competitive ratios for MLAP. These proofs consist of two steps, first showing a lower bound for online 1P-MLAP and then arguing that, in the online scenario, 1P-MLAP can be expressed as a special case of MLAP. (A corresponding property holds in the offline case as well, but is quite trivial.)

Since most lower bounds on competitive ratio of online algorithms for variants of MLAP use Single-Phase instances, a natural approach is to try such a construction in an attempt to construct super-constant lower bounds for MLAP. We show that this approach cannot be successful in the context of MLAP.

▶ **Theorem 5.1.** *There exists a 4-competitive algorithm for the Single-Phase MLAP.*

A detailed description of the algorithm will be given in the full version of the paper.

6 MLAP on Paths

We now consider the case when the tree is just a path. For simplicity we will assume a generalization to the continuous case, that we refer to as *the MLAP problem on the line*, when the path is represented by the half-line $[0, \infty)$; that is the requests can occur at any point $x \in [0, \infty)$. Then the point 0 corresponds to the root, each vertex is a point $x \in [0, \infty)$, and each service is an interval of the form $[0, x]$. We say that an algorithm *delivers from* x if it serves the interval $[0, x]$.

We provide several results for the MLAP problem on the line. We first prove that the competitive ratio of MLAP-D (the variant with deadlines) on the line is exactly 4, by providing matching upper and lower bounds. Then later we will show that the lower bound of 4 can be modified to work for MLAP-L (that is, for linear waiting costs).

Algorithm OnlLine

The algorithm creates a service only when a deadline of a pending request is reached. If a deadline of a request at x is reached, then OnlLine delivers from $2x$.

▶ **Theorem 6.1.** *Algorithm* OnlLine *is* 4*-competitive for MLAP-D on the line.*

Proof. The proof uses a charging strategy. We represent each adversary service, say when the adversary delivers from a point y, by an interval $[0, y]$. The cost of each service of OnlLine is then charged to a segment of one of those adversary service intervals.

Consider a service triggered by a deadline t of a request ρ at some point x; that is, OnlLine delivered from $2x$. The adversary must have served ρ between its arrival time and its deadline t. Fix the last such service of the adversary, where at a time $t' \leq t$ the adversary delivered from a point $x' \geq x$. We charge the cost $2x$ of the algorithm's service to the segment $[x/2, x]$ of the adversary's service interval $[0, x']$ at time t'.

We now claim that no part of the adversary's service is charged twice. To justify this claim, suppose that there are two services of OnlLine, at times $t_1 < t_2$, triggered by requests from points x_1 and x_2, respectively, that both charge to an adversary's service from x' at time $t' \leq t_1$. By the definition of charging, the request at x_2 was already present at time t'. As x_2 was not served by OnlLine's service at t_1, it means that $x_2 > 2x_1$, and thus the charged segments $[x_1/2, x_1]$ and $[x_2/2, x_2]$ of the adversary service interval at time t' are disjoint.

Summarizing, for any adversary service interval $[0, y]$, its charged segments are disjoint. Any charged segment receives the charge equal to 4 times its length. Thus this interval receives the total charge at most $4y$. This implies that the competitive ratio is at most 4. ◀

Lower bounds

We now show lower bounds of 4 for MLAP-D and MLAP-L on the line. In both proofs we show the bound for the corresponding variant of 1P-MLAP, using a reduction from the online bidding problem [13, 12]. Roughly speaking, in online bidding, for a given universe $\mathcal{U}$ of real numbers, the adversary chooses a secret value $u \in \mathcal{U}$ and the goal of the algorithm is to find an upper-bound on u. To this end, the algorithm outputs an increasing sequence of numbers $x_1, x_2, x_3, \ldots$. The game is stopped after the first x_k that is at least u and the bidding ratio is then defined as $\sum_{i=1}^{k} x_i/u$.

Chrobak et al. [12] proved that the optimal bidding ratio is exactly 4, even if it is restricted to sets $\mathcal{U}$ of the form $\{1, 2, \ldots, B\}$, for some integer B. More precisely, they proved the following result.

▶ **Lemma 6.2.** *For any $R < 4$, there exists B, such that any sequence of integers $0 = x_0 < x_1 < x_2 < \ldots < x_{m-1} < x_m = B$ has an index $k \geq 1$ with $\sum_{i=0}^{k} x_i > R \cdot (x_{k-1} + 1)$.*

▶ **Theorem 6.3.** *There is no online algorithm for MLAP-D on the line with competitive ratio smaller than 4.*

Proof. We show that no online algorithm for 1P-MLAP-D (the deadline variant of 1P-MLAP) on the line can attain competitive ratio smaller than 4. Assume the contrary, i.e., that there exists a deterministic algorithm ALG that is R-competitive, where $R < 4$. Let B be the integer whose existence is guaranteed by Lemma 6.2. We create an instance of 1P-MLAP-D, where for every $x \in \{1, \ldots, B\}$ there is a request at x with deadline x.

Without loss of generality, ALG issues services only at integer times $1, 2, ..., B$. The strategy of ALG can be now defined as a sequence of services at times $t_1 < t_2 < \ldots < t_m$, where at time t_i it delivers from $x_i \in \{t_i, t_i + 1, ..., B\}$. Without loss of generality, $x_1 < x_2 < \ldots < x_m$. We may assume that $x_m = B$ (otherwise the algorithm is not competitive at all); we also add a dummy service from $x_0 = 0$ at time $t_0 = 0$.

The adversary now chooses $k \geq 1$ and stops the game at the expiration time that is right after the algorithm's kth service, say $\theta = t_k + \frac{1}{2}$. ALG's cost is then $\sum_{i=0}^{k} x_i$. The request at $x_{k-1} + 1$ is not served at time t_{k-1}, so, to meet the deadline of this request, the schedule of ALG must satisfy $t_k \leq x_{k-1} + 1$. This implies that $\theta < x_{k-1} + 2$, that is, all requests at points $x_{k-1} + 2, x_{k-1} + 3, ..., B$ expire before their deadlines and do not need to be served. Therefore, to serve this instance, the optimal solution may simply deliver from $x_{k-1} + 1$ at time 0. Hence, the competitive ratio of ALG is at least $\sum_{i=0}^{k} x_i/(x_{k-1} + 1)$. By Lemma 6.2, it is possible to choose k such that this ratio is strictly greater than R, a contradiction with R-competitiveness of ALG. ◀

Next, we show that the same lower bound applies to MLAP-L, the version of MLAP where the waiting cost function is linear. This improves the lower bound of 3.618 from [7].

▶ **Theorem 6.4.** *There is no online algorithm for MLAP-L on the line with competitive ratio smaller than 4.*

Proof. Similarly to the proof of Theorem 6.3, we create an instance of 1P-MLAP-L (the variant of 1P-MLAP with linear waiting cost functions) that does not allow a better than 4-competitive online algorithm. Fix any online algorithm ALG for 1P-MLAP-L and, towards a contradiction, suppose that it is R-competitive, for some $R < 4$. Again, let B be the integer whose existence is guaranteed by Lemma 6.2. In our instance of 1P-MLAP-L, there are 6^{B-x} requests at x for any $x \in \{1, 2, \ldots, B\}$.

Without loss of generality, we make the same assumptions as in the proof of Theorem 6.3: algorithm ALG is defined by a sequence of services at times $0 = t_0 < t_1 < t_2 < \ldots < t_m$, where at each time t_i it delivers from some point x_i. Without loss of generality, we can assume that $0 = x_0 < x_1 < \ldots < x_m = B$.

Again, the strategy of the adversary is to stop the game at some expiration time θ that is right after some time t_k, say $\theta = t_k + \epsilon$, for some small $\epsilon > 0$. The algorithm pays $\sum_{i=0}^{k} x_i$ for serving the requests. The requests at $x_{k-1} + 1$ waited for time t_k in ALG's schedule and hence ALG's waiting cost is at least $6^{B-x_{k-1}-1} \cdot t_k$.

The adversary delivers from point $x_{k-1} + 1$ at time 0. The remaining, unserved requests at points $x_{k-1}+2, x_{k-1}+3, \ldots, B$ pay time θ each for waiting. There are $\sum_{j=x_{k-1}+2}^{B} 6^{B-j} \leq \frac{1}{5} \cdot 6^{B-x_{k-1}-1}$ such requests and hence the adversary's waiting cost is at most $\frac{1}{5} \cdot 6^{B-x_{k-1}-1} \cdot (t_k + \epsilon)$.

Therefore, the algorithm-to-adversary ratio on the waiting costs is at least $5t_k/(t_k+\epsilon)$. For any k we can choose a sufficiently small ϵ so that this ratio is larger than 4. By Lemma 6.2, it is possible to choose k for which the ratio on servicing cost is strictly greater than R. This yields a contradiction to the R-competitiveness of ALG. ◀

We point out that the analysis in the proof above gives some insight into the behavior of any 4-competitive algorithm for 1P-MLAP-L (we know such an algorithm exists, by the results in Section 5), namely that, for the type of instances used in the above proof, its waiting cost must be negligible compared to the service cost.

References

1 Alok Aggarwal and James K. Park. Improved algorithms for economic lot sizing problems. *Operations Research*, 41:549–571, 1993.

2 Esther Arkin, Dev Joneja, and Robin Roundy. Computational complexity of uncapacitated multi-echelon production planning problems. *Operations Research Letters*, 8(2):61–66, 1989.

3 B. R. Badrinath and Pradeep Sudame. Gathercast: the design and implementation of a programmable aggregation mechanism for the internet. In *Proc. 9thInternational Conference on Computer Communications and Networks (ICCCN)*, pages 206–213, 2000.

4 Luca Becchetti, Alberto Marchetti-Spaccamela, Andrea Vitaletti, Peter Korteweg, Martin Skutella, and Leen Stougie. Latency-constrained aggregation in sensor networks. *ACM Transactions on Algorithms*, 6(1):13:1–13:20, 2009.

5 Marcin Bienkowski, Jaroslaw Byrka, Marek Chrobak, Neil B. Dobbs, Tomasz Nowicki, Maxim Sviridenko, Grzegorz Swirszcz, and Neal E. Young. Approximation algorithms for the joint replenishment problem with deadlines. *Journal of Scheduling*, 18(6):545–560, 2015.

6 Marcin Bienkowski, Jaroslaw Byrka, Marek Chrobak, Łukasz Jeż, Dorian Nogneng, and Jiří Sgall. Better approximation bounds for the joint replenishment problem. In *Proc. 25th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 42–54, 2014.

7 Marcin Bienkowski, Jaroslaw Byrka, Marek Chrobak, Łukasz Jeż, Jiří Sgall, and Grzegorz Stachowiak. Online control message aggregation in chain networks. In *Proc. 13th Int. Workshop on Algorithms and Data Structures (WADS)*, pages 133–145, 2013.

8 Edward Bortnikov and Reuven Cohen. Schemes for scheduling of control messages by hierarchical protocols. In *Proc. 17th IEEE Int. Conference on Computer Communications (INFOCOM)*, pages 865–872, 1998.

9 Carlos Brito, Elias Koutsoupias, and Shailesh Vaya. Competitive analysis of organization networks or multicast acknowledgement: How much to wait? *Algorithmica*, 64(4):584–605, 2012.

10 Niv Buchbinder, Tracy Kimbrel, Retsef Levi, Konstantin Makarychev, and Maxim Sviridenko. Online make-to-order joint replenishment model: Primal-dual competitive algorithms. In *Proc. 19th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 952–961, 2008.

11 Niv Buchbinder and Joseph (Seffi) Naor. The design of competitive online algorithms via a primal-dual approach. *Foundations and Trends in Theoretical Computer Science*, 3(2–3):93–263, 2009.

12 Marek Chrobak, Claire Kenyon, John Noga, and Neal E. Young. Incremental medians via online bidding. *Algorithmica*, 50(4):455–478, 2008.

13 Marek Chrobak and Claire Kenyon-Mathieu. SIGACT news online algorithms column 10: competitiveness via doubling. *SIGACT News*, 37(4):115–126, 2006.

14 Wallace B. Crowston and Michael H. Wagner. Dynamic lot size models for multi-stage assembly systems. *Management Science*, 20(1):14–21, 1973.

15 Daniel R. Dooly, Sally A. Goldman, and Stephen D. Scott. On-line analysis of the TCP acknowledgment delay problem. *Journal of the ACM*, 48(2):243–273, 2001.

16 Fei Hu, Xiaojun Cao, and C. May. Optimized scheduling for data aggregation in wireless sensor networks. In *Int. Conference on Information Technology: Coding and Computing (ITCC)*, volume 2, pages 557–561, 2005.

17 Anna R. Karlin, Claire Kenyon, and Dana Randall. Dynamic TCP acknowledgement and other stories about e/(e - 1). *Algorithmica*, 36(3):209–224, 2003.

18 Sanjeev Khanna, Joseph Naor, and Danny Raz. Control message aggregation in group communication protocols. In *Proc. 29th Int. Colloq. on Automata, Languages and Programming (ICALP)*, pages 135–146, 2002.

19 Alf Kimms. *Multi-Level Lot Sizing and Scheduling: Methods for Capacitated, Dynamic, and Deterministic Models.* Springer-Verlag, 1997.

20 Douglas M Lambert and Martha C Cooper. Issues in supply chain management. *Industrial Marketing Management*, 29(1):65–83, 2000.

21 Retsef Levi, Robin Roundy, and David B. Shmoys. A constant approximation algorithm for the one-warehouse multi-retailer problem. In *Proc. 16th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 365–374, 2005.

22 Retsef Levi, Robin Roundy, and David B. Shmoys. Primal-dual algorithms for deterministic inventory problems. *Mathematics of Operations Research*, 31(2):267–284, 2006.

23 Retsef Levi, Robin Roundy, David B. Shmoys, and Maxim Sviridenko. A constant approximation algorithm for the one-warehouse multiretailer problem. *Management Science*, 54(4):763–776, 2008.

24 Retsef Levi and Maxim Sviridenko. Improved approximation algorithm for the one-warehouse multi-retailer problem. In *Proc. 9th Int. Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX)*, pages 188–199, 2006.

25 Tim Nonner and Alexander Souza. Approximating the joint replenishment problem with deadlines. *Discrete Mathematics, Algorithms and Applications*, 1(2):153–174, 2009.

26 Christos Papadimitriou. Computational aspects of organization theory. In *Proc. 4th European Symp. on Algorithms (ESA)*, pages 559–564, 1996.

27 Lehilton L. C. Pedrosa. Private communication. 2013.

28 Steven S. Seiden. A guessing game and randomized online algorithms. In *Proc. 32nd ACM Symp. on Theory of Computing (STOC)*, pages 592–601, 2000.

29 Shailesh Vaya. Brief announcement: Delay or deliver dilemma in organization networks. In *Proc. 31st ACM Symp. on Principles of Distributed Computing (PODC)*, pages 339–340, 2012.

30 H.M. Wagner and T. Whitin. Dynamic version of the economic lot size model. *Management Science*, 5:89–96, 1958.

31 Wei Yuan, Srikanth V. Krishnamurthy, and Satish K. Tripathi. Synchronization of multiple levels of data fusion in wireless sensor networks. In *Proc. Global Telecommunications Conference (GLOBECOM)*, pages 221–225, 2003.

Compact and Fast Sensitivity Oracles for Single-Source Distances*

Davide Bilò[1], Luciano Gualà[2], Stefano Leucci[3], and Guido Proietti[4]

1 **DUMAS, Università di Sassari, Sassari, Italy**
`davide.bilo@uniss.it`
2 **DII, Università di Roma "Tor Vergata", Rome, Italy**
`guala@mat.uniroma2.it`
3 **DI, "Sapienza" Università di Roma, Rome, Italy**
`leucci@di.uniroma1.it`
4 **DISIM, Università degli Studi dell'Aquila, L'Aquila, Italy**
IASI, CNR, Roma, Italy
`guido.proietti@univaq.it`

Abstract

Let s denote a distinguished source vertex of a non-negatively real weighted and undirected graph G with n vertices and m edges. In this paper we present two efficient *single-source approximate-distance sensitivity oracles*, namely *compact* data structures which are able to *quickly* report an approximate (by a multiplicative stretch factor) distance from s to any node of G following the failure of any edge in G. More precisely, we first present a sensitivity oracle of size $O(n)$ which is able to report 2-approximate distances from the source in $O(1)$ time. Then, we further develop our construction by building, for any $0 < \varepsilon < 1$, another sensitivity oracle having size $O\left(n \cdot \frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$, and is able to report a $(1+\varepsilon)$-approximate distance from s to any vertex of G in $O\left(\log n \cdot \frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$ time. Thus, this latter oracle is essentially optimal as far as size and stretch are concerned, and it only asks for a logarithmic query time. Finally, our results are complemented with a space lower bound for the related class of single-source *additively-stretched* sensitivity oracles, which is helpful to realize the hardness of designing compact oracles of this type.

1998 ACM Subject Classification G.2.2 [Graph Theory] Graph algorithms, Trees

Keywords and phrases fault-tolerant shortest-path tree, approximate distance, distance sensitivity oracle

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.13

1 Introduction

The term *distance oracle* was coined by Thorup and Zwick [19], to emphasize the quality of a data structure that, despite its sparseness, is able to report very quickly provably good approximate distances between any pair of nodes in a graph. Indeed, it is well-known that in huge graphs the trade-off between time and space for *exact* distance queries is a very critical issue: at its extremes, either we use a quadratic (unfeasible) space to reply in constant time, or we use a linear space to reply at an unsustainable large time. Thus, a wide body of literature focused on the problem of developing intermediate solutions in between these two opposite approaches, with the goal of designing more and more compact and fast oracles. This already complex task is further complicated as soon as edge or vertex failures enter

* A full version of the paper is available at `http://arxiv.org/abs/1608.04769`.

into play: here, the oracle should be able to return (approximate) distances following the failure of some component(s) in the underlying graph, or in other words to be *fault-tolerant*, thus introducing an additional overload to the problem complexity. This kind of oracle is also known as *distance sensitivity oracle*. In this paper we focus our attention on a such challenging scenario, but we restrict our attention to the prominent case in which concerned distances are from a fixed source only, which is of special interest in several network-based applications.

1.1 Related work

Let s denote a distinguished source vertex of a non-negatively real weighted and undirected n-vertex and m-edge graph $G = (V(G), E(G), w)$. For the sake of avoiding technicalities, we assume that G is 2-edge-connected, although this assumption can be easily relaxed without affecting our results. A *single-edge-fault-tolerant α-single-source distance oracle* (EFT α-SSDO in the following), with $\alpha \geq 1$, is a data structure that for any $v \in V(G)$ and any $e \in E(G)$ is able to return an estimate of the distance in $G - e$ (i.e., the graph G deprived by e) between s and v, say $d_{G-e}(s, v)$, within the range $[d_{G-e}(s, v), \alpha \cdot d_{G-e}(s, v)]$. The term α is a.k.a. the *stretch factor* of the oracle.

A natural counterpart of such an oracle is an EFT *α-approximate shortest-path tree* (α-ASPT), i.e., a subgraph of G which, besides a SPT of G rooted at s, contains α-stretched shortest paths from s after the failure of any edge e in G. Such a structure is also known as a *single-source EFT α-spanner*. In some sense, a SSDO aims to convert in an explicit form the distance information that a corresponding ASPT may retain just in an implicit form, similarly to the process of maintaining in an n-size array all the distances from the source induced by the paths of a corresponding SPT. However, such a conversion process is far to be trivial in general and should be accomplished carefully, since the exploitation of the implicit information may introduce a dilatation in the final size of the oracle.

While the study of sensitivity oracles for all-pairs distances started right after the first appearance of [19], the single-source case was faced only later. More precisely, in [10] it was first proven that if we aim at *exact* distances, then $\Theta(n^2)$ space may be needed, already for undirected graphs and single edge failures, and independently of the query time. Then, in [1] the authors build in $O(m \log n + n \log^2 n)$ time a *single-vertex-fault-tolerant* (VFT) 3-SSDO of size $O(n \log n)$ and with constant query time. In the same paper, for *unweighted* graphs and for any $\varepsilon > 0$, the authors build in $O(m\sqrt{n/\varepsilon})$ time a VFT $(1 + \varepsilon)$-SSDO of size $O(\frac{n}{\varepsilon^3} + n \log n)$ and with constant query time. Both oracles are *path reporting*, i.e., they are able to report the corresponding approximate shortest path from the source in time proportional to the path size. Moreover, as discussed in [5], in both oracles/spanners the log-term in the size can be removed if *edge* failures are considered, instead of vertex failures. Finally, they can easily be transformed into corresponding E/VFT ASPTs having a same size and stretch. As far as this latter result is concerned, this was improved in [5], where it was given, for any (even non-constant) $\varepsilon > 0$, an E/VFT $(1 + \varepsilon)$-ASPT of size $O(\frac{n \log n}{\varepsilon^2})$, without providing a corresponding oracle, though.

Summarizing, we therefore have the following state-of-the-art for EFT SSDOs: if we insist on having linear-size and constant query time, then a $(1 + \varepsilon)$-stretch can be obtained only for unweighted graphs, while for weighted graphs the best current stretch is 3. Actually, this latter value can be reduced only by either paying a quadratic size (by storing for every $e \in E(G)$, the explicit distances from s in $G - e$), or an almost linear size but a super-linear query time (by storing and then inspecting the structure provided in [5]). So, the main open question is the following: can we develop a good space-time trade-off (ideally, linear space and constant query time) by guaranteeing a stretch less than 3 (ideally, arbitrarily close to 1)? In this paper, we make significant progresses in this direction.

1.2 Our results

Our main result is, for any arbitrary small $\varepsilon > 0$, the construction in $O(mn + n^2 \log n)$ time and $O\left(m + n \cdot \frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$ space of an EFT $(1+\varepsilon)$-SSDO having size $O\left(n \cdot \frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$ and query time $O\left(\log n \cdot \frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$. Thus, when ε is constant w.r.t. n, we get close to the ideal situation we were depicting above: our oracle has linear space, stretch arbitrarily close to 1, and a logarithmic query time. Moreover, it is interesting to notice that size and query time have an almost linear dependency on $1/\varepsilon$.

To the best of our knowledge, this is the first EFT SSDO guaranteeing a $(1+\varepsilon)$-stretch factor on weighted graphs. Interestingly, our construction is not obtained by the EFT $(1+\varepsilon)$-ASPT of size $O(\frac{n \log n}{\varepsilon^2})$ given in [5], whose conversion to a same size-stretch trade-off oracle sounds very hard, and is instead based on a quite different approach. More precisely, to get our size and query time bounds, we select a subset of *landmark* nodes of G, and for each one of them we store $O\left(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$ *exact* post-failure (for an appropriate set of failing edges) distances from s. Then, when an edge e fails and we want to retrieve an approximate distance from s towards a fixed destination node t, we efficiently select with the promised query time a pivotal landmark node that actually sits on a path in $G - e$ from s to t whose length is within the bound. Notice that such a path is not *explicitly* stored in our oracle, so unfortunately we cannot return it in a time proportional to its size (besides the query time). In other words, our oracle is not inherently path-reporting, an we leave this point as a challenging open problem.

To get the reader acquainted with our technique, we first develop in $O(mn + n^2 \log n)$ time and $O(m)$ space an EFT 2-SSDO of size $O(n)$ and constant query time. This result is of independent interest, since it is the first EFT SSDO with both optimal size and query time having a stretch better than the long-standing barrier of 3. In this other oracle, once again we select a subset of landmark nodes of G, but in this case, to get the promised stretch, we do not need to maintain explicitly any exact distances towards them. Rather, for the failure of an edge e of G and for a fixed destination node t, a structural property of 2-stretched post-failure paths will allow us to return the 2-approximate distance from s by simply understanding whether there exists a pivotal landmark node associated with t. Actually, we show that such an association can be established by formulating a corresponding *bottleneck vertex query* problem on a rooted tree, that can be answered in $O(1)$ time by using a linear-size efficient data structure developed in [9].

Finally, in order to better appreciate the quality of our former oracle, we provide a lower bound on the bit size of any EFT *β-additive* SSDO, i.e., an oracle which is able to report a distance from s following an edge failure which is exact unless an additive term β. Notice that for weighted graphs, as in our setting, it only makes sense that such a β is depending on the actual queried distance d. Notice also that our linear-size EFT $(1+\varepsilon)$-SSDO can be revised as an EFT $(\varepsilon \cdot d)$-additive SSDO. So, a naturally arising question is: for a given $0 < \delta \leq 1$, can we devise a compact EFT $(\varepsilon \cdot d^{1-\delta})$-additive SSDO? We provide an answer in the negative, by showing a class of graphs for which a corresponding set of oracles of this sort would contain at least an element of $\Omega(n^2)$ bit size, regardless of its query time. Due to space limitations, the proof of this latter result will be given in the full version of the paper.

1.3 Other related results

Besides the aforementioned related work on single-source distance sensitivity oracles, we mention some further papers on the topic. For directed graphs with integer positive edge weights bounded by M, in [12] the authors show how to build efficiently in $\widetilde{O}(Mn^{\omega})$ time a *randomized* EFT SSDO of size $\Theta(n^2)$ and with $O(1)$ query time, where returned distances are exact w.h.p., and $\omega < 2.373$ denotes the matrix multiplication exponent. As far as *multiple* edge failures are concerned, in [6], for the failure of any set $F \subseteq E(G)$ of at most f edges of G, the authors build in $O(fm\,\alpha(m,n) + fn\log^3 n)$ time an f-EFT $(2|F|+1)$-SSDO of size $O(\min\{m, fn\}\log^2 n)$, with a query time of $O(|F|^2 \log^2 n)$, and that is also able to report the corresponding path in the same time plus the path size. Notice that this oracle is obtained by converting a corresponding single-source f-EFT spanner having size $O(fn)$ and a same stretch. Notice also that if one is willing to use $O(m\log^2 n)$ space, such oracle will be able to handle any number of edge failures (i.e., up to m). Recently in [8], the authors faced the special case of *shortest-path failures*, in which the failure of a set F of at most f adjacent edges along any source-leaf path has to be tolerated. For this problem, they build in $O(n(m+f^2))$ time, a $(2k-1)(2|F|+1)$-SSDO of size $O(kn\,f^{1+1/k})$ and constant query time, where $|F|$ denotes the size of the actual failing path, and $k \geq 1$ is a parameter of choice. Moreover, for the special case of $f = 2$, they give an ad-hoc solution, i.e., a 3-SSDO that can be built in $O(nm + n^2\log n)$ time, has size $O(n\log n)$ and constant query time.

In the past, several other research efforts have been devoted to *all-pairs distance oracles* (APDO) tolerating single/multiple edge/vertex failures. Quite interestingly, here $\widetilde{O}(n^2)$-size exact-distance sensitivity oracles are instead known, as opposed to the $\Omega(n^2)$ lower bound for the single-source case. More precisely, in [4] the authors built (on directed graphs) in $\widetilde{O}(mn)$ time a 1-E/VFT 1-APDO of size $\widetilde{O}(n^2)$ and with query time $O(1)$. For two failures, in [11] the authors built, still on directed graphs, a 2-E/VFT 1-APDO of size $\widetilde{O}(n^2)$ and with query time $O(\log n)$. Concerning multiple-edge failures, in [7] the authors built, for any integer $k \geq 1$, an f-EFT $(8k-2)(f+1)$-APDO of size $O(fk\,n^{1+1/k}\log(nW))$, where W is the ratio of the maximum to the minimum edge weight in G, and with a query time of $\widetilde{O}(|F|\,\log\log d)$, where F is the actual set of failing edges, and d is the distance between the queried pair of nodes in $G - F$.

As we said before, the natural counterpart of distance sensitivity oracles are the fault-tolerant spanners. Due to space limitations, for this related topic we refer the reader to the discussion and the references provided in [6]. However, it is worth mentioning that there is a line of papers on EFT ASPTs [14, 15, 16, 17], that as we said are very close in spirit to EFT SSDOs.

Finally, we mention that there is a large body of literature concerned with the design of ordinary (i.e., fault-free) distance oracles, and an extensive recent survey on the topic is given in [18].

1.4 Notation

For two given vertices x and y of an edge weighted graph H, we denote by $\pi_H(x,y)$ a shortest path between x and y in H and we denote by $d_H(x,y)$ the total length of $\pi_H(x,y)$. For two given paths P and P' such that P is a path between x and y and P' is a path between y and z, we denote by $P \circ P'$ the path from x to z obtained by concatening P and P'.

Let T be an SPT of G rooted at s, and let $e = (u,v)$ be an edge of T. In the rest of the paper, we always assume that u is closer to s than v w.r.t. the number of hops in T. Furthermore, we denote by T_v the subtree of T rooted at v. Finally, for a vertex $t \in T_v$, we denote by $A(t,e) = V(\pi_T(v,t))$ the set of *living ancestors* of T, t included, contained in T_v.

2 The EFT 2-SSDO

In this section we describe our EFT 2-SSDO with linear size and constant query time. Some of the ideas we develop here will be used in the next section, where we provide our main result.

For the rest of the paper, let T be a fixed SPT of G rooted at s that is stored in our distance oracle. First of all, observe that if there is no edge failure or the edge that has failed is not contained in T, then, for any vertex t, our distance oracle can return the (exact) distance value $d_T(s,t)$ in constant time. This is the case also when the edge $e=(u,v)$ that has failed is contained in T, but t is not a vertex of T_v. Therefore, in the rest of this section, we describe only how our distance oracle computes an approximate distance from s to t in $G-e$ when the edge $e=(u,v)$ that has failed is contained in T and the vertex t is contained in the subtree T_v.

The following lemma describes a simple but still interesting property that we exploit as key ingredient in our oracle. Let $e=(u,v)$ be a failing edge, we define a special replacement path from s to t as follows: $P_e(t)=\pi_{G-e}(s,v)\circ\pi_G(v,t)$.

▶ **Lemma 1.** *Let $e=(u,v)$ be a failing edge and $t\in V(T_v)$. At least one of the following conditions holds: (i) $d_{G-e}(s,t)\le w(P_e(t))\le 2d_{G-e}(s,t)$, (ii) $d_{G-e}(s,t)<2d_G(s,t)$.*

Proof. We assume that (ii) is false (i.e., $d_{G-e}(s,t)\ge 2d_G(s,t)$) and we prove that (i) must hold. Indeed:

$$\begin{aligned}d_{G-e}(s,t)\le w(P_e(t)) &= d_{G-e}(s,v)+d_G(v,t)\le d_{G-e}(s,t)+d_{G-e}(v,t)+d_G(v,t)\\ &= d_{G-e}(s,t)+2d_G(v,t)\le d_{G-e}(s,t)+2d_G(s,t)\le 2d_{G-e}(s,t).\end{aligned}$$ ◀

Notice that the length of $P_e(t)$ is available in constant time once we store $O(n)$ distance values, namely $d_{G-e}(s,v)$ for each $e=(u,v)\in E(T)$. Hence, the challenge here is to understand when $w(P_e(t))$ provides a 2-approximation of the distance $d_{G-e}(s,t)$ and when we can instead return the value $2d_G(s,t)\le 2d_{G-e}(s,t)$ (observe that $2d_G(s,t)$ could be in general smaller than $d_{G-e}(s,t)$). The idea of our oracle is that of selecting a subset of *marked* vertices for which this information can be stored and retrieved efficiently and from which we can derive the same information for the other nodes.

To this aim, we now describe an algorithm that preprocesses the graph and collects compact information that we will use later to efficiently answer distance queries. Consider the edges of T as traversed by a preorder visit from s. We define a total order relation $\prec$ on $E(T)$ as follows: we say that $e'\prec e''$ iff e' is traversed before e''. We also use $e'\preceq e''$ to denote that either $e'\prec e''$ or $e'=e''$.

Algorithm 1 considers the failing edges $e\in E(T)$ in preorder and computes a label $\ell(v)$ for each vertex $v\in V(G)$. This value will be either ∞ or a suitable edge $e\in E(T)$. Here we treat ∞ as a special label that satisfies $e'\prec\infty$ for every edge $e'\in E(T)$. We say that v is *marked* if $\ell(v)\ne\infty$, and we say that v is *marked at time* e if $\ell(v)\preceq e$. Intuitively, $\ell(v)$ is the time at which v first becomes marked.

More precisely, for each failing edge e, Algorithm 1, marks a vertex $t\in V(T_v)$ (at time e) iff vertex t fails two tests: the *distance test* and the *ancestor test*. In the distance test we check whether the path $P_e(t)$ suffices to provide a 2-stretched distance to t, while in the ancestor test we check whether a living ancestor of t has already been marked. Notice that the ancestor test guarantees that each vertex t is marked at most once during the whole execution of the algorithm (since $t\in A(t,e)$ by definition).

As a simple consequence of the above algorithm, we have:

Algorithm 1: Mark-up algorithm

```
1  for $v \in V$ do
2      $\ell(v) \leftarrow \infty$
3  for $e = (u, v) \in E(T)$ in preorder w.r.t. $T$ do
4      for $t \in V(T_v)$ in preorder w.r.t. $T$ do
5          if $w(P_e(t)) \leq 2d_{G-e}(s, t)$ then                // Distance test
6              do nothing
7          else if $\exists z \in A(t, e) : \ell(z) \neq \infty$ then      // Ancestor test
8              do nothing
9          else                                            // Both tests failed
10             $\ell(t) \leftarrow e$                                 // Mark $t$ at time $e$
```

▶ **Lemma 2.** *Let $e \in E(T)$ be a failing edge and let t be a vertex such that $\ell(t) = e$, we have $d_{G-e}(s, t) < 2d_G(s, t)$.*

Proof. Since t is first marked at time e, it must have failed the distance test, i.e., $w(P_e(t)) > 2d_{G-e}(s, t)$. This means that condition (i) of Lemma 1 is false and hence condition (ii) must hold. ◀

Another useful property of the marked vertices is the following:

▶ **Lemma 3.** *Let $e \in E(T)$ be a failing edge and let t be a vertex such that $\ell(t) = e$, then $\pi_{G-e}(s, t)$ and $\pi_T(v, t)$ are edge disjoint.*

Proof. Let $e = (u, v)$ and assume by contradiction that $\pi_{G-e}(s, t)$ and $\pi_T(v, t)$ are not edge disjoint. Let (z, z') be an edge belonging to both paths, with z closer to v than z'. Notice that both z and z' are living ancestors of t, and that $z \neq t$.

Since t is first marked at time e, it must have failed the ancestor test. This implies that no other living ancestor of t is marked at time e. Moreover, as z is visited by the algorithm before t, it must have failed the ancestor test as well. Since z it is not marked at time e, it follows that it must have passed the distance test, i.e., $w(P_e(z)) \leq 2d_{G-e}(s, z)$. We have $P_e(t) = P_e(z) \circ \pi_G(z, t)$ and hence:

$$\begin{aligned} w(P_e(t)) = w(P_e(z)) + d_G(z, t) &\leq 2d_{G-e}(s, z) + d_G(z, t) \\ &\leq 2d_{G-e}(s, z) + 2d_{G-e}(z, t) = 2d_{G-e}(s, t) \end{aligned}$$

which implies that t has passed the distance test and contradicts the hypothesis $\ell(t) = e$. ◀

The next lemma is the last ingredient of our oracle, and allows to distinguish the two cases of Lemma 1.

▶ **Lemma 4.** *Let $e = (u, v) \in E(T)$ be a failing edge and let $t \in V(T_v)$. If there exists $z \in A(t, e)$ such that $\ell(z) \preceq e$, then $d_{G-e}(s, t) \leq 2d_G(s, t)$. If no such vertex z exists, then $d_{G-e}(s, t) \leq w(P_e(t)) \leq 2d_{G-e}(s, t)$.*

Proof. Let z be any vertex in $A(t, e)$ such that $\ell(z) \preceq e$, and let $e' = \ell(z)$. By the definition of living ancestor and by Lemma 3 we have that $\pi_{G-e'}(s, z)$ does not use the edge e (see

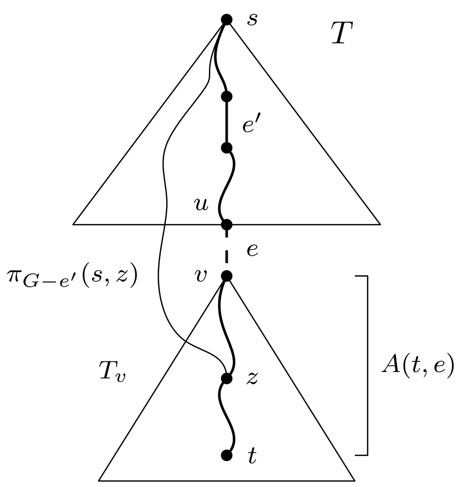


Figure 1 Representation of the proof of Lemma 4. The shortest path between s and t in T is shown in bold while the failing edge e is dashed. Notice that the path $\pi_{G-e'}(s,z)$ is edge disjoint from the path $\pi_T(v,z)$.

Figure 1). Since z is marked at time e' we have $d_{G-e'}(s,z) < 2d_G(s,z)$ (see Lemma 2). Thus, we have that $d_{G-e}(s,z) \le w(\pi_{G-e'}(s,z)) = d_{G-e'}(s,z) < 2d_G(s,z)$. Therefore:

$$d_{G-e}(s,t) \le d_{G-e}(s,z) + d_G(z,t) \le 2d_G(s,z) + d_G(z,t) \le 2d_G(s,z) + 2d_G(z,t) = 2d_G(s,t).$$

If no such vertex z exists, then when Algorithm 1 considered edge e, the vertex z failed the ancestor test. Since t is not marked at time e (as otherwise we could choose $z = t$) it must have passed the distance test, i.e., $w(P_e(t)) \le 2d_{G-e}(s,t)$. ◀

This latter lemma is exactly what we need in order to implement the query operation of our oracle. When edge $e = (u,v)$ is failing and we are queried for the distance of a vertex t, we first test whether $e \in E(T)$ and $t \in V(T_v)$: if the test fails we return the original distance $d_G(s,t)$.[1] If the test succeeds, we look for a vertex $z \in A(t,e)$ such that $\ell(z) \preceq e$. If such a vertex exists we return $2d_G(s,t)$, otherwise we return $w(P_e(t))$. Observe that in both cases we return a feasible 2-approximation of the distance $d_{G-e}(s,t)$.

In the following we will show how it is possible to determine in constant time whether such a vertex z exists. More precisely we only need to look for a vertex $x \in A(t,e)$ minimizing $\ell(x)$. If such a vertex satisfies $\ell(x) \preceq e$ then $z = x$ and we are done. On the converse, if $e \prec \ell(x)$, then we know that no vertex $z \in A(t,e)$ with $\ell(z) \preceq e$ can exist.

To this aim, we use a data structure for the *bottleneck vertex query* problem on trees (`BVQ` for short). In the `BVQ` problem we want to preprocess a vertex-weighted tree $\mathcal{T}$ in order to answer queries of this form: given two vertices $x, y \in V(\mathcal{T})$ report the lightest vertex on the (unique) path between x and y in $\mathcal{T}$. In [9], the authors show how to build, in $O(|V(\mathcal{T})| \log |V(\mathcal{T})|)$ time, a data structure having linear size and constant query time.[2]

[1] To see whether t is contained in $V(T_v)$ or not, it suffices to check whether the least common ancestor of t and v in T corresponds to v or not. The least common ancestor between any pair of vertices of a tree can be computed in constant time after a linear time preprocessing [13].

[2] Actually, in [9] the *bottleneck edge query* (`BEQ`) problem is considered instead. However it is easy to see that the `BEQ` and the problems `BVQ` are essentially equivalent.

In our preprocessing, we build such a structure on the tree T where each vertex $x \in T$ weighs $\ell(x)$, and then we use it to locate x in the path between v and t whenever we need to report an approximate distance for $d_{G-(u,v)}(s,t)$.

We are now ready to state the main result of this section.

▶ **Theorem 5.** *Let G be a non-negatively real weighted and undirected n-vertex and m-edge graph, and let s be a source node. There exists an EFT 2-SSDO that has size $O(n)$ and constant query time, and that can be constructed using $O(mn + n^2 \log n)$ time and $O(m)$ space.*

Proof. As we already discussed it is easy to answer a query in constant time once we store: (i) the SPT T of G w.r.t. s, (ii) the label $\ell(v)$ for each v, (iii) the value $w(\pi_{G-e}(s,v))$ for each $(u,v) \in E(T)$, and (iv) a data structure for the `BVQ` problem. The total space used is hence $O(n)$.

Concerning the time and the space used by Algorithm 1, observe that for each edge $e = (u,v)$, we can compute an SPT of $G - e$ with source s in $O(m + n \log n)$ time and $O(m)$ space. Therefore, for each t the distance test can be accomplished in $O(1)$ time. It remains to show that also the ancestor test can be done in constant time. To this aim, it is sufficient to maintain for each vertex x the (current) number ν_x of marked ancestors of x in T, and check whether $\nu_t - \nu_u > 0$. The maintenance of these values can be clearly done with constant time and space overhead, from which the claim follows. ◀

3 The EFT $(1+\varepsilon)$-SSDO

In this section we describe our main result, namely how to build, given any $0 < \varepsilon < 1$, an EFT $(1+\varepsilon)$-SSDO having $O\left(n \cdot \frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$ size and $O\left(\log n \cdot \frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$ query time.

Our distance oracle stores a set of $O\left(n \cdot \frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$ (exact) distance values that are computed by a preprocessing algorithm that we describe below. From a high-level point of view, we follow the same approach used in the previous section, but here a vertex t can be marked several times, each corresponding to a specific failing edge $e = (u,v) \in E(T)$ for which the algorithm computes the shortest path $\pi_{G-e}(s,t)$ that is edge disjoint from $\pi_T(v,t)$. We will show that such paths have strictly decreasing lengths and that they are $O\left(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$ in number. We will store all these distance values and we will show that they can be used to efficiently answer any distance query by suitably combining them with distances in T.

More precisely, for every $e = (u,v) \in E(T)$ and every $t \in V(T_v)$, the preprocessing algorithm computes a value $\texttt{dist}(t,e)$ that satisfies $d_{G-e}(s,t) \le \texttt{dist}(t,e) \le \sqrt{1+\varepsilon} \cdot d_{G-e}(s,t)$. Furthermore, each value $\texttt{dist}(t,e)$ represents the total length of a path P from s to t in $G - e$, whose structure can be either of the following two types:

type 1: $P = \pi_{G-e}(s,t)$;

type 2: P can be decomposed into $\pi_{G-e'}(s,z)$, for some e' and z such that $\texttt{dist}(z,e') = d_{G-e'}(s,z)$, and $\pi_T(z,t)$ (possibly, either $e = e'$ or $z = t$).

Since each path of type 2 can be easily derived by combining a path of type 1 with a path in T, our oracle stores only all the values $\texttt{dist}(t,e) = d_{G-e}(s,t)$ that represent paths of type 1. In the next two subsections, we will show that, for every $e \in E(T)$ and every t, our distance oracle can compute a $(\sqrt{1+\varepsilon})$-approximation of $\texttt{dist}(t,e)$ in $O\left(\log n \cdot \frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$ time.

3.1 The preprocessing algorithm

The preprocessing algorithm (see the pseudocode of Algorithm 2) visits all the edges of T in preorder and, for each $e = (u, v) \in E(T)$, it visits all the vertices of T_v in preorder. For the rest of this section, unless stated otherwise, let $e = (u, v)$ be a fixed edge of T that is visited by the algorithm. The algorithm sets $\texttt{dist}(v, e) = d_{G-e}(s, v)$, i.e., $\texttt{dist}(v, e)$ always represents a path of type 1. When the algorithm visits t, with $t \neq v$, it first checks whether the shortest, among several paths from s to t in $G - e$ of type 2, has a total length of at most $\sqrt{1+\varepsilon} \cdot d_{G-e}(s, t)$. If this is the case, then the algorithm sets $\texttt{dist}(t, e)$ equal to the total length of such a path, otherwise it sets $\texttt{dist}(t, e) = d_{G-e}(s, t)$, i.e., $\texttt{dist}(t, e)$ represents a path of type 1. The preprocessing algorithm returns the set of all distance values that represent the paths of type 1.

For each vertex t, the algorithm stores the total length of the last path from s to t of type 1 that has computed in the variable $\texttt{last}(t)$.

Algorithm 2: Selects paths of type 1 whose lengths are stored in the oracle.

```
  // Initialization of variables
1 $S, S' = \emptyset$ for every $t \in V(G)$ do
2     $\texttt{last}(t) = \infty$

  // All the values $\texttt{dist}(t, e)$ are computed
3 for every $e = (u, v) \in E(T)$ in preorder w.r.t. $T$ do
4     $\texttt{last}(v), \texttt{dist}(v, e) = d_{G-e}(s, v)$; add $d_{G-e}(s, v)$ to $S'$        // path of type 1
5     for every $t \in V(T_v) \setminus \{v\}$ in preorder w.r.t. $T$ do
          // The length of a path from $s$ to $t$ in $G - e$ of type 2 is computed
6         $\texttt{dist}(t, e) = \min\{\texttt{last}(z) + d_T(z, t) \mid z \in A(t, e)\}$
7         if $\texttt{dist}(t, e) > \sqrt{1+\varepsilon} \cdot d_{G-e}(s, t)$ then
8             $\texttt{last}(t), \texttt{dist}(t, e) = d_{G-e}(s, t)$; add $d_{G-e}(s, t)$ to $S$   // path of type 1

9 return $S$ and $S'$.
```

For the rest of this section, unless stated otherwise, let t be a fixed vertex of T_v that is visited by the algorithm. The proof of the following proposition is trivial.

▶ **Proposition 6.** *At the end of the visit of t, $\texttt{dist}(t, e) \leq \sqrt{1+\varepsilon} \cdot d_{G-e}(s, t)$.*

The following lemma is similar to Lemma 3 and it is useful to prove that $\texttt{dist}(t, e) \geq d_{G-e}(s, t)$.

▶ **Lemma 7.** *If $d_{G-e}(s, t)$ is added to $S \cup S'$, then $\pi_{G-e}(s, t)$ and $\pi_T(v, t)$ are edge disjoint.*

Proof. The claim trivially holds when $t = v$ since $\pi_T(v, v)$ contains no edge. Therefore, we assume that $t \neq v$. We prove the claim by contradiction by showing that if $\pi_{G-e}(s, t)$ and $\pi_T(v, t)$ were not edge disjoint, then the algorithm would not add $d_{G-e}(s, t)$ to $S \cup S'$. So, we assume that $\pi_{G-e}(s, t)$ and $\pi_T(v, t)$ are not edge disjoint. Let t' be, among the vertices that are contained in both $\pi_{G-e}(s, t)$ and $\pi_T(v, t)$, the one that is closest to v w.r.t. the number of hops in $\pi_T(v, t)$. Clearly, $t' \neq t$ and $\pi_T(t', t)$ is a shortest path from t' to t in G as well as in $G - e$. Thus, by the suboptimality property of shortest paths, $d_{G-e}(s, t) = d_{G-e}(s, t') + d_{G-e}(t', t) = d_{G-e}(s, t') + d_T(t', t)$. Let $z \in A(t', e)$ be the vertex such that $\texttt{dist}(t', e) = \texttt{last}(z) + d_T(z, t')$ (possibly $z = t'$). As the algorithm visits t' before visiting t, by Proposition 6, $\texttt{dist}(t',e) \le \sqrt{1+\varepsilon} \cdot d_{G-e}(s,t')$ at the beginning of the visit of t. Therefore

$$\begin{aligned}\texttt{last}(z) + d_T(z,t) &= \texttt{last}(z) + d_T(z,t') + d_T(t',t)\\ &= \texttt{dist}(t',e) + d_T(t',t)\\ &\le \sqrt{1+\varepsilon} \cdot d_{G-e}(s,t') + d_T(t',t)\\ &\le \sqrt{1+\varepsilon} \cdot d_{G-e}(s,t).\end{aligned}$$

As $\texttt{dist}(t,e) \le \texttt{last}(z) + d_T(z,t)$ already before the execution of the if statement during the visit of t, the algorithm never adds $d_{G-e}(s,t)$ to $S \cup S'$. The claim follows. ◀

We now prove that $\texttt{dist}(t,e) \ge d_{G-e}(s,t)$.

▶ **Lemma 8.** *At the end of the visit of t, $\texttt{dist}(t,e) \ge d_{G-e}(s,t)$.*

Proof. The claim trivially holds if the algorithm sets $\texttt{dist}(t,e) = d_{G-e}(s,t)$. Therefore, we need to prove the claim when the condition of the if statement during the visit of t is not satisfied, i.e., $\texttt{dist}(t,e) = \texttt{last}(z) + d_T(z,t)$, for some vertex $z \in A(t,e)$ (possibly, $z = t$). Let $\texttt{last}(z) = d_{G-e'}(s,z)$, for some $e' = (u',v')$ such that z is a vertex of $T_{v'}$ (possibly $e' = e$). We divide the proof into the following two cases according to whether $e' = e$ or not.

Consider the case in which $e' = e$ and observe that e is not contained in $\pi_T(z,t)$. Therefore $\texttt{dist}(t,e) = d_{G-e'}(s,z) + d_T(z,t) = d_{G-e}(s,z) + d_{G-e}(z,t) \ge d_{G-e}(s,t)$.

Consider the case in which $e' \ne e$ and observe that e is an edge of the path $\pi_T(v',z)$. Furthermore, $\texttt{last}(z) = d_{G-e'}(s,z)$ implies that the algorithm has added $d_{G-e'}(s,z)$ to $S \cup S'$. Therefore, by Lemma 7, $\pi_{G-e'}(s,z)$ and $\pi_T(v',z)$ are edge disjoint. This implies that e is contained neither in $\pi_{G-e'}(s,z)$ nor in $\pi_T(z,t)$. Therefore, $d_{G-e}(s,t) \le d_{G-e'}(s,z) + d_T(z,t) = \texttt{last}(z) + d_T(z,t) = \texttt{dist}(t,e)$, and the claim follows. ◀

The following proposition allows us to prove that the number of paths of type 1 computed by the algorithm is almost linear in n.

▶ **Proposition 9.** *Let $e_0, e_1, \dots, e_k$ be all the pairwise distinct edges of T, in the order in which they are visited by the algorithm, such that $d_{G-e_i}(s,t) \in S$. Then, for every $i = 0,1,\dots,k$, $d_{G-e_i}(s,t) < 2/\big((\sqrt{1+\varepsilon}-1)(1+\varepsilon)^{i/2}\big) d_G(s,t)$. Furthermore, $k < 2 \cdot \frac{\log\left(2/(\sqrt{1+\varepsilon}-1)\right)}{\log(1+\varepsilon)}$.*

Proof. Let $e_0 = (u_0,v_0)$ and observe that at the end of the visit of e_0 and v_0

$$\begin{aligned}\texttt{last}(v_0) + d_T(v_0,t) &= d_{G-e_0}(s,v_0) + d_T(v_0,t)\\ &\le d_{G-e_0}(s,t) + d_T(t,v_0) + d_T(v_0,t)\\ &\le d_{G-e_0}(s,t) + 2d_T(s,t)\\ &= d_{G-e_0}(s,t) + 2d_G(s,t).\end{aligned}$$

Since $d_{G-e_0}(s,t) \in S$, $\sqrt{1+\varepsilon} \cdot d_{G-e_0}(s,t) < \texttt{last}(v_0) + d_T(v_0,t)$, and therefore

$$d_{G-e_0}(s,t) < \frac{2}{\sqrt{1+\varepsilon}-1} d_G(s,t). \qquad (1)$$

Next, observe that the value $\texttt{last}(t)$ at the beginning of the visit of edge e_i, with $1 \le i \le k$, is equal to $d_{G-e_{i-1}}(s,t)$. Since $d_{G-e_i}(s,t) \in S$, we have that

$$\sqrt{1+\varepsilon} \cdot d_{G-e_i}(s,t) < d_{G-e_{i-1}}(s,t) \qquad \text{for every } i = 1,\dots,k. \qquad (2)$$

Thus, if, for any $i > 0$, we combine inequality (1) and all the inequalities (2) with $j \le i$, we obtain $(1+\varepsilon)^{i/2} d_{G-e_i}(s,t) < 2/(\sqrt{1+\varepsilon}-1) d_G(s,t)$, i.e.,

$$d_{G-e_i}(s,t) < \frac{2}{(\sqrt{1+\varepsilon}-1)(1+\varepsilon)^{i/2}} d_G(s,t).$$

Moreover, using $d_G(s,t) \le d_{G-e_k}(s,t)$ in $d_{G-e_k}(s,t) < 2/\big((\sqrt{1+\varepsilon}-1)(1+\varepsilon)^{k/2}\big) d_G(s,t)$ we obtain $(1+\varepsilon)^{k/2} < 2/(\sqrt{1+\varepsilon}-1)$, i.e.,

$$k < 2 \cdot \frac{\log\big(2/(\sqrt{1+\varepsilon}-1)\big)}{\log(1+\varepsilon)}.$$

The claim follows. ◀

Observe that $\log\big(2/(\sqrt{1+\varepsilon}-1)\big) = O(\log(1/\varepsilon))$, and that $\log(1+\varepsilon) = \Theta(\varepsilon)$. Therefore, using Proposition 9 and the fact that $|S'| = n-1$, we obtain

▶ **Corollary 10.** $|S \cup S'| = O\left(n \cdot \frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$.

▶ **Lemma 11.** *Algorithm 2 can be implemented to run in $O(mn + n^2 \log n)$ time and $O\left(m + n \cdot \frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$ space.*

Proof. First we prove the time bound. Clearly, the inizialization of variables takes $O(n)$ time. Let $e = (u,v)$ be an edge that is visited by the algorithm. The algorithm computes an SPT of $G - e$ rooted at s in $O(m + n\log n)$ time. Let $t \ne v$ be the vertex that is going to be visited by the algorithm and let t' be the parent of t in T. Observe that

$$\begin{aligned}\min_{z\in A(t,e)}\{\texttt{last}(z) + d_T(z,t)\} &= \min\left\{\texttt{last}(t), \min_{z\in A(t',e)}\{\texttt{last}(z) + d_T(z,t)\}\right\}\\ &= \min\left\{\texttt{last}(t), \min_{z\in A(t',e)}\{\texttt{last}(z) + d_T(z,t')\} + w(t',t)\right\} \qquad (3)\\ &= \min\{\texttt{last}(t), \texttt{dist}(t',e) + w(t',t)\},\end{aligned}$$

Therefore, each value $\texttt{dist}(t,e)$ can be computed in constant time rather than in $O(n)$ time. Hence, the overall running time is $O(mn + n^2 \log n)$.

Concerning the space complexity, observe that, from Equation (3), the algorithm does not need to store all the values $\texttt{dist}(t,e)$ but, for each t, it is enough to remember the last computed value $\texttt{dist}(t,e)$. This can be clearly done with an array of n elements. Next, observe that, during the visit of e, the algorithm only needs the one-to-all distances in $G - e$. This implies that there is no need to keep all the $n-1$ SPT's of $G - e$, for every $e \in E(T)$, at the same time and therefore, all these SPT's can share the same $O(n)$ space. Finally, $|S \cup S'| = O\left(n \cdot \frac{1}{\varepsilon}\log\frac{1}{\varepsilon}\right)$ by Corollary 10. The claim follows. ◀

3.2 The data structure

We now describe how the values in S and S' can be organized in a data structure of size $O\left(n \cdot \frac{1}{\varepsilon}\log\frac{1}{\varepsilon}\right)$ so that our distance oracle can compute a $(\sqrt{1+\varepsilon})$-approximation of $\texttt{dist}(t,e)$ in $O\left(\log n \cdot \frac{1}{\varepsilon}\log\frac{1}{\varepsilon}\right)$ time.

Remind that we say that $e' \prec e''$ if the preprocessing algorithm has visited e' before visiting e'', and that we also use $e' \preceq e''$ to denote that either $e' \prec e''$ or $e' = e''$. Let $k = \left\lfloor 2 \cdot \frac{\log\left(2/(\sqrt{1+\varepsilon}-1)\right)}{\log(1+\varepsilon)} \right\rfloor$ and let $a_i = \frac{2}{(\sqrt{1+\varepsilon}-1)(1+\varepsilon)^{i/2}}$. Finally, for every $i = 0, 1, \ldots, k$, let

$$S_i = \Big\{d_{G-e'}(s,z) \in S \mid a_{i+1} \cdot d_G(s,z) \le d_{G-e'}(s,z) < a_i \cdot d_G(s,z)\Big\}.$$

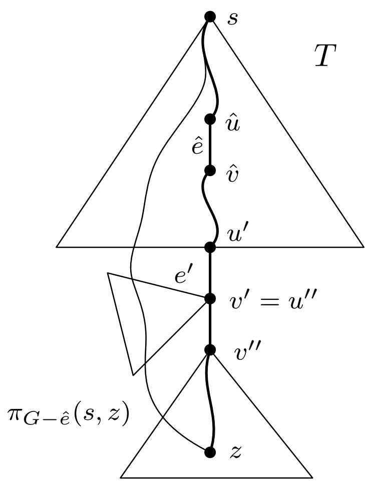


Figure 2 Representation of the proof of Proposition 12. The shortest path between s and t in T is shown. Notice that the path $\pi_{G-\hat{e}}(s,z)$ is edge disjoint from the path $\pi_T(\hat{v},z)$.

By Proposition 9, we have that $\{S_i \mid i = 0,1,\ldots,k\}$ is a partition of S.

We maintain a set of $k+1$ trees $\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_k$, one for each S_i. Each tree $\mathcal{T}_i$ is a copy of T, where each vertex z, such that $d_{G-e'}(s,z) \in S_i$, has a label $\ell_i(z) = e'$. Every other vertex $z \in V(G) \setminus S_i$ has a label $\ell_i(z) = \infty$ such that $e' \prec \infty$, for every edge $e' \in E(T)$.

In the following, we denote the value of $\texttt{last}(z)$ at the end of the visit of edge e' by $\texttt{last}(z,e')$. First of all, we prove the following proposition.

▶ **Proposition 12.** *If* $e' \preceq e''$*, then* $\texttt{last}(z,e'') \leq \texttt{last}(z,e')$.

Proof. Let $e' = (u',v')$ and $e'' = (u'',v'')$. Notice that the claim can be proved by showing that it holds under the assumption that $v' = u''$. Furthermore, we can also assume that $\texttt{last}(z,e') \neq \infty$ as well as $\texttt{last}(z,e'') \neq \texttt{last}(z,e')$, otherwise the claim would be trivially true. This last assumption together with $v' = u''$ imply that $\texttt{last}(z,e'') = d_{G-e''}(s,z)$. Let $\texttt{last}(z,e') = d_{G-\hat{e}}(s,z)$, for some $\hat{e} \preceq e'$, with $\hat{e} = (\hat{u},\hat{v})$. Clearly, $d_{G-\hat{e}}(s,z) \in S \cup S'$. Therefore, by Lemma 7, $\pi_{G-\hat{e}}(s,z)$ and $\pi_T(\hat{v},z)$ are edge disjoint (see Figure 2). Since e'' is an edge of $\pi_T(\hat{v},z)$, $\pi_{G-\hat{e}}(s,z)$ is also a path from s to t in $G-e''$ and therefore $\texttt{last}(z,e'') = d_{G-e''}(s,z) \leq d_{G-\hat{e}}(s,z) = \texttt{last}(z,e')$. ◀

Let $e = (u,v) \in E(T)$ and let t be a vertex of T_v. Using Proposition 12, we have that either $\texttt{dist}(t,e) = \texttt{last}(v,e) + d_T(v,t) = d_{G-e}(s,v) + d_T(v,t)$, or

$$\begin{aligned}
&\texttt{dist}(t,e) = \min\left\{\texttt{last}(z,e) + d_T(z,t) \mid z \in A(t,e) \setminus \{v\}\right\}\\
&= \min\left\{\texttt{last}(z,e) + d_T(z,t) \mid z \in A(t,e)\right\}\\
&= \min_{i=0,1,\ldots,k}\left\{\min\left\{\texttt{last}(z,\ell_i(z)) + d_T(z,t) \mid z \in A(t,e) \wedge \ell_i(z) \preceq e\right\}\right\}\\
&= \min_{i=0,1,\ldots,k}\left\{\delta_i := \min\left\{d_{G-e'}(s,z) + d_T(z,t) \mid z \in A(t,e) \wedge d_{G-e'}(s,z) \in S_i \wedge e' \preceq e\right\}\right\}.
\end{aligned}$$

In the former case, $\texttt{dist}(t,e)$ is available in $O(1)$ time, since $d_{G-e}(s,v)$ is stored in S'. In the latter case, we now show how to compute, for any fixed $i = 0,1,\ldots,k$, a $(\sqrt{1+\varepsilon})$-approximate upper bound to δ_i in $O(\log n)$ time. Using Proposition 9, this will imply that our oracle is able to answer a query in $O\left(\log n \cdot \frac{1}{\varepsilon}\log\frac{1}{\varepsilon}\right)$ time.

First of all, we prove that the labels of each $\mathcal{T}_i$ satisfy a nice property.

▶ **Lemma 13.** *Let z' and z'' be two distinct vertices of $A(t,e)$ such that z'' is a proper ancestor of z' and $\ell_i(z') = e'$ and $\ell_i(z'') = e''$, for some edges $e', e'' \in E(T)$, with $e', e'' \preceq e$ (possibly, $e' = e''$). We have that $d_{G-e''}(s,z'') + d_T(z'',t) \leq \sqrt{1+\varepsilon} \cdot \big(d_{G-e'}(s,z') + d_T(z',t)\big)$.*

Proof. Since $d_{G-e''}(s,z'') \in S_i$, we have that $d_{G-e''}(s,z'') < a_i \cdot d_G(s,z'')$. Furthermore, $d_{G-e'}(s,z') \in S_i$ implies that $d_{G-e'}(s,z') \geq a_{i+1} \cdot d_G(s,z') = a_i/\sqrt{1+\varepsilon} \cdot d_G(s,z')$. As a consequence, $d_{G-e''}(s,z'') + d_T(z'',t) < a_i \cdot d_G(s,z'') + d_T(z'',z') + d_T(z',t) \leq a_i \cdot d_G(s,z') + d_T(z',t) \leq \sqrt{1+\varepsilon} \cdot d_{G-e'}(s,z') + d_T(z',t) \leq \sqrt{1+\varepsilon} \cdot \big(d_{G-e'}(s,z') + d_T(z',t)\big)$. ◀

Let $z'' \in A(t,e)$ be the vertex closest to v w.r.t. T such that $\ell_i(z) = e'' \preceq e$, if such a vertex exist. Let $\delta_i = d_{G-e'}(s,z') + d_T(z',t)$, for some e' and z' such that $z' \in A(t,e)$, $d_{G-e'}(s,z') \in S_i$, and $e' \preceq e$. Observe that $d_{G-e''}(s,z'') + d_T(z'',t) \geq \delta_i$. Moreover, since z' and z'' satisfy all the hyphotesis of Lemma 13, we have that

$$\delta_i \leq d_{G-e''}(s,z'') + d_T(z'',t) \leq \sqrt{1+\varepsilon} \cdot \delta_i.$$

Therefore, the value $d_{G-e''}(s,z'') + d_T(z'',t)$ is a $(\sqrt{1+\varepsilon})$-approximate upper bound to the value δ_i. Now we show how the vertex z'' can be computed in $O(\log n)$ time.

To this aim, we preprocess each tree $\mathcal{T}_i$ in order to build a linear-size data structure that answers `BVQ` queries in constant time. This can be done in $O(n \log n)$ time per tree. We also preprocess T so we are able to perform *level-ancestor* queries in constant time. The size needed by this latter data structure is $O(n)$ and it can be built in linear-time [3, 2]. In a level ancestor query, we are given a vertex $x \in V(T)$ and a positive integer h, and we ask for the ancestor y of x such that $\pi_T(x,y)$ contains exactly h edges. We can then find z'' by performing a binary search over the vertices of $A(t,e)$, as follows.

Let $e = (u,v)$, we perform a level ancestor query on T to find the vertex x of $\pi_T(v,t)$ that divides the path into roughly two halves. Let x' be the parent of x, and let y and y' be the vertices of $\pi_T(x,t)$ and $\pi_T(v,x')$ of minimum labels, respectively. Notice that y and y' can be found in constant time by performing two `BVQ` queries on $\mathcal{T}_i$. If $\ell_i(y') \preceq e$, then we remember y' as the best vertex found so far and we iterate the binary search in $\pi_T(v,x')$. Otherwise, if $e \prec \ell_i(y')$, then we compare $\ell_i(y)$ and e. If $\ell_i(y) \preceq e$, then we remember y as the best vertex found so far and we iterate the binary search in $\pi_T(x,t)$. If $e \prec \ell_i(y)$, then we can complete our binary search and return the best vertex found, if any.

We have then proven the following:

▶ **Theorem 14.** *Let G be a non-negatively real weighted and undirected n-vertex and m-edge graph, and let s be a source node. For any arbitrarily small $0 < \varepsilon < 1$, there exists an EFT $(1+\varepsilon)$-SSDO that has size $O\left(n \cdot \frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$ and $O\left(\log n \cdot \frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$ query time, and that can be constructed using $O(mn + n^2 \log n)$ time and $O\left(m + n \cdot \frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$ space.*

4 Lower bounds on the size of additive EFT ASPT and SSDO

In this section, we give a lower bound on the bit size of an EFT $\beta(d)$-additive SSDO. Recall that after the failure of any edge, such an oracle must return an estimation d' of the actual distance d between s and any node such that $d \leq d' \leq d + \beta(d)$, where β is any positive real function. Due to space limitations, the proof of next theorem is omitted and will be given in the full version of the paper.

▶ **Theorem 15.** *Let $\beta(d) = kd^{1-\delta}$, for arbitrary $k \geq 1$ and $0 < \delta \leq 1$. Then, there exist classes of polynomially weighted graphs with n nodes such that:*

1. *any* EFT *$\beta(d)$-additive ASPT has $\Omega(n^2)$ edges;*
2. *any* EFT *$\beta(d)$-additive SSDO has $\Omega(n^2)$ bit size for at least an input graph, regardless of its query time.*

References

1 Surender Baswana and Neelesh Khanna. Approximate shortest paths avoiding a failed vertex: Near optimal data structures for undirected unweighted graphs. *Algorithmica*, 66(1):18–50, 2013. doi:10.1007/s00453-012-9621-y.

2 Michael A. Bender and Martin Farach-Colton. The level ancestor problem simplified. *Theor. Comput. Sci.*, 321(1):5–12, 2004. doi:10.1016/j.tcs.2003.05.002.

3 Omer Berkman and Uzi Vishkin. Finding level-ancestors in trees. *J. Comput. Syst. Sci.*, 48(2):214–230, 1994. doi:10.1016/S0022-0000(05)80002-9.

4 Aaron Bernstein and David R. Karger. A nearly optimal oracle for avoiding failed vertices and edges. In *STOC*, pages 101–110, 2009.

5 Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Fault-tolerant approximate shortest-path trees. In *ESA*, pages 137–148, 2014. doi:10.1007/978-3-662-44777-2_12.

6 Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Multiple-edge-fault-tolerant approximate shortest-path trees. In *STACS*, pages 18:1–18:14, 2016.

7 Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. f-sensitivity distance oracles and routing schemes. In *ESA*, pages 84–96, 2010.

8 Annalisa D'Andrea, Mattia D'Emidio, Daniele Frigioni, Stefano Leucci, and Guido Proietti. Path-fault-tolerant approximate shortest-path trees. In *SIROCCO*, pages 224–238, 2015. doi:10.1007/978-3-319-25258-2_16.

9 Erik D. Demaine, Gad M. Landau, and Oren Weimann. On cartesian trees and range minimum queries. *Algorithmica*, 68(3):610–625, 2014. doi:10.1007/s00453-012-9683-x.

10 Camil Demetrescu, Mikkel Thorup, Rezaul Alam Chowdhury, and Vijaya Ramachandran. Oracles for distances avoiding a failed node or link. *SIAM J. Comput.*, 37(5):1299–1318, 2008. doi:10.1137/S0097539705429847.

11 Ran Duan and Seth Pettie. Dual-failure distance and connectivity oracles. In *SODA*, pages 506–515, 2009. URL: http://dl.acm.org/citation.cfm?id=1496770.1496826.

12 Fabrizio Grandoni and Virginia Vassilevska Williams. Improved distance sensitivity oracles via fast single-source replacement paths. In *FOCS*, pages 748–757, 2012.

13 Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984. doi:10.1137/0213024.

14 Enrico Nardelli, Guido Proietti, and Peter Widmayer. Swapping a failing edge of a single source shortest paths tree is good and fast. *Algorithmica*, 35(1):56–74, 2003.

15 Merav Parter. Dual failure resilient BFS structure. In *PODC*, pages 481–490, 2015.

16 Merav Parter and David Peleg. Sparse fault-tolerant BFS trees. In *ESA*, pages 779–790, 2013. doi:10.1007/978-3-642-40450-4_66.

17 Merav Parter and David Peleg. Fault tolerant approximate BFS structures. In *SODA*, pages 1073–1092, 2014.

18 Christian Sommer. Shortest-path queries in static networks. *ACM Comput. Surv.*, 46(4):45:1–45:31, 2014. doi:10.1145/2530531.

19 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005. A preliminary version of this paper appeared in *SODA'01*. doi:10.1145/1044731.1044732.

Efficient Algorithms with Asymmetric Read and Write Costs*

Guy E. Blelloch[1], Jeremy T. Fineman[2], Phillip B. Gibbons[3], Yan Gu[4], and Julian Shun[5]

1 **Carnegie Mellon University, Pittsburgh, USA**
2 **Georgetown University, Washington, USA**
3 **Carnegie Mellon University, Pittsburgh, USA**
4 **Carnegie Mellon University, Pittsburgh, USA**
5 **University of California, Berkeley, USA**

Abstract

In several emerging technologies for computer memory (main memory), the cost of reading is significantly cheaper than the cost of writing. Such asymmetry in memory costs poses a fundamentally different model from the RAM for algorithm design. In this paper we study lower and upper bounds for various problems under such asymmetric read and write costs. We consider both the case in which all but $O(1)$ memory has asymmetric cost, and the case of a small cache of symmetric memory. We model both cases using the (M, ω)-ARAM, in which there is a small (symmetric) memory of size M and a large unbounded (asymmetric) memory, both random access, and where reading from the large memory has unit cost, but writing has cost $\omega \gg 1$.

For FFT and sorting networks we show a lower bound cost of $\Omega(\omega n \log_{\omega M} n)$, which indicates that it is not possible to achieve asymptotic improvements with cheaper reads when ω is bounded by a polynomial in M. Moreover, there is an asymptotic gap (of $\min(\omega, \log n)/\log(\omega M)$) between the cost of sorting networks and comparison sorting in the model. This contrasts with the RAM, and most other models, in which the asymptotic costs are the same. We also show a lower bound for computations on an $n \times n$ diamond DAG of $\Omega(\omega n^2/M)$ cost, which indicates no asymptotic improvement is achievable with fast reads. However, we show that for the minimum edit distance problem (and related problems), which would seem to be a diamond DAG, we can beat this lower bound with an algorithm with only $O(\omega n^2/(M \min(\omega^{1/3}, M^{1/2})))$ cost. To achieve this we make use of a "path sketch" technique that is forbidden in a strict DAG computation. Finally, we show several interesting upper bounds for shortest path problems, minimum spanning trees, and other problems. A common theme in many of the upper bounds is that they require redundant computation and a tradeoff between reads and writes.

1998 ACM Subject Classification F.1.1 Models of Computation, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Computational Model, Lower Bounds, Shortest-paths, Non-Volatile Memory, Sorting Networks, Fast Fourier Transform, Diamond DAG, Minimum Spanning Tree

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.14

* This research was supported in part by NSF grants CCF-1314590, CCF-1314633 and CCF-1533858, the Intel Science and Technology Center for Cloud Computing, and the Miller Institute for Basic Research in Science at UC Berkeley.

1 Introduction

Fifty years of algorithms research has focused on settings in which reads and writes (to memory) have similar cost. But what if reads and writes to memory have significantly *different* costs? How would that impact algorithm design? What new techniques are useful for trading-off doing more cheaper operations (say more reads) in order to do fewer expensive operations (say fewer writes)? What are the fundamental limitations on such trade-offs (lower bounds)? What well-known equivalences for traditional memory fail to hold for asymmetric memories?

Such questions are coming to the fore with the arrival of new **main-memory** technologies [29, 32] that offer key potential benefits over existing technologies such as DRAM, including nonvolatility, signicantly lower energy consumption, and higher density (more bits stored per unit area). These emerging memories will sit on the processor's memory bus and be accessed at byte granularity via loads and stores (like DRAM), and are projected to become the dominant main memory within the decade [36, 49].[1] Because these emerging technologies store data as "states" of a given material, the cost of reading (checking the current state) is significantly cheaper than the cost of writing (modifying the physical state of the material): Reads are up to an order of magnitude or more lower energy, lower latency, and higher (per-module) bandwidth than writes [5, 6, 10, 11, 20, 21, 30, 31, 33, 41, 47].

This paper provides a first step towards answering these fundamental questions about asymmetric memories. We introduce a simple model for studying such memories, and a number of new results. In the simplest model we consider, there is an asymmetric random-access memory such that reads cost 1 and writes cost $\omega \gg 1$, as well as a constant number of symmetric "registers" that can be read or written at unit cost. More generally, we consider settings in which the amount of symmetric memory is $M \ll n$, where n is the input size: We define the (M, ω)-*Asymmetric RAM (ARAM)*, comprised of a symmetric small-memory of size M and an asymmetric large-memory of unbounded size with write cost ω. The *ARAM cost* Q is the number of reads from large-memory plus ω times the number of writes to large-memory. The *time* T is Q plus the number of reads and writes to small-memory.

We present a number of lower and upper bounds for the (M, ω)-ARAM, as summarized in Table 1. These results consider a number of fundamental problems and demonstrate how the asymptotic algorithm costs decrease as a function of M, e.g., polynomially, logarithmically, or not at all.

For FFT we show an $\Omega(\omega n \log_{\omega M} n)$ lower bound on the ARAM cost, and a matching upper bound. Thus, even allowing for redundant (re)computation of nodes (to save writes), it is not possible to achieve asymptotic improvements with cheaper reads when $\omega \in O(M^c)$ for a constant c. Prior lower bound approaches for FFTs for symmetric memory fail to carry over to asymmetric memory, so a new lower bound technique is required. We use an interesting new accounting argument for fractionally assigning a unit weight for each node of the network to subcomputations that each have cost ωM. The assignment shows that each subcomputation has on average at most $M \log(\omega M)$ weight assigned to it, and hence the total cost across all $\Theta(n \log n)$ nodes yields the lower bound.

For sorting, we show the surprising result that on asymmetric memories, comparison sorting is asymptotically faster than sorting networks. This contrasts with the RAM model (and I/O models, parallel models such as the PRAM, etc.), in which the asymptotic costs are the same! The lower bound leverages the same key partitioning lemma as in the FFT proof.

[1] While the exact technology is continually evolving, candidate technologies include phase-change memory, spin-torque transfer magnetic RAM, and memristor-based resistive RAM.

Table 1 Summary of Our Results for the (M, ω)-ARAM (†indicates main results).

problem	**ARAM cost** $Q(n)$ **or** $Q(n,m)$	**time** $T(n)$ **or** $T(n,m)$	**section**
FFT	$\Theta(\omega n \log n / \log(\omega M))^\dagger$	$\Theta(Q(n) + n \log n)$	3, 6
sorting networks	$\Omega(\omega n \log n / \log(\omega M))^\dagger$	$\Omega(Q(n) + n \log n)$	3
sorting (comparison)	$O(n(\log n + \omega))$	$\Theta(n(\log n + \omega))$	6, [10]
diamond DAG	$\Theta(n^2\omega/M)^\dagger$	$\Theta(Q(n) + n^2)$	3
longest common subsequence, edit distance	$O(n^2\omega/\min(\omega^{1/3}M, M^{3/2}))^\dagger$	$O(n^2(1 + \omega/\min(\omega^{1/3}M^{2/3}, M^{3/2})))^\dagger$	4
single-source shortest path	$O(\min(n(\omega + m/M), (m + n\log n)\omega, m(\omega + \log n)))^\dagger$	$O(Q(n,m) + n \log n)$	5
all-pairs shortest-path	$O(n^2(\omega + n/\sqrt{M}))$	$O(Q(n) + n^3)$	6
search tree, priority queue	$O(\omega + \log n)$ per update	$O(\omega + \log n)$ per update	6
2D convex hull, triangulation	$O(n(\log n + \omega))$	$\Theta(n(\log n + \omega))$	6
BFS, DFS, topological sort, biconnected components, SCC	$\Theta(\omega n + m)$	$\Theta(\omega n + m)$	6
minimum spanning tree	$O(m \min(\log n, n/M) + \omega n)^\dagger$	$O(Q(n,m) + n \log n)$	6

We present a tight lower bound for DAG computation on diamond DAGs that shows there is no asymptotic advantage of cheaper reads. On the other hand, we also show that allowing a vertex to be "partially" computed before all its immediate predecessors have been computed (thereby violating a DAG computation rule), we can beat the lower bound and show asymptotic advantage. Specifically, for both the longest common subsequence and edit distance problems (normally thought of as diamond DAG computations), we devise a new "path sketch" technique that leverages partial aggregation on the DAG vertices. Again we know of no other models in which such techniques are needed.

Finally, we show how to adapt Dijkstra's single-source shortest-paths algorithm using phases so that the priority queue is kept in small-memory, and briefly sketch how to adapt Borůvka's minimum spanning tree algorithm to reduce the number of shortcuts and hence writes that are needed. A common theme in many of our algorithms is that they use redundant computations and require a tradeoff between reads and writes.

Related Work. Prior work [7, 22, 25, 37, 38, 46] has studied read-write asymmetries in NAND flash memory, but this work has focused on (i) the asymmetric *granularity* of reads and writes in NAND flash chips: bits can only be cleared by incurring the overhead of erasing a large block of memory, and/or (ii) the asymmetric *endurance* of reads and writes: individual cells wear out after tens of thousands of writes to the cell. Emerging memories, in contrast, can read and write arbitrary bytes in-place and have many orders of magnitude higher write endurance, enabling system software to readily balance application writes across individual physical cells by adjusting its virtual-to-physical mapping. Other prior work has studied database query processing under asymmetric read-write costs [12, 13, 45, 46] or looked at other systems considerations [14, 30, 35, 48, 50, 51]. Our recent paper [10] introduced the general study of parallel (and external memory) models and algorithms with asymmetric read and write costs, focusing on sorting. Our follow-on paper [8] defined an abstract nested-parallel model of computation with asymmetric read-write costs that maps efficiently onto more concrete parallel machine models using a work-stealing scheduler, and presented reduced-write, work-efficient, highly-parallel algorithms for a number of fundamental problems such as tree contraction and convex hull. In contrast, this paper considers a much simpler model (the sequential (M, ω)-ARAM) and presents not just algorithms but also lower bounds—plus, the techniques are new. Finally, concurrent with this paper, Carson et al. [11] developed interesting upper and lower bounds for various linear algebra problems and direct N-body methods under asymmetric read and write costs. For sequential algorithms, they define a model similar to the (M, ω)-ARAM (as well as a cache-oblivious variant), and show that for "bounded data reuse" algorithms, i.e., algorithms in which each input or computed value is used only a constant number of times, the number of writes to asymmetric memory is asymptotically the same as the sum of the reads and writes to asymmetric memory. This implies, for example, a tight $\Omega(n \log n / \log M)$ lower bound on the number of writes for FFT under the bounded data reuse restriction; in contrast, our tight bounds for FFT do not have this restriction and use fewer writes. They also presented algorithms without this restriction for matrix multiplication, triangular solve, and Cholesky factorization that reduce the number of writes to Θ(output size), without increasing the number of reads, as well as various distributed-memory parallel algorithms.

2 Model and Preliminaries

We analyze algorithms in an *(M, ω)-ARAM*. In the model we assume a symmetric *small-memory* of size $M \geq 1$, an asymmetric *large-memory* of unbounded size, and a *write cost* $\omega \geq 1$, which we assume without loss of generality is an integer. (Typically, we are interested in the setting where $n \gg M$, where n is the input size, and $\omega \gg 1$.) We assume standard random access machine (RAM) instructions. We consider two cost measures for computations in the model. We define the (asymmetric) *ARAM cost Q* as the total number of reads from large-memory plus ω times the number of writes to large-memory. We define the (asymmetric) *time T* as the ARAM cost plus the number of reads from and writes to small-memory.[2] Because all instructions are from memory, this includes any cost of computation. In the paper we present results for both cost measures.

The model contrasts with the widely-studied external-memory model [1] in the asymmetry of the read and write costs. Also for simplicity in this paper we do not partition the memory into blocks of size B. Another difference is that the asymmetry implies that even the case of $M = O(1)$ (studied in [10] for sorting) is interesting. We note that our ARAM cost is a special case of the general flash model cost proposed in [4]; however that paper presents algorithms only for another special case of the model with symmetric read-write costs.

We use the term *value* to refer to an object that fits in one word (location) of the memory. We assume words are of size $\Theta(\log n)$ for input size n. The size M is the number of words in small-memory. All logarithms are base 2 unless otherwise noted. The DAG computation problem is given a DAG and a value for each of its input vertices (in-degree = 0), compute the value for each of its output vertices (out-degree = 0). The value of any non-input vertex can be computed in unit time given the value of all its immediate predecessors. As in standard I/O models [1] we assume values are atomic and cannot be split when mapped into the memory. The DAG computation problem can be modeled as a pebbling game

[2] The time metric models the fact that reads to certain emerging asymmetric memories are projected to be roughly as fast as reads to symmetric memory (DRAM). The ARAM cost metric Q does not make this assumption and hence is more generally applicable.

on the DAG [28]. Note that we allow (unbounded) recomputation of a DAG vertex, and indeed recomputation is a useful technique for reducing the number of writes (at the cost of additional reads).

3 Lower Bounds

We start by showing lower bounds for FFT DAGs, sorting networks and diamond DAGs. The idea in showing the lower bounds is to partition a computation into subcomputations that each have a lower bound on cost, but an upper bound on the number of inputs and outputs they can use. Our lower bound for FFT DAGs then uses an interesting accounting technique that gives every node in the DAG a unit weight, and fractionally assigns this weight across the subcomputations. In the special case $\omega = 1$, this leads to a simpler proof for the lower bound on the I/O complexity of FFT DAGs than the well-known bound by Hong and Kung [27].

We refer to a *subcomputation* as any contiguous sequence of instructions. The *outputs* of a subcomputation are the values written by the subcomputation that are either an output of the full computation or read by a later subcomputation. Symmetrically, the *inputs* of a subcomputation are the values read by the subcomputation that are either an input of the full computation or written by a previous subcomputation. An (l, m)-*partitioning* of a computation is a partitioning of instructions into subcomputations such that each has at most l inputs and at most m outputs. We allow for recomputation—instructions in different subcomputations might compute the same value.

▶ **Lemma 1.** *Any computation in the (M, ω)-ARAM has an $((\omega + 1)M, 2M)$-partitioning such that at most one of the subcomputations has ARAM cost $Q < \omega M$.*

Proof. We generate the partitioning constructively. Starting at the beginning, partition the instructions into contiguous blocks such that all but possibly the last block has cost $Q \geq \omega M$, but removing the last instruction from the block would have cost $Q < \omega M$. To remain within the cost bound each such subcomputation can read at most ωM values from large-memory. It can also read the at most M values that are in the small-memory when the subcomputation starts. Therefore it can read at most $(\omega + 1)M$ distinct values from the input or from previous subcomputations. Similarly, each subcomputation can write at most M values to large-memory, and an additional M values that remain in small-memory when the subcomputation ends. Therefore it can write at most $2M$ distinct values that are available to later subcomputations or the output. ◀

FFT. We now consider lower bounds for the DAG computation problem for the family of FFT DAGs (also called FFT networks, or butterfly networks). The FFT DAG of input size $n = 2^k$ consists of $k + 1$ levels each with n vertices (for a total of $n \log 2n$ vertices). Each vertex (i, j) at level $i \in 0, \ldots, k - 1$ and row j has two out-edges, which go to vertices $(i + 1, j)$ and $(i + 1, j \oplus 2^i)$ ($\oplus$ is the exclusive-or of the bit representation). This is the DAG used by the standard FFT (Fast Fourier Transform) computation. We note that in the FFT DAG there is at most a single path from any vertex to another.

▶ **Lemma 2.** *Any (l, m)-partitioning of a computation for simulating an n input FFT DAG has at least $n \log n/(m \log l)$ subcomputations.*

Proof. We refer to all vertices whose values are outputs of any subcomputation, as *partition output vertices*. We assign each such vertex arbitrarily to one of the subcomputations for which it is an output.

Consider the following accounting scheme for fractionally assigning a unit weight for each non-input vertex to some set of partition output vertices. If a vertex is a partition output vertex, then assign the weight to itself. Otherwise take the weight, divide it evenly between its two immediate descendants (out-edges) in the FFT DAG, and recursively assign that weight to each. For example, for a vertex x that is not a partition output vertex, if an immediate descendant y is a partition output vertex, then y gets a weight of $1/2$ from x, but if not and one of y's immediate descendants z is, then z gets a weight of $1/4$ from x. Since each non-input vertex is fully assigned across some partition output vertices, the sum of the weights assigned across the partition output vertices exactly equals $|V| - n = n \log n$. We now argue that every partition output vertex can have at most $\log l$ weight assigned to it. Looking back from an output vertex we see a binary tree rooted at the output. If we follow each branch of the tree until we reach an input for the subcomputation, we get a tree with at most l leaves, since there are at most l inputs and at most a single path from every vertex to the output. The contribution of each vertex in the tree to the output is $1/2^i$, where i is its depth (the root is depth 0). The leaves (subcomputation inputs) are not included since they are partition output vertices themselves, or inputs to the whole computation, which we have excluded. By induction on the tree structure, the weight of that tree is maximized when it is perfectly balanced, which gives a total weight of $\log l$.

Because every subcomputation can have at most m outputs, the total weight assigned to each subcomputation is at most $m \log l$. Since the total weight across all subcomputations is $n \log n$, the total number of subcomputations is at least $n \log n/(m \log l)$. ◀

▶ **Theorem 3** (FFT Lower Bound). *Any solution to the DAG computation problem on the family of FFT DAGs parametrized by input size n has costs $Q(n) = \Omega(\omega n \log n/ \log(\omega M))$ and $T(n) = \Omega(Q(n) + n \log n)$ on the (M, ω)-ARAM.*

Proof. By Lemma 1 every computation must have an $((\omega + 1)M, 2M)$-partitioning with subcomputation cost $Q \geq \omega M$ (except perhaps one). Plugging in Lemma 2 we have $Q(n) \geq \omega M n \log n/(2M \log((\omega + 1)M))$, which gives our bound on $Q(n)$. For $T(n)$ we just add in the cost of the computation of each vertex. ◀

Note that when $\omega \in O(M^c)$ for a constant c, these lower bounds match those for the standard external memory model [1, 27] assuming both reads and writes have cost ω. This implies that cheaper reads do not help asymptotically in this case. However, when this is not the case, cheaper reads do provide an asymptotic advantage, as will be seen by our matching upper bound given in Section 6.

Sorting Networks. A sorting network is a acyclic network of comparators, each of which takes two input keys and returns the minimum of the keys on one output, and the maximum on the other. For a family of sorting networks parametrized by n, each network takes n inputs, has n ordered outputs, and when propagating the inputs to the outputs must place the keys in sorted order on the outputs. A sorting network can be modeled as a DAG in the obvious way. Ajtai, Komlós and Szemerédi [3] described a family of sorting networks that have $O(n \log n)$ comparators and $O(\log n)$ depth. Follow-on work has provided many simplifications and constant factor improvements, including the well-known Patterson variant [40] and a simplification by Seiferas [43]. Recently Goodrich [26] gave a much simpler construction of an $O(n \log n)$ comparator network, but it requires polynomial depth. Here we show lower bounds for simulating any sorting network on the (M, ω)-ARAM.

▶ **Theorem 4** (Sorting Lower Bound). *Simulating any family of sorting networks parametrized on input size n has $Q(n) = \Omega(\omega n \log n / \log(\omega M))$ and $T(n) = \Omega(Q(n) + n \log n)$ on the (M, ω)-ARAM.*

Proof. Consider an (l, m)-partitioning of the computation. Each subcomputation has at most l inputs from the network, and m outputs for the network. The computation is oblivious to the values in the network (it can only place the minimum and maximum on the outputs of each comparator). Therefore locations of the inputs and outputs are independent of the input values. The total number of choices the subcomputation has is therefore $\binom{l}{m} m! = l!/(l-m)! < l^m$. Since there are $n!$ possible permutations, we have that the number of subcomputations k must satisfy $(l^m)^k \geq n!$. Taking the logarithm of both sides, rearranging, and using Stirling's formula we have $k > \log(n!)/(m \log l) > \frac{1}{2} n \log n/(m \log l)$ (for $n > e^2$). By Lemma 1 we have $Q(n) > \omega M \cdot \frac{1}{2} n \log n/(2M \log((1+\omega)M)) = \frac{1}{4} \omega n \log n / \log((1+\omega)M)$ (for $n > e^2$). ◀

These bounds are the same as for simulating an FFT DAG, and, as with FFTs, they indicate that faster reads do not asymptotically affect the lower bound unless ω is larger than any polynomial in M. These lower bounds rely on the sort being done on a network, and in particular that the location of all read and writes are oblivious to the data itself. As discussed in Section 6, for *general* comparison sorting algorithms, we can get better upper bounds than indicated by these lower bounds.

Diamond DAG. We consider the family of *diamond DAGs* parametrized by size $n \times m$. Each DAG has nm vertices arranged in a $n \times m$ grid such that every vertex $(i, j), 0 \leq i < (n-1), 0 \leq j < (m-1)$ has two out-edges to $(i+1, j)$ and $(i, j+1)$. The DAG has one input at $(0, 0)$ and one output at $(n-1, m-1)$. Diamond DAGs have many applications in dynamic programs, such as for the edit distance (ED), longest common subsequence (LCS), and optimal sequence alignment problems.

▶ **Lemma 5** (Cook and Sethi, 1976). *Solving the DAG computation problem on the family of diamond DAGs of size $n \times n$ requires n memory locations to store vertex values from the DAG.*

Proof. Cook and Sethi [16] showed that evaluating the top half of an $n \times n$ diamond DAG $(i + j \geq n - 1)$, which they call a pyramid DAG, requires n memory locations to store partial results. Because all paths of the diamond DAG must go through the top half, it follows for the diamond DAG. ◀

▶ **Theorem 6** (Diamond DAG Lower Bound). *The family of diamond DAGs parametrized by size $n \times m$ has $Q(n, m) = \Omega(\omega nm/M)$ and $T(n, m) = \Omega(Q(n, m) + nm)$ on the (M, ω)-ARAM.*

Proof. Consider the sub-DAG induced by a $2M \times 2M$ diamond $(a \leq i < a + 2M, b \leq j < b + 2M)$ of vertices. By Lemma 5 any subcomputation that computes the last output vertex of the sub-DAG requires $2M$ memory locations to store values from the diamond. The extra in-edges along two sides and out-edges along the other two can only make the problem harder. Half of the $2M$ required memory can be from small-memory, and so the remaining M must require writing those values to large-memory. Therefore every $2M \times 2M$ diamond requires M writes of values within the diamond. Partitioning the full diamond DAG into $2M \times 2M$ sub-diamonds, gives us $nm/(2M)^2$ partitions. Therefore the total number of writes is at least $M \times nm/(2M)^2 = nm/(4M)$, each with cost ω. For the time bound we need to add the nm calculations for all vertex values. ◀

This lower bound is asymptotically tight since a diamond DAG can be evaluated with matching upper bounds by evaluating each $M/2 \times M/2$ diamond sub-DAG as a subcomputation with M inputs, outputs and memory.

These bounds show that for the DAG computation problem on the family of diamond DAGs there is no asymptotic advantage of having cheaper reads. In Section 4 we show that for the ED and LCS problems (normally thought of as a diamond DAG computation), it is possible to do better than the lower bounds. This requires breaking the DAG computation rule by partially computing the values of each vertex before all inputs are ready. The lower bounds are interesting since they show that improving asymptotic performance with cheaper reads requires breaking the DAG computation rule.

4 Longest Common Subsequence and Edit Distance

This section describes a more efficient dynamic-programming algorithm for longest common subsequence (LCS) and edit distance (ED). The standard approach for these problems (an $M \times M$ tiling) results in an ARAM cost of $O(mn\omega/M)$ and time of $O(mn+mn\omega/M)$, where m and n are the length of the two input strings. Lemma 6 states that the standard bound is optimal under the standard DAG computation rule that all inputs must be available before evaluating a node. Perhaps surprisingly, we are able to beat these bounds by leveraging the fact that dynamic programs do not perform arbitrary functions at each node, and hence we do not necessarily need all inputs to begin evaluating a node.

Our main result is captured by the following theorem for large input strings. For smaller strings, we can do even better (see the full version of the paper [9]).

▶ **Theorem 7.** *Let $k_T = \min((\omega/M)^{1/3}, \sqrt{M})$ and suppose $m, n = \Omega(k_T M)$. Then it is possible to compute the ED or length of the LCS with time $T(m,n) = O(mn + mn\omega/(k_T M))$.*

Let $k_Q = \min(\omega^{1/3}, \sqrt{M})$ and suppose $m, n = \Omega(k_Q M)$. Then it is possible to compute the ED or length of the LCS with an ARAM cost of $Q(m,n) = O(mn\omega/(k_Q M))$.

To understand these bounds, our algorithm beats the ARAM cost of the standard tiling algorithm by a k_Q factor. And if $\omega \geq M$, our algorithm (using different tuning parameters) beats the time of the standard tiling algorithm by a k_T factor.

Overview. The dynamic programs for LCS and ED correspond to computing the shortest path through an $m \times n$ grid with diagonal edges, where m and n are the string lengths. We focus here on computing the length of the shortest path, but it is possible to output the path as well with the same asymptotic complexity (see [9]). Without loss of generality, we assume that $m \leq n$, so the grid is at least as wide as it is tall. For LCS, all horizontal and vertical edges have weight 0; the diagonal edges have weight -1 if the corresponding characters in the strings match, and weight ∞ otherwise. For ED, horizontal and vertical edges have weight 1, and diagonal edges have weights either 0 or 1 depending on whether the characters match. Our algorithm is not sensitive to the particular weights of the edges, and thus it applies to both problems and their generalizations.

Note that the $m \times n$ grid is not built explicitly since building and storing the graph would take $\Theta(mn)$ writes if $mn \gg M$. To get any improvement, it is important that subgrids reuse the same space. The weights of each edge can be inferred by reading the appropriate characters in each input string.

Our algorithm partitions the implicit grid into size-$(hM' \times kM')$, where h and k are parameters of the algorithm to be set later, and $M' = M/c$ for large enough constant $c > 1$

to give sufficient working space in small-memory. When string lengths m and $n \geq m$ are both "large", we use $h = k$ and thus usually work with $kM' \times kM'$ square subgrids. If the smaller string length m is small enough, we instead use parameters $h < k$. To simplify the description of the algorithm, we assume without loss of generality that m and n are divisible by hM' and kM', respectively, and that M is divisible by c.

Our algorithm operates on one $hM' \times kM'$ rectangle at a time, where the edges are directed right and down. The shortest-path distances to all nodes along the bottom and right boundary of each rectangle are explicitly written out, but all other intermediate computations are discarded. We label the vertices $u_{i,j}$ for $1 \leq i \leq hM'$ and $1 \leq j \leq kM'$ according to their row and column in the square, respectively, starting from the top-left corner. We call the vertices immediately above or to the left of the square the *input nodes*. The input nodes are all outputs for some previously computed rectangle. We call the vertices $u_{hM',j}$ along the bottom boundary and $u_{i,kM'}$ along the right boundary the *output nodes*.

The goal is to reduce the number of writes, thereby decreasing the overall cost of computing the output nodes, which we do by sacrificing reads and time. It is not hard to see that recomputing internal nodes enables us to reduce the number of writes. Consider, for example, the following simple approach assuming $M = \Theta(1)$: For each output node of a $k \times k$ square, try all possible paths through the square, keeping track of the best distance seen so far; perform a write at the end to output the best value.[3] Each output node tries $2^{\Theta(k)}$ paths, but only a $\Theta(1/k)$-fraction of nodes are output nodes. Setting $k = \Theta(\lg \omega)$ reduces the number of writes by a $\Theta(\lg \omega)$-factor at the cost of $\omega^{O(1)}$ reads. This same approach can be extended to larger M, giving the same $\lg \omega$ improvement, by computing "bands" of nearby paths simultaneously. But our main algorithm, which we discuss next, is much better as M gets larger (see Theorem 7).

Path sketch. The key feature of the grid leveraged by our algorithm is that shortest paths do not cross, which enables us to avoid the exponential recomputation of the simple approach. The noncrossing property has been exploited previously for building shortest-path data structures on the grid (e.g., [42]) and more generally planar graphs (e.g., [23, 34]). These previous approaches do not consider the cost of writing to large-memory, and they build data structures that would be too large for our use. Our algorithm leverages the available small-memory to compute bands of nearby paths simultaneously. We capture both the noncrossing and band ideas through what we call a path sketch, which we define as follows. The path sketch enables us to cheaply recompute the shortest paths to nodes.

We call every M'-th row in the square a *superrow*, meaning there are h superrows in the square. The algorithm partitions the i-th superrow into *segments* $\langle i, \ell, r\rangle$ of consecutive elements $u_{iM',\ell}, u_{iM',\ell+1}, \ldots, u_{iM',r}$. The main restriction on segments is that $r < \ell + M'$, i.e., each segment consists of at most M' consecutive elements in the superrow. Note that the segment boundaries are determined by the algorithm and are input dependent.

A *path sketch* is a sequence of segments $\langle s, \ell_s, r_s\rangle$, $\langle s+1, \ell_{s+1}, r_{s+1}\rangle$, $\langle s+2, \ell_{s+2}, r_{s+2}\rangle, \ldots,$ $\langle i, \ell_i, r_i\rangle$, summarizing the shortest paths to the segment. Specifically, this sketch means that for each vertex in the last segment, there is a shortest path to that vertex that goes through a vertex in each of the segments in the sketch. If the sketch starts at superrow 1, then the path originates from a node above the first superrow (i.e., the top boundary or the topmost

[3] This approach requires constant small-memory to keep the best distance, the current distance, and working space for computing the current distance. We also need bits proportional to the path length to enumerate paths.

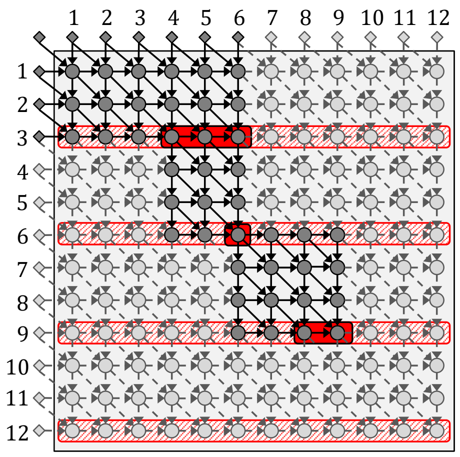


Figure 1 Example square grid and path sketch for $M' = 3$ and $h = k = 4$. The circles are nodes in the square. The diamonds are input nodes (outputs of adjacent squares), omitting irrelevant edges. The red slashes are the 4 superrows, and the solid red are the sketch segments.

M' nodes of the left boundary). If the sketch starts with superrow $s > 1$, then the path originates at one of the M' nodes on the left boundary between superrows $s-1$ and s. Since paths cannot go left, the path sketch also satisfies $\ell_s \leq \ell_{s+1} \leq \cdots \leq \ell_i$.

Evaluating a path sketch. Given a path sketch, we refer to the process of determining the shortest-path distances to all nodes in the final segment $\langle i, \ell_i, r_i \rangle$ as *evaluating the path sketch* or *evaluating the segment*, with the distances in small-memory when the process completes. Note that we have not yet described how to build the path sketch, as the building process uses evaluation as a subroutine.

The main idea of evaluating the sketch is captured by Figure 1 for the example sketch $\langle 1, 4, 6 \rangle, \langle 2, 6, 6 \rangle, \langle 3, 8, 9 \rangle$. The sketch tells us that shortest paths to $u_{9,8}$ and $u_{9,9}$ pass through one of $u_{3,4}, u_{3,5}, u_{3,6}$ and the node $u_{6,6}$. Thus, to compute the distances to $u_{9,8}$ and $u_{9,9}$, we need only consider paths through the darker nodes and solid edges—the lighter nodes and dashed edges are not recomputed during evaluation.

The algorithm works as follows. First compute the shortest-path distances to the first segment in the sketch. To do so, horizontally sweep a height-$(M'+1)$ column across the $(M'+1) \times kM'$ slab raising above the s-th superrow, keeping two columns in small-memory at a time. Also keep the newly computed distances to the first segment in small-memory, and stop the sweep at the right edge of the segment. More generally, given the distances to a segment in small-memory, we can compute the values for the next segment in the same manner by sweeping a column through the slab. This algorithm yields the following performance.

▶ **Lemma 8.** *Given a path sketch $\langle s, \ell_s, r_s \rangle, \ldots, \langle i, \ell_i, r_i \rangle$ in an $hM' \times kM'$ grid with distances to all input nodes computed, our algorithm correctly computes the shortest-path distances to all nodes in the segment $\langle i, \ell_i, r_i \rangle$. Assuming $k \geq h$ and small-memory size $M \geq 5M' + \Theta(1)$, the algorithm requires $O(kM^2)$ operations in small-memory, $O(kM)$ reads, and 0 writes.*

Proof. Correctness follows from the definition of the path sketch: the sweep performed by the algorithm considers all possible paths that pass through these segments.

The algorithm requires space in small-memory to store two columns in the current slab, the previous segment in the sketch, and the next segment in the sketch, and the two segment boundaries themselves, totaling $4M' + \Theta(1)$ small-memory. Due to the monotonically increasing left endpoints of each segment, the horizontal sweep repeats at most M' columns per supperrow, so the total number of column iterations is $O(kM' + hM') = O(kM')$. Multiplying by M' gives the number of nodes computed.

The main contributor to reads is the input strings themselves to infer the structure/weights of the grid. With M' additional small-memory, we can store the "vertical" portion of the input string used while computing each slab, and thus the vertical string is read only once with $O(hM') = O(kM')$ reads. The "horizontal" input characters can be read with each of the $O(kM')$ column-sweep iterations. An additional k reads suffice to read the sketch itself, which is a lower-order term. ◀

Building the path sketch. The main algorithm on each rectangle involves building the set of sketches to segments in the bottom superrow. At some point during the sketch-building process, the distances to each output node is computed, at which point it can be written out. The main idea of the algorithm is a sketch-extension subroutine: given segments in the i-th superrow and their sketches, extend the sketches to produce segments in the $(i+1)$-th superrow along with their sketches.

Our algorithm builds up an ordered list of consecutive path sketches, one superrow at a time. The first superrow is partitioned into k segments, each containing exactly M' consecutive nodes. The list of sketches is initialized to these segments.

Given a list of sketches to the i-th superrow, our algorithm extends the list of sketches to the $(i+1)$-th superrow as follows. The algorithm sweeps a height-$(M'+1)$ column across the $(M'+1) \times kM'$ slab between these superrows (inclusive). The sweep begins at the left end of the slab, reading the input values from the left boundary, and continuing across the entire width of the slab. In small-memory, we evaluate the first segment of the i-th superrow (using the algorithm from Lemma 8). Whenever the sweep crosses a segment boundary in the i-th superrow, again evaluate the next segment in the i-th superrow. For each node in the slab, the sweep calculates both the shortest-path distance and a pointer to the segment in the previous superrow from whence the shortest path originates (or a null pointer if it originates from the left boundary). When the originating segment of the bottom node (the node in the $(i+1)$-th superrow) changes, the algorithm creates a new segment for the $(i+1)$-th superrow and appends it to the sketch of the originating segment. If the segment in the current segment in the $(i+1)$-th superrow grows past M' elements, a new segment is created instead and the current path sketch is copied and spliced into the list of sketches. Any sketch that is not extended through this process is no longer relevant and may be spliced out of the list of sketches. When the sweep reaches a node on the output boundary (right edge or bottom edge of the square), the distance to that node is written out.

▶ **Lemma 9.** *The sketching algorithm partitions the i-th superrow into at most ik segments.*

Proof. The proof is by induction over superrows. As a base case, the first superrow consists of exactly k segments. For the inductive step, there are two cases in which a new segment is started in the $(i+1)$-th superrow. The first case is that the originating segment changes, which can occur at most ik times by inductive assumption. The second case is that the current segment grows too large, which can occur at most k times. We thus have at most $(i+1)k$ segments in the $(i+1)$-th superrow. ◀

▶ **Lemma 10.** *Suppose $h \le k$ and small-memory $M \ge 11M' + \Theta(1)$, and consider an $hM' \times kM'$ grid with distances to input nodes already computed. Then the sketch building algorithm correctly computes the distances to all output boundary nodes using $O((hk)^2 M^2)$*

operations in small-memory, $O((hk)^2M)$ reads from large-memory, and $O(h^2k + X)$ writes to large-memory, where $X = O(kM)$ is the number of boundary nodes written out.

Proof. Consider the cost of computing each slab, ignoring the writes to the output nodes. We reserve $5M' + \Theta(1)$ small-memory for the process of evaluating segments in the previous superrow. To perform the sweep in the current slab, we reserve M' small-memory to store one segment in the previous row, M' small-memory to store characters in the “vertical” input string, $4(M' + 1)$ small-memory to store two columns (each with distances and pointers) for the sweep, and an additional $\Theta(1)$ small-memory to keep, e.g., the current segment boundaries. Since there are at most hk segments in the previous superrow (Lemma 9), the algorithm evaluates at most hk segments; applying Lemma 8, the cost is $O(hk^2M^2)$ operations, $O(hk^2M)$ reads, and 0 writes. There are an additional $O(kM^2)$ operations to sweep through the kM^2 nodes in the slab, plus $O(kM)$ reads to scan the “horizontal” input string. Finally, there are $O(hk)$ writes to extend existing sketches and $O(hk)$ writes to copy at most k sketches.

Summing across all h slabs and accounting for the output nodes, we get $O((hk)^2M^2 + hkM^2)$ operations, $O((hk)^2M + hkM)$ reads, and $O(h^2k + X)$ writes. Removing the lower-order terms gives the lemma. ◀

Combining across all rectangles in the grid, we get the following corollary.

▶ **Corollary 11.** *Let $m \leq n$ be the length of the two input strings, with $m \geq M$. Suppose $h = O(m/M)$ and $k = O(n/M)$ with $h \leq k$. Then it is possible to compute the LCS or edit distance of the strings with $O(mnhk)$ operations in small-memory, $O(mnhk/M)$ reads to large-memory, and $O(mnh/M^2 + mn/(hM))$ writes to large-memory.*

Proof. There are $\Theta(mn/(hkM^2))$ size-$(hM/11) \times (kM/11)$ subgrids. Multiplying by the cost of each grid (Lemma 10) gives the bound. ◀

Setting $h = k = 1$ gives the standard $M \times M$ tiling with $O(nm)$ time and $O(mn\omega/M)$ ARAM cost. As the size of squares increase, the fraction of output nodes and hence writes decreases, at the cost of more overhead for operations in small-memory and reads from large-memory. Assuming both n and m are large enough to do so, plugging in $h = k = \max\{1, k_T\}$ or $h = k = k_Q$ with a few steps of algebra to eliminate terms yields Theorem 7.

5 Single-Source Shortest Paths

The single-source shortest-paths (SSSP) problem takes a directed weighted graph $G = (V, E)$ and a source vertex $s \in V$, and outputs the shortest distances $d(s, v)$ from s to every other vertex in $v \in V$. For graphs with non-negative edge weights, the most efficient algorithm is Dijkstra’s algorithm [19].

In this section we will study (variants of) Dijkstra’s algorithm in the asymmetric setting. We describe and analyze three versions (two classical and one new variant) of Dijkstra’s algorithm, and the best version can be chosen based on the values of M, ω, the number of vertices $n = |V|$, and the number of edges $m = |E|$.

▶ **Theorem 12.** *The SSSP problem on a graph $G = (V, E)$ with non-negative edge weights can be solved with $Q(n, m) = O(\min(n(\omega + m/M), (m + n \log n)\omega, m(\omega + \log n)))$ and $T(n, m) = O(Q(n, m) + n \log n)$, both in expectation, on the (M, ω)-ARAM.*

We start with the classical Dijkstra's algorithm [19], which maintains for each vertex v, $\delta(v)$, a tentative upper bound on the distance, initialized to $+\infty$ (except for $\delta(s)$, which is initialized to 0). The algorithm consists of $n-1$ iterations, and the final distances from s are stored in $\delta(\cdot)$. In each iteration, the algorithm selects the unvisited vertex u with smallest finite $\delta(u)$, marks it as *visited*, and uses its outgoing edges to relax (update) all of its neighbors' distances. A priority queue is required to efficiently select the unvisited vertex with minimum distance. Using a Fibonacci heap [24], the time of the algorithm is $O(m+n\log n)$ in the standard (symmetric) RAM model. In the (M,ω)-ARAM, the costs are $Q = T = O((m+n\log n)\omega)$ since the Fibonacci heap requires asymptotically as many writes as reads. Alternatively, using a binary search tree for the priority queue reduces the number of writes (see Section 6) at the cost of increasing the number of reads, giving $Q = T = O(m\log n + \omega m)$. These bounds are better when $m = o(\omega n)$. Both of these variants store the priority queue in large-memory, requiring at least one write to large-memory per edge.

We now describe an algorithm, which we refer to as *phased Dijkstra*, that fully maintains the priority queue in small-memory and only requires $O(n)$ writes to large-memory. The idea is to partition the computation into phases such that for a parameter M' each phase needs a priority queue of size at most $2M'$ and visits at least M' vertices. By selecting $M' = M/c$ for an appropriate constant c, the priority queue fits in small-memory, and the only writes to large-memory are the final distances.

Each phase starts and ends with an empty priority queue P and consists of two parts. A Fibonacci heap is used for P, but is kept small by discarding the M' largest elements (vertex distances) whenever $|P| = 2M'$. To do this P is flattened into an array, the M'-th smallest element d_{max} is found by selection, and the Fibonacci heap is reconstructed from the elements no greater than d_{max}, all taking linear time. All further insertions in a given phase are not added to P if they have a value greater than d_{max}. The first part of each phase loops over all edges in the graph and relaxes any that go from a visited to an unvisited vertex (possibly inserting or decreasing a key in P). The second part then runs the standard Dijkstra's algorithm, repeatedly visiting the vertex with minimum distance and relaxing its neighbors until P is empty. To implement relax, the algorithm needs to know whether a vertex is already in P, and if so its location in P so that it can do a decrease-key on it. It is too costly to store this information with the vertex in large-memory, but it can be stored in small-memory using a hash table.

The correctness of this phased Dijkstra's algorithm follows from the fact that it only ever visits the closest unvisited vertex, as with the standard Dijkstra's algorithm.

▶ **Lemma 13.** *Phased Dijkstra's has $Q(n,m) = O(n(\omega + m/M))$ and $T(n,m) = O(Q(n,m) + n\log n)$ both in expectation (for $M \le n$).*

Proof. During a phase either the size of P will grow to $2M'$ (and hence delete some entries) or it will finish the algorithm. If P grows to $2M'$ then at least M' vertices are visited during the phase since that many need to be deleted with delete-min to empty P. Therefore the number of phases is at most $\lceil n/M' \rceil$. Visiting all edges in the first part of each phase involves at most m insertions and decrease-keys into P, each taking $O(1)$ amortized time in small-memory, and $O(1)$ time to read the edge from large-memory. Since compacting Q when it overflows takes linear time, its cost can be amortized against the insertions that caused the overflow. The cost across all phases for the first part is therefore $Q = W = O(m\lceil n/M' \rceil)$. For the second part, every vertex is visited once and every edge relaxed at most once across all phases. Visiting a vertex requires a delete-min in small-memory and a write to large-memory, while relaxing an edge requires an insert or decrease-key in small-memory, and $O(1)$ reads from large-memory. We therefore have for this second part (across all phases) that $Q = O(\omega n + m)$

and $W = O(n(\omega + \log n) + m)$. The operations on P each include an expected $O(1)$ cost for the hash table operations. Together this gives our bounds. ◀

Compared to the first two versions of Dijkstra's algorithm with $Q = T = O(\omega m + \min(\omega n \log n, m \log n))$, the new algorithm is strictly better when $\omega M > n$. More specifically, the new algorithm performs better when $nm/M < \max\{\omega m, \min(\omega n \log n, m \log n)\}$. Combining these three algorithms proves Theorem 12, when the best one is chosen based on the parameters M, ω, n, and m.

6 Further Results and Open Problems

In this section we outline results for a variety of other problems. The bounds are summarized as part of Table 1.

FFT. For the FFT we can match the lower bound using the algorithm described elsewhere [10], although in that case the computation cost was not considered. The idea is to first split the DAG into layers of $\log(\omega M)$ levels. Then divide each layer so that the last $\log M$ levels are partitioned into FFT networks of output size M. Attach to each partition all needed inputs from the layer and the vertices needed to reach them (note that these vertices will overlap among partitions). Each extended partition will have ωM inputs and M outputs, and can be computed in M small-memory with $Q = O(\omega M)$, and $T = O((\omega + \log M)M)$. This gives a total upper bound of $Q = O(\omega M \times n \log n/(M \log(\omega M))) = O(\omega n \log n/\log(\omega M))$, and $T = O(Q(n) + n \log n)$, which matches the lower bound (asymptotically). All computations are done within the DAG model.

Search Trees and Priority Queues. We now consider algorithms for some problems that can be implemented efficiently using balanced binary search trees. In the following discussion we assume $M = O(1)$. Red-black trees with appropriate rebalancing rules require only $O(1)$ amortized time per update (insertion or deletion) once the location for the key is found [44]. For a tree of size n finding a key's location uses $O(\log n)$ reads but no writes, so the total amortized cost $Q = T = O(\omega + \log n)$ per update in the (M, ω)-ARAM. For arbitrary sequences of searches and updates, $\Omega(\omega + \log n)$ is a matching lower bound on the amortized cost per operation when $M = O(1)$. Because priority queues can be implemented with a binary search tree, insertion and delete-min have the same bounds. It seems more difficult, however, to reduce the number of writes for priority queues that support efficient melding or decrease-key.

Sorting. Sorting can be implemented with $Q = T = O(n(\log n + \omega))$ by inserting all keys into a red-black tree and then reading them off in priority order [10]. We note that this bound on time is better than the sorting network lower bound (Theorem 4). For example, when $\omega = M = \log n$ it gives a factor of $\log n/\log\log n$ improvement. The additional power is a consequence of being able to randomly write to one of n locations at the leaves of the tree for each insertion. The bound is optimal for T because n writes are required for the output and comparison-based sorting requires $O(n \log n)$ operations.

Convex Hull and Triangulation. A variety of problems in computational geometry can be solved optimally using balanced trees and sorting. The planar convex-hull problem can be solved by first sorting the points by x coordinates and then either using Overmars' technique or Graham's scan [18]. In both cases, the second part takes linear time so the overall cost is

$O(\mathit{Sort}(n))$. The planar Delaunay triangulation problem can be solved efficiently with the plane sweep method [18]. This involves maintaining a priority queue on x coordinate, and maintaining a balanced binary search tree on the y coordinate. A total of $O(n)$ operations are required on each, again giving bounds $O(\mathit{Sort}(n))$.

BFS and DFS. Breadth-first and depth-first search can be performed with $Q = T = O(\omega n + m)$. In particular each vertex only requires a constant number of writes when it is first added to the frontier (the stack or queue) and a constant number of writes when it is finished (removed from the stack or queue). Searches along an edge to an already visited vertex require no writes. This implies that several problems based on BFS and DFS also only require $Q = T = O(\mathit{DFS}(n))$. Such problems include topological sort, biconnected components, and strongly connected components. The analysis is based on the fact that there are only $O(n)$ forward edges in the DFS. However when using priority-first search on a weighted graph (e.g., Dijkstra's or Prim's algorithm) then the problem is more difficult to perform optimally as the priority queue might need to be updated for every visited edge.

Dynamic Programming. With regards to dynamic programming, some problems are reasonably easy and some harder. We covered LCS and ED in Section 4. The standard Floyd-Warshall algorithm for the all-pairs shortest-path (APSP) problem uses $O(n^3)$ writes. However, by rearranging the loops and carefully scheduling the writes it is possible to implement the algorithm using only $O(n^2)$ writes and $O(n^3)$ reads, giving $T = O(\omega n^2 + n^3)$ [9]. This version, however, is not efficient in terms of Q. Kleene's divide-and-conquer algorithm [2] can be used to reduce the ARAM cost [39]. Each recursive call makes two calls to itself on problems of half the size, and six calls to matrix multiply over the semiring $(\min, +)$. Here we analyze the algorithm in the (M, ω)-ARAM. The matrix multiplies on two matrices of size $n \times n$ can be done in the model in $Q_M(n) = O(n^2(\omega + n/\sqrt{M}))$ [10]. This leads to the recurrence $Q_{\mathit{Kleene}}(n) = 2Q_{\mathit{Kleene}}(n/2) + O(Q_M(n)) + O(\omega n^2)$, which solves to $Q_{\mathit{Kleene}}(n) = O(Q_M(n))$ because the cost is dominated at the root of the recurrence. It is not known whether this is optimal. A similar approach can be used for several other problems, including sequence alignment with gaps, optimal binary search trees, and matrix chain multiplication [15].

Minimum Spanning Tree. The standard algorithms for the minimum spanning tree (MST) problem are not write efficient. However using a variant of Borůvka's algorithm, and careful management of a union-find data structure, MST can be made write-efficient. In particular, for an input graph in adjacency-list format with n vertices and m edges, MST has cost $Q(n, m) = T(n, m) = O(m \log n + \omega n)$ on the (M, ω)-ARAM, with $M = O(1)$. We outline the idea here—a more detailed description is given in the full paper [9].

The algorithm proceeds in two phases. For the first phase, consisting of the first $\log\log n$ rounds of Borůvka steps, the algorithm performs no shortcuts on a union-find tree. Thus it will leave chains of length up to $\log\log n$ that need to be followed for each root lookup. Also all vertices of each component are maintained in a linked list so that a component (root) can be searched with just reads. Because there are at most $O(m \log\log n)$ queries during the first $\log\log n$ rounds, the total number of reads for identifying the minimum edges between components in the first phase is $O(m(\log\log n)^2)$. There are $O(n)$ writes for creating links in the tree, linking components in the lists, and outputting selected edges.

After this first phase, the algorithm shortcuts all vertices to point directly to their root ($O(n)$ reads and writes). We refer to the roots as the phase-one components. In the second phase, after each of the remaining rounds the algorithm shortcuts the phase-one components to point directly to their new root. Because there can only be at most $n/\log n$ phase-one components, and at most $\log n - \log\log n$ rounds in the second phase, the total number of reads and writes for these updates is $O(n)$. Looking up a vertex takes two steps: one to find its phase-one component and another to get to the current component (root), both taking constant time. Therefore the total number of reads for identifying the minimum edges between components in the second phase is $O(m) \times (\log n - \log\log n) = O(m\log n)$. Summing the reads and writes gives the above bounds. Observing that the SSSP algorithm in Section 5 can be adapted to implement Prim's MST algorithm [17] yields the bounds in Table 1 (the n/M term is in expectation).

References

1 Alok Aggarwal and Jeffrey S. Vitter. The Input/Output complexity of sorting and related problems. *Communications of the ACM*, 31(9), 1988. doi:10.1145/48529.48535.

2 Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA, 1974.

3 Miklós Ajtai, János Komlós, and Endre Szemerédi. An $O(n\log n)$ sorting network. In *Proc. ACM Symposium on Theory of Computing (STOC)*, 1983.

4 Deepak Ajwani, Andreas Beckmann, Riko Jacob, Ulrich Meyer, and Gabriel Moruz. On computational models for flash memory devices. In *Proc. ACM International Symposium on Experimental Algorithms (SEA)*, 2009.

5 Ameen Akel, Adrian M. Caulfield, Todor I. Mollov, Rajech K. Gupta, and Steven Swanson. Onyx: A prototype phase change memory storage array. In *Proc. USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage)*, 2011.

6 Manos Athanassoulis, Bishwaranjan Bhattacharjee, Mustafa Canim, and Kenneth A. Ross. Path processing using solid state storage. In *Proc. International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures (ADMS)*, 2012.

7 Avraham Ben-Aroya and Sivan Toledo. Competitive analysis of flash-memory algorithms. In *Proc. European Symposium on Algorithms (ESA)*, 2006.

8 Naama Ben-David, Guy E. Blelloch, Jeremy T. Fineman, Phillip B. Gibbons, Yan Gu, Charlie McGuffey, and Julian Shun. Parallel algorithms for asymmetric read-write costs. In *Proc. ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2016.

9 Guy E. Blelloch, Jeremy T. Fineman, Phillip B. Gibbons, Yan Gu, and Julian Shun. Efficient algorithms with asymmetric read and write costs. *arXiv preprint arXiv:1511.01038*, 2015.

10 Guy E. Blelloch, Jeremy T. Fineman, Phillip B. Gibbons, Yan Gu, and Julian Shun. Sorting with asymmetric read and write costs. In *Proc. ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2015.

11 Erin Carson, James Demmel, Laura Grigori, Nicholas Knight, Penporn Koanantakool, Oded Schwartz, and Harsha V. Simhadri. Write-avoiding algorithms. In *Proc. IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, 2016.

12 Shimin Chen, Phillip B. Gibbons, and Suman Nath. Rethinking database algorithms for phase change memory. In *Proc. Conference on Innovative Data Systems Research (CIDR)*, 2011.

13 Shimin Chen and Qin Jin. Persistent B$^+$-trees in non-volatile main memory. *PVLDB*, 8(7), 2015.

14 Sangyeun Cho and Hyunjin Lee. Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance. In *Proc. IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009.

15 Rezaul A. Chowdhury and Vijaya Ramachandran. Cache-oblivious dynamic programming. In *Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2006.

16 Stephen Cook and Ravi Sethi. Storage requirements for deterministic polynomial time recognizable languages. *Journal of Computer and System Sciences*, 13(1), 1976.

17 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms (3rd edition)*. MIT Press, 2009.

18 Mark de Berg, Otfried Cheong, Mark van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 2008.

19 Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1), 1959.

20 Xiangyu Dong, Norman P. Jouupi, and Yuan Xie. PCRAMsim: System-level performance, energy, and area modeling for phase-change RAM. In *Proc. ACM International Conference on Computer-Aided Design (ICCAD)*, 2009.

21 Xiangyu Dong, Xiaoxia Wu, Guangyu Sun, Yuan Xie, Hai H. Li, and Yiran Chen. Circuit and microarchitecture evaluation of 3D stacking magnetic RAM (MRAM) as a universal memory replacement. In *Proc. ACM Design Automation Conference (DAC)*, 2008.

22 David Eppstein, Michael T. Goodrich, Michael Mitzenmacher, and Pawel Pszona. Wear minimization for cuckoo hashing: How not to throw a lot of eggs into one basket. In *Proc. ACM International Symposium on Experimental Algorithms (SEA)*, 2014.

23 Jittat Fakcharoenphol and Satish Rao. Planar graphs, negative weight edges, shortest paths, near linear time. In *Proc. IEEE Symposium on Foundations of Computer Science (FOCS)*, 2001.

24 Michael L. Fredman and Robert E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3), 1987.

25 Eran Gal and Sivan Toledo. Algorithms and data structures for flash memories. *ACM Computing Surveys*, 37(2), 2005.

26 Michael T. Goodrich. Zig-zag sort: A simple deterministic data-oblivious sorting algorithm running in $O(n \log n)$ time. In *Proc. ACM Symposium on Theory of Computing (STOC)*, 2014.

27 Jia-Wei Hong and H. T. Kung. I/O complexity: The red-blue pebble game. In *Proc. ACM Symposium on Theory of Computing (STOC)*, 1981.

28 John Hopcroft, Wolfgang Paul, and Leslie Valiant. On time versus space. *Journal of the ACM*, 24(2), 1977.

29 HP, SanDisk partner on memristor, ReRAM technology. http://www.bit-tech.net/news/hardware/2015/10/09/hp-sandisk-reram-memristor, October 2015.

30 Jingtong Hu, Qingfeng Zhuge, Chun Jason Xue, Wei-Che Tseng, Shouzhen Gu, and Edwin Sha. Scheduling to optimize cache utilization for non-volatile main memories. *IEEE Transactions on Computers*, 63(8), 2014.

31 www.slideshare.net/IBMZRL/theseus-pss-nvmw2014, 2014.

32 Intel and Micron produce breakthrough memory technology. http://newsroom.intel.com/community/intel_newsroom/blog/2015/07/28/intel-and-micron-produce-breakthrough-memory-technology, July 2015.

33 Hyojun Kim, Sangeetha Seshadri, Clement L. Dickey, and Lawrence Chu. Evaluating phase change memory for enterprise storage systems: A study of caching and tiering approaches. In *Proc. USENIX Conference on File and Storage Technologies (FAST)*, 2014.

34 Philip N. Klein, Shay Mozes, and Oren Weimann. Shortest paths in directed planar graphs with negative lengths: A linear-space $O(n \log^2 n)$-time algorithm. *ACM Transactions on Algorithms*, 6(2), 2010. doi:10.1145/1721837.1721846.

35 Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger. Architecting phase change memory as a scalable DRAM alternative. In *Proc. ACM International Symposium on Computer Architecture (ISCA)*, 2009.

36 Jagan S. Meena, Simon M. Sze, Umesh Chand, and Tseung-Yuan Tseng. Overview of emerging nonvolatile memory technologies. *Nanoscale Research Letters*, 9, 2014.

37 Suman Nath and Phillip B. Gibbons. Online maintenance of very large random samples on flash storage. *VLDB J.*, 19(1), 2010.

38 Hyoungmin Park and Kyuseok Shim. FAST: Flash-aware external sorting for mobile database systems. *Journal of Systems and Software*, 82(8), 2009. `doi:10.1016/j.jss.2009.02.028`.

39 Joon-Sang Park, Michael Penner, and Viktor K. Prasanna. Optimizing graph algorithms for improved cache performance. *IEEE Transactions on Parallel and Distributed Systems*, 15(9), 2004.

40 Mike S. Paterson. Improved sorting networks with $O(\log n)$ depth. *Algorithmica*, 5(1), 1990.

41 Moinuddin K. Qureshi, Sudhanva Gurumurthi, and Bipin Rajendran. *Phase Change Memory: From Devices to Systems.* Morgan & Claypool, 2011.

42 Jeanette P. Schmidt. All shortest paths in weighted grid graphs and its application to finding all approximate repeats in strings. *SIAM Journal on Computing*, 27, 1998.

43 Joel Seiferas. Sorting networks of logarithmic depth, further simplified. *Algorithmica*, 53(3), 2009.

44 Robert E. Tarjan. Updating a balanced search tree in $O(1)$ rotations. *Information Processing Letters*, 16(5), 1983.

45 Stratis D. Viglas. Adapting the B$^+$-tree for asymmetric I/O. In *Proc. East European Conference on Advances in Databases and Information Systems (ADBIS)*, 2012.

46 Stratis D. Viglas. Write-limited sorts and joins for persistent memory. *PVLDB*, 7(5), 2014.

47 Cong Xu, Xiangyu Dong, Norman P. Jouppi, and Yuan Xie. Design implications of memristor-based RRAM cross-point structures. In *Proc. IEEE Design, Automation and Test in Europe (DATE)*, 2011.

48 Byung-Do Yang, Jae-Eun Lee, Jang-Su Kim, Junghyun Cho, Seung-Yun Lee, and Byoung-Gon Yu. A low power phase-change random access memory using a data-comparison write scheme. In *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, 2007.

49 Yole Developpement. Emerging non-volatile memory technologies, 2013.

50 Ping Zhou, Bo Zhao, Jun Yang, and Youtao Zhang. A durable and energy efficient main memory using phase change memory technology. In *Proc. ACM International Symposium on Computer Architecture (ISCA)*, 2009.

51 Omer Zilberberg, Shlomo Weiss, and Sivan Toledo. Phase-change memory: An architectural perspective. *ACM Computing Surveys*, 45(3), 2013.

Hyperbolic Random Graphs: Separators and Treewidth

Thomas Bläsius[1], Tobias Friedrich[2], and Anton Krohmer[3]

1 **Hasso Plattner Institute, Potsdam, Germany**
thomas.blaesius@hpi.de
2 **Hasso Plattner Institute, Potsdam, Germany**
tobias.friedrich@hpi.de
3 **Hasso Plattner Institute, Potsdam, Germany**
anton.krohmer@hpi.de

Abstract

Hyperbolic random graphs share many common properties with complex real-world networks; e.g., small diameter and average distance, large clustering coefficient, and a power-law degree sequence with adjustable exponent β. Thus, when analyzing algorithms for large networks, potentially more realistic results can be achieved by assuming the input to be a hyperbolic random graph of size n. The worst-case run-time is then replaced by the expected run-time or by bounds that hold *with high probability (whp)*, i.e., with probability $1 - O(1/n)$. Though many structural properties of hyperbolic random graphs have been studied, almost no algorithmic results are known.

Divide-and-conquer is an important algorithmic design principle that works particularly well if the instance admits small separators. We show that hyperbolic random graphs in fact have comparatively small separators. More precisely, we show that they can be expected to have balanced separator hierarchies with separators of size $O(\sqrt{n^{3-\beta}})$, $O(\log n)$, and $O(1)$ if $2 < \beta < 3$, $\beta = 3$, and $3 < \beta$, respectively. We infer that these graphs have whp a treewidth of $O(\sqrt{n^{3-\beta}})$, $O(\log^2 n)$, and $O(\log n)$, respectively. For $2 < \beta < 3$, this matches a known lower bound.

To demonstrate the usefulness of our results, we give several algorithmic applications.

1998 ACM Subject Classification G.2.1 Combinatorics, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases hyperbolic random graphs, scale-free networks, power-law graphs, separators, treewidth

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.15

1 Introduction

A geometric random graph is obtained by randomly placing vertices into the plane and connecting two vertices if and only if they are close. When using the hyperbolic plane, one obtains a (threshold) hyperbolic random graph, as introduced by Krioukov et al. [21]. More precisely, vertices are placed in a disk D_R of radius R (which depends on n) and two vertices are connected if their distance is at most R. An important property of the hyperbolic plane is that the perimeter of a circle grows exponentially with the radius. Thus, when sampling uniformly, we obtain many vertices close to the boundary and few close to the center of D_R. As distances close to the boundary are larger (due to the exponentially growing perimeter), this leads to many vertices of low degree and few vertices of high degree. The resulting degree distribution actually follows a power law with exponent $\beta = 3$ [18, 28]. One can tweak this exponent using a parameter α. Choosing $1/2 \leq \alpha < 1$ increases the probability of vertices with small radius (and thus of higher degree), while for $\alpha > 1$, the vertices are shifted towards the boundary of D_R. The resulting power-law exponent is $\beta = 2\alpha + 1$.

Besides a power-law degree-distribution, hyperbolic random graphs exhibit other properties of large real-world graphs. Due to the geometric notion of closeness, vertices with a common neighbor are likely also connected, leading to a constant clustering coefficient [18]. We note that this property distinguishes hyperbolic random graphs from other models that also generate scale-free graphs (i.e., graphs with a power-law degree-distribution) such as the Chung-Lu model [7] and the Barabási-Albert model [2]. These models produce graphs that have clustering coefficient $o(1)$ [5, 29]. Beyond a non-vanishing clustering coefficient, hyperbolic random graphs have polylogarithmic diameter [15, 20] and average distance $O(\log \log n)$ between vertex pairs [1, 6], i.e., they are ultra-small-world networks. Thus, hyperbolic random graphs seem to be well suited for representing large real-world networks. This is further supported by the work of Boguñá, Papadopoulos and Krioukov [4] who embedded the internet into the hyperbolic plane and demonstrated that the resulting coordinates lead to an almost optimal greedy routing.

Despite these promising properties, we note that hyperbolic random graphs are clearly not a perfect and domain-independent representation of the real world. The degree distribution of real-world data does for example not always follow a power-law [8]. Moreover, very large cliques [14] seem unrealistic as one would expect at least a few edges to be missing. Such missing edges can be achieved by considering the so-called binomial model, in which vertices are connected with a certain probability depending on their hyperbolic distance (see Section 6). Recently, hyperbolic random graphs have been generalized to geometric inhomogeneous random graphs (GIRGs) [6]. In a GIRG, the degree distribution depends on chosen weights and is thus not necessarily fixed to a power law. Moreover, GIRGs allow for more flexibility in the choice of the underlying (potentially higher dimensional) geometry.

Divide-and-conquer algorithms separate the given instance into smaller subinstances, solve these subinstances recursively, and then combine the results to a solution of the original instance. Such an approach works well if (i) both subinstances have roughly the same size (leading to a logarithmic recursion depth); and (ii) a small interface between the subinstances allows a combination of partial solutions with only few tweaks. For graphs, one is thus interested in finding small *balanced separators*, i.e., sets of vertices whose removal separates the graph into disconnected subgraphs of roughly the same size. A famous example is the planar separator theorem by Lipton and Tarjan [24] stating that every planar graph with n vertices has a separator of size $O(\sqrt{n})$ such that both resulting subgraphs have at least $1/3n$ vertices. This for example leads to a PTAS for Independent Set on planar graphs [25].

Closely related to separators is the concept of treewidth, which is a key concept in the field of parameterized complexity as many NP-hard graph problems are actually FPT with respect to the treewidth [9, 10]. The treewidth has been intensively studied on different random graph models (all results we mention in the following hold with probability tending to 1 for $n \to \infty$). Erdős-Rényi graphs [12] have linear treewidth if the edge-vertex ratio is above $1/2$ [16, 17, 22]. This bound is sharp as the treewidth is 2 if the edge-vertex ratio is below $1/2$ [22]. For random intersection graphs [19] and for the Barabási-Albert model (which produces scale-free graphs) [2], Gao [17] gave linear lower bounds for the treewidth.

Besides these negative results, there are positive results for geometric random graphs (in the euclidean geometry). Depending on the maximum distance for which two vertices are connected, the treewidth can be shown to be $\Theta(\log n / \log \log n)$ [27] or $\Theta(r\sqrt{n})$ [23, 27]. Recently, Bringmann et al. [6] showed that a GIRG has a balanced cut of sub-linear size, which implies the same for hyperbolic random graphs.

Contribution & Outline. We present a hierarchical decomposition (the *hyperdisk decomposition*) of a disk in the hyperbolic plane into equally sized regions that have large distance from each other while the separators have small area; see Section 3. This decomposition carries over to hyperbolic random graphs leading to a hierarchy of balanced separators, each of expected size $O(n^{1-\alpha})$, $O(\log n)$, and $O(1)$ if $\alpha < 1$, $\alpha = 1$, and $\alpha > 1$, respectively.

In Section 4, we infer that hyperbolic random graphs have whp treewidth $O(n^{1-\alpha})$, $O(\log^2 n)$, and $O(\log n)$ if $\alpha < 1$, $\alpha = 1$, and $\alpha > 1$, respectively. For $\alpha < 1$, this matches the lower bound implied by the clique number of hyperbolic random graphs [14]. For $\alpha = 1$ and $\alpha > 1$, this is above the lower bounds by factors of $\log n \log\log n$ and $\log\log n$, respectively.

We demonstrate algorithmic applications in Section 5. For Independent Set, we give an approximation scheme whose algorithms have expected approximation ratio $1-\varepsilon$ ($\varepsilon > 0$) and expected polynomial run-time (in n). Choosing ε suitably, the polynomial run-time and the approximation ratio $1-O(1)/\log^{\alpha} n$ hold whp. Moreover, we show that maximum matchings in hyperbolic random graphs can whp be computed in time $O(n^{2-\alpha})$ ($O(n^{2-\alpha}\log n)$ with edge weights). As $\alpha \geq 1/2$, this improves upon the worst-case complexity of the fastest known algorithm for general graphs with run-time $O(\sqrt{n}m)$ [26]. For both results, we assume that the geometry of the graph is known, which lets us compute the hyperdisk decomposition. Otherwise, one can still apply the results by Fomin et al. [13] to obtain fast algorithms for various problems; see Section 5.3.

In Section 6, we consider the binomial model, where two vertices are connected with a certain probability depending on their distance. For $\alpha < 1$ we obtain the treewidth $O(n^{2-(\alpha+1)/(\alpha t+1)})$ (t is a constant and $t \to 0$ leads to the threshold model).

2 Preliminaries

Hyperbolic Random Graphs. We consider three parameters, the number of vertices n, the parameter $\alpha \geq 1/2$ controlling the power-law exponent, and C controlling the average degree. We obtain a hyperbolic random graph by sampling n points in the disk D_R with radius $R = 2\log n + C$. A point is described using radial coordinates (r, θ) with the center of D_R as origin. The angle θ is drawn uniformly from $[0, 2\pi]$ and the radius r is chosen according to the density $d(r) = \alpha \sinh(\alpha r)/(\cosh(\alpha R) - 1)$. Thus, the points are distributed according to the following density function (which depends on the radius r but not on the angle θ).

$$f(r,\theta) = f(r) = \frac{\alpha}{2\pi} \cdot \frac{\sinh(\alpha r)}{\cosh(\alpha R) - 1} \tag{1}$$

For $S \subseteq D_R$, the probability measure $\mu(S) = \int_S f(r)\,\mathrm{d}r$ gives the probability that a specific vertex lies in S. Note that $\mu(D_R) = 1$ and for $\alpha = 1$, $\mu(S)$ is the area of S divided by the area of D. Two vertices are connected if and only if their (hyperbolic) distance is at most R.

We are often interested in the number of vertices lying in a certain region S. To this end, let $X_i \in \{0, 1\}$ be the random variable with the interpretation that $X_i = 1$ if and only if the vertex i lies in S. Note that $\mathbb{E}[X_i] = \mu(S)$. Moreover, the random variable $X = \sum_{i=1}^{n} X_i$ describes the number of vertices in S. The following theorem directly follows from Chernoff bounds [11] and helps to give bounds that hold whp (i.e., with probability $1 - O(1/n)$).

▶ **Theorem 1.** *Let $X_1, \dots, X_n$ be independent random variables with $X_i \in \{0, 1\}$ and let X be their sum. Let $f(n) = \Omega(\log n)$. If $f(n)$ is an upper (resp. lower) bound for $\mathbb{E}[X]$, then for each constant c there is a constant c' such that $X \leq c' f(n)$ (resp. $X \geq c' f(n)$) holds with probability $1 - O(n^{-c})$.*

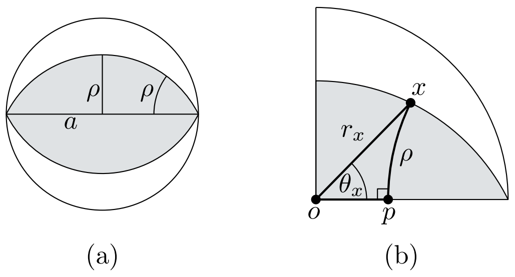


Figure 1 (a) The hyperdisk (gray) with radius ρ and horizontal axis a going through the origin. (b) A right triangle in the hyperbolic plane.

Separator Hierarchy. Let D be a metric space (e.g., a disk in hyperbolic space or a graph). A *separator hierarchy* of D is a rooted tree T with t nodes where each node i is associated with a subset $S_i \in D$ such that $\mathcal{S} = \{S_1, \dots, S_t\}$ partitions D, i.e., $D = \bigcup_{S_i \in \mathcal{S}} S_i$ and $S_i \cap S_j = \emptyset$ for $i \neq j$. For every node i, let $U_i = S_i \cup \bigcup_{j \in \mathrm{desc}(i)} S_j$, where $\mathrm{desc}(i)$ are the descendants of i in T. For $(T, \mathcal{S})$ to be a separator hierarchy, we require that for every pair of sibling nodes i and j (i.e., nodes with the same parent) the set U_i is disconnected from U_j. For the parent k of i and j, we also say that S_k is the *separator* that separates U_i from U_j.

The *diameter* of the separator S_k (separating U_i and U_j) is the largest value d such that every element in U_i has distance at least d form every element in U_j. The diameter of the separator hierarchy $(T, \mathcal{S})$ is the minimum diameter of all its separators. For a given measure μ on D, the separator S_k is *balanced* if $\mu(U_i) \leq 1/2\mu(U_k)$ and $\mu(U_j) \leq 1/2\mu(U_k)$. If these inequalities hold in expectation, we say that S_k is *expected to be balanced.* The separator hierarchy is balanced (in expectation) if each separator is balanced (in expectation).

Treewidth. Let T be a tree with t nodes and let $\mathcal{X} = \{X_1, \dots, X_t\}$ be a family of sets. For each node i of T, the set X_i is called the *bag* of i. The pair $(T, \mathcal{X})$ is a *tree decomposition* of a graph $G = (V, E)$ if the bags are subsets of V satisfying the following two properties.

(i) For each vertex $v \in V$, the nodes of T whose bags contain v induce a subtree of T.
(ii) For each edge $uv \in E$, there exist a bag $X \in \mathcal{X}$ with $u \in X$ and $v \in X$.

The *width* of a tree decomposition is the size of the largest bag minus 1. The *treewidth* $\mathrm{tw}(G)$ of a graph G is the smallest k for which G has a tree decomposition of width k.

3 Hyperdisk Decomposition

In this section, we define a separator hierarchy for a disk D in the hyperbolic plane. We assume that D is centered at the origin. The separators are (parts of) hyperdisks, which are defined as follows. Let a be a line in the hyperbolic plane. The set of points with distance ρ from a form the *hypercircle* with *radius* ρ and *axis* a. The set of points with distance at most ρ from a is the corresponding *hyperdisk*; see[1] Fig. 1a. We usually consider lines through the origin as axes. By a_γ, we denote such a line whose points have angle γ or $\gamma + \pi$.

Let $x = (r_x, \theta_x)$ be a point on the hypercircle with axis a_0 and radius ρ. Moreover, let o be the origin and let p be the point on a_γ such that the line through x and p is perpendicular to a_γ. Then o, p, and x form a triangle with right angle at p; see Fig. 1b. The angle at o

[1] We use the Poincaré disk model in illustrations. Thus, the disk shown in Fig. 1 (as well as the outer-most disk in every later illustration) is not D but the Poincaré disk representing the whole hyperbolic plane.

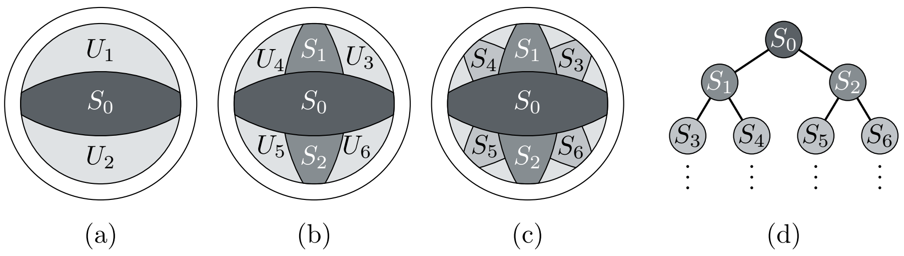


Figure 2 (a) The disk D (gray area) is separated by S_0 (dark gray) into the two regions U_1 and U_2 (light gray). (b, c) The separators on levels 1 and 2. (d) The corresponding separator hierarchy represented as tree in which each node corresponds to a separator.

is θ_x, the length of the opposite side xp is ρ, and the length of the hypotenuse is r_x. The trigonometry of hyperbolic right triangles yields the following equation, which we need later.

$$\sin(\theta_x) = \frac{\sinh(\rho)}{\sinh(r_x)} \tag{2}$$

The *hyperdisk decomposition* of D is the following separator hierarchy. The top-level separator (level 0) S_0 is the intersection of D with the hyperdisk with axis a_0 (all hyperdisks we consider have arbitrary but fixed radius ρ). This symmetrically separates D into regions U_1 and U_2 above and below S_0; see Fig. 2a. The region U_1 (and analogously U_2) is again symmetrically separated into two parts by its intersection with the hyperdisk with axis $a_{\pi/2}$. Denote the resulting separator by S_1 and the two separated regions by U_3 and U_4; see Fig. 2b. On the next level, U_3 (and analogously $U_4, \dots, U_6$) is separated by its intersection with the hyperdisk with axis $a_{\pi/4}$; see Fig. 2c. We continue this decomposition until $S_i = U_i$. Clearly, this leads to a separator hierarchy $(T, \mathcal{S})$ and T is a complete binary tree; see Fig. 2d.

3.1 Properties of the Hyperdisk Decomposition

In the following, we investigate different properties of the hyperdisk decomposition, depending on the radius R of the disk D, on the radius ρ of the hyperdisks, and on a measure μ on D. We start with two simple observations. The first observation follows from the fact that two points on different sides of a hyperdisk with radius ρ have distance at least 2ρ (they have distance ρ from the hyperdisk's axis and the line segment connecting them crosses the axis).

▶ **Observation 2.** *The hyperdisk decomposition has diameter at least* 2ρ.

We will later set $\rho = R/2$ (Section 3.2) or to something even larger (Section 6). However, the bounds we prove in this section hold for more general choices of ρ.

The next observation follows from the fact that for two nodes i and j on the same level, the regions U_i and U_j are symmetric with respect to rotation around the origin.

▶ **Observation 3.** *The hyperdisk decomposition is balanced for every measure that is invariant under rotation around the origin.*

The measure we are particularly interested in is μ given by $\mu(S) = \int_S f(r)\, \mathrm{d}r$ with the density function $f(r)$ as defined in Equation (1). Note that μ is clearly invariant under rotation around the origin as f does not depend on the angle of a given point. In the remainder of this section, we always assume μ to be this measure.

Our main goal in the following is to bound the measure of the separators in the hyperdisk decomposition. Clearly, the measure of the separators decreases for increasing level. To quantify this, we first show the following lemma.

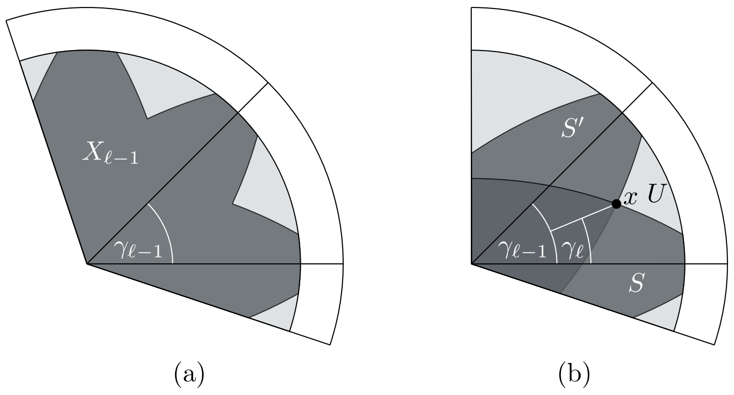


Figure 3 (a) The region $X_{\ell-1}$. (b) Two consecutive hyperdisks enclosing the region U.

▶ **Lemma 4.** *Let S be a separator on level $\ell \geq 1$ of the hyperdisk decomposition and let $x \in S$ be a point with radius r_x. Then the following holds:*

$$r_x \geq r_{\min} = \max\{\rho, \rho + \log(1 - e^{-2\rho}) - \log(2^{2-\ell} - 2^{2-2\ell})\}\,.$$

Proof. We show that the inequality holds for every point x that is not contained in a separator of level less than ℓ. Clearly, $r_x \geq \rho$ holds as every point with smaller radius is contained in the top-level separator. It remains to show $r_x \geq \rho + \log(1 - e^{-2\rho}) - \log(2^{2-\ell} - 2^{2-2\ell})$.

First assume $\ell = 1$. In this case $\log(2^{2-\ell} - 2^{2-2\ell}) = 0$. Moreover, $\log(1 - e^{-2\rho}) < 0$. Thus the claim is weaker than $r_x \geq \rho$, which we already proved above. We assume in the following that $\ell \geq 2$, which implies that there are at least two separators with level less than ℓ.

Let $\gamma_\ell = \pi/2^\ell$ and consider all hyperdisks that have an axis whose angle is a multiple of $\gamma_{\ell-1}$. Let $X_{\ell-1}$ be the union of these hyperdisks; see Fig. 3a. By the definition of the hyperdisk decomposition, the union of all separators of level less than ℓ equals $X_{\ell-1}$. We show that all points not in $X_{\ell-1}$ (and thus all points in a separator of level ℓ) satisfy the claimed inequality. To this end, first note that rotating the disc D by a multiple of $\gamma_{\ell-1}$ around the origin maps $X_{\ell-1}$ to itself. Thus, it suffices to prove the claim for points with angles between 0 and $\gamma_{\ell-1}$.

Let S and S' be the hyperdisks whose axes have angles 0 and $\gamma_{\ell-1}$, respectively, let U be the region between them, and let x be the point where S, S', and U touch; see Fig. 3b. In the following we first show that actually no point in U has radius smaller than x (which is intuitively true when looking at Fig. 3a). Afterwards it remains to show the claimed lower bound for the point x.

Let $y \in U$ be a point with coordinates (r_y, θ_y) and let ρ_y be the distance of y from the horizontal axis a_0. By Equation (2), we have $\sinh(\rho_y) = \sinh(r_y)\sin(\theta_y)$. Thus, for all relevant angles θ_y, the distance ρ_y is increasing with increasing radius and with increasing angle. Hence, in case $\theta_y \leq \theta_x$, the assumption $r_y < r_x$ implies that $\rho_y < \rho$ (recall that ρ is the distance of x from the horizontal axis). Thus, y is contained in the hyperdisk S and cannot be contained in U. Symmetrically, if $\theta_y \geq \theta_x$ and $r_y < r_x$, then y is contained in S'. Hence, no point in U has smaller radius than x.

It thus remains to show the claimed inequality for the point x. Note that x has the angle $\theta_x = \gamma_\ell$. Thus, by Equation (2), we have the following.

$$\sin(\gamma_\ell) = \frac{\sinh(\rho)}{\sinh(r_x)}$$

$$\begin{aligned}
\Leftrightarrow\qquad && \sinh(r_x) &= \frac{\sinh(\rho)}{\sin(\gamma_\ell)} \\
\Leftrightarrow\qquad && e^{r_x} - e^{-r_x} &= \frac{e^{\rho} - e^{-\rho}}{\sin(\gamma_\ell)} \\
\Rightarrow\qquad && e^{r_x} &\geq \frac{e^{\rho} - e^{-\rho}}{\sin(\gamma_\ell)} \\
\Leftrightarrow\qquad && r_x &\geq \log(e^{\rho} - e^{-\rho}) - \log(\sin(\gamma_\ell)) \\
&& &= \rho + \log(1 - e^{-2\rho}) - \log(\sin(\gamma_\ell))
\end{aligned}$$

We use that $\sin(\gamma) \leq 1 - (1 - 2/\pi \cdot \gamma)^2$ for $\gamma \in [0, \pi/2]$ to obtain the following.

$$\sin(\gamma_\ell) \leq 1 - \left(1 - \frac{2}{\pi}\gamma_\ell\right)^2 = 1 - \left(1 - 2^{1-\ell}\right)^2 = 2^{2-\ell} - 2^{2-2\ell}$$

Together with the previous inequality, this yields the claimed bound. ◀

The above lemma together with a simple calculation shows the following. With the height of the hyperdisk decomposition (which is a separator hierarchy), we refer to the height of the corresponding tree, i.e., to the maximum distance from a node to the root.

▶ **Lemma 5.** *The hyperdisk decomposition has height $O(R)$ if $\rho \geq \varepsilon$ for a constant ε.*

Proof. We show that setting $\ell = \log_2(c \cdot e^R) + 2 = O(R)$ (for a suitable constant c) results in $r_{\min} \geq R$. Thus, the separators with smaller levels already cover the whole disk of radius R. The last part of the formula given for $r_{\min}$ in Lemma 4 can be rewritten as follows.

$$-\log\left(2^{2-\ell} - 2^{2-2\ell}\right) \geq -\log\left(2^{2-\ell}\right) = \log\left(2^{\ell-2}\right) = \log(c) + R$$

When choosing $c = (1 - e^{-2\varepsilon})^{-1}$, Lemma 4 yields $r_{\min} \geq R$. ◀

In the following, we upper bound the measure of a separator S on level ℓ of the hyperdisk decomposition. Let H be the hyperdisk corresponding to S. To simplify the calculations, we rotate the disk such that a_0 (i.e., the horizontal line through the origin) is the axis of H. First assume that $\ell \geq 1$ and let $r_{\min}$ be the lower bound for the radius shown in Lemma 4. Consider the set S' of all points in H with radius at least $r_{\min}$ that lie to the right of the origin (angle between $-\pi/2$ and $\pi/2$). Clearly $S \subseteq S'$ and thus $\mu(S) \leq \mu(S')$.

To compute $\mu(S')$, let $\theta_\rho(r)$ be the angle between 0 and $\pi/2$ such that the point $(r, \theta_\rho(r))$ lies on the hypercircle bounding H. By Equation (2), we have $\theta_\rho(r) = \arcsin(\sinh(\rho)/\sinh(r))$. We obtain the following (recall that $f(r)$ is the density function).

$$\mu(S') = \int_{S'} f(r)\,\mathrm{d}r = \int_{r_{\min}}^{R} \int_{-\theta_\rho(r)}^{\theta_\rho(r)} f(r)\,\mathrm{d}\theta\,\mathrm{d}r = \int_{r_{\min}}^{R} 2\theta_\rho(r) f(r)\,\mathrm{d}r \tag{3}$$

Note that $\theta_\rho(r)$ is only well-defined if $r \geq \rho$. For $\ell \geq 1$, this is not an issue as $r_{\min} \geq \rho$ by Lemma 4. However, the case that $S = H$ is the top-level separator needs special treatment. In this case, we partition S into the disk D_ρ of radius ρ and the subsets S' and S'' of H whose points have radius at least ρ and lie to the right and left of the origin, respectively. Due to symmetry, $\mu(S') = \mu(S'')$. Thus, $\mu(S) = \mu(D_\rho) + 2\mu(S')$. Gugelmann et al. [18] showed that $\mu(D_\rho) = e^{-\alpha(R-\rho)}(1 + o(1))$. Moreover, for $\mu(S')$ we again obtain Equation (3) with $r_{\min} = \rho$. We thus obtain the following lemma by bounding $\mu(S')$ from above. Note that we require ρ to be linear in R here.

▶ **Lemma 6.** *Let S be a separator on level ℓ of the hyperdisk decomposition of D_R with hyperdisks of radius $\rho \in \Omega(R)$. Then the following holds.*

$$\mu(S) = \begin{cases} O\left(e^{\alpha\rho-\alpha R}\right) \cdot \left(2^{1-\alpha}\right)^{-\ell} & \textit{for } \alpha < 1 \\ O\left(e^{\rho-R} \cdot R\right) & \textit{for } \alpha = 1 \\ O\left(e^{\rho-R}\right) & \textit{for } \alpha > 1 \end{cases}$$

Proof. For the special case $\ell = 0$, $\mu(D_\rho) = O(e^{\alpha\rho-\alpha R})$ is dominated by the claimed bounds for all α. Thus, for all cases, it remains to prove the bounds for $\mu(S')$. Starting with Equation (3) and using that $\arcsin(x) \le x \cdot \pi/2$ (for $x \ge 0$), we get the following.

$$\begin{aligned}
\int_{r_{\min}}^{R} 2\theta_\rho(r) f(r)\,\mathrm{d}r &\le \int_{r_{\min}}^{R} 2 \cdot \frac{\pi}{2} \cdot \frac{\sinh(\rho)}{\sinh(r)} \cdot \frac{\alpha}{2\pi} \cdot \frac{\sinh(\alpha r)}{\cosh(\alpha R) - 1}\,\mathrm{d}r \\
&= \frac{\alpha}{2} \cdot \frac{\sinh(\rho)}{\cosh(\alpha R) - 1} \cdot \int_{r_{\min}}^{R} \frac{\sinh(\alpha r)}{\sinh(r)}\,\mathrm{d}r \\
&= \frac{\alpha}{2} \cdot \frac{\sinh(\rho)}{\cosh(\alpha R) - 1} \cdot \int_{r_{\min}}^{R} \frac{e^{\alpha r} - e^{-\alpha r}}{e^{r} - e^{-r}}\,\mathrm{d}r \\
&= \frac{\alpha}{2} \cdot \frac{\sinh(\rho)}{\cosh(\alpha R) - 1} \cdot \int_{r_{\min}}^{R} \frac{e^{r}}{e^{r} - e^{-r}} \cdot \frac{e^{\alpha r} - e^{-\alpha r}}{e^{r}}\,\mathrm{d}r \\
&\le \frac{\alpha}{2} \cdot \frac{\sinh(\rho)}{\cosh(\alpha R) - 1} \cdot \frac{e^{\rho}}{e^{\rho} - e^{-\rho}} \cdot \int_{r_{\min}}^{R} \frac{e^{\alpha r}}{e^{r}}\,\mathrm{d}r \\
&= O\left(e^{\rho-\alpha R}\right) \cdot \int_{r_{\min}}^{R} e^{(\alpha-1)r}\,\mathrm{d}r
\end{aligned}$$

The last inequality follows from the facts that $\frac{e^r}{e^r - e^{-r}}$ is monotonically decreasing (and thus maximal for $r = r_{\min} \ge \rho$) and that $e^{-\alpha r}$ is positive. In case $\alpha = 1$, the integral equals to $R - r_{\min} \le R$, yielding $\mu(S') = O(e^{\rho-R} \cdot R)$. Otherwise, we have the following.

$$\int_{r_{\min}}^{R} \frac{e^{\alpha r}}{e^{r}}\,\mathrm{d}r = \left[\frac{e^{(\alpha-1)r}}{\alpha - 1}\right]_{r_{\min}}^{R} = \frac{e^{(\alpha-1)R} - e^{(\alpha-1)r_{\min}}}{\alpha - 1}$$

If $\alpha > 1$, the integral is dominated by $e^{(\alpha-1)R}$, yielding $\mu(S') = O(e^{\rho-\alpha R} \cdot e^{\alpha R - R}) = O(e^{\rho-R})$. If $\alpha < 1$, the integral is dominated by $e^{(\alpha-1)r_{\min}}$. Using the bound for $r_{\min}$ given by Lemma 4, we obtain the following.

$$\begin{aligned}
\mu(S') &= O\left(e^{\rho-\alpha R}\right) \cdot e^{(\alpha-1)r_{\min}} \\
&= O\left(e^{\rho-\alpha R}\right) \cdot e^{(\alpha-1)\rho} \cdot \left(1 - e^{-2\rho}\right)^{\alpha-1} \cdot \left(2^{2-\ell} - 2^{2-2\ell}\right)^{1-\alpha}
\end{aligned}$$

In this product, the first two factors simplify to $O(e^{\alpha\rho-\alpha R})$. The third factor tends to 1 for increasing ρ (and $\rho \in \Omega(R)$). The fourth factor can be written as $2^{-\ell(1-\alpha)}(2^2 - 2^{2-\ell})^{1-\alpha}$. As $(2^2 - 2^{2-\ell})^{1-\alpha}$ is bounded by the constant $4^{1-\alpha}$, we obtain the claimed bound. ◀

3.2 Decomposing Hyperbolic Random Graphs

Recall that one obtains a hyperbolic random graph G by randomly placing n vertices in the disk D of radius $R = 2\log n + C$ according to the probability measure μ and connecting two vertices if and only if their distance is less than R. For a subset $S \subseteq D$, let $G[S]$ be the subgraph of G induced by vertices in S.

Let $(T, \mathcal{S})$ be the hyperdisk decomposition of D with hyperdisks of radius $\rho = R/2$. Let further k be a node of T with children i and j. By Observation 2, the diameter of the separator S_k (separating U_i from U_j) is at least $2\rho = R$. This implies that no vertex in U_i is connected to a vertex in U_j. Thus, the graph $G[U_k]$ is separated by the vertices in S_k into the subgraphs $G[U_i]$ and $G[U_j]$. Hence, the hyperdisk decomposition of D translates into a separator hierarchy of the graph G. We also call this separator hierarchy the *hyperdisk decomposition* of G. The properties of the separators in D directly translate to the separators in G, i.e., we can expect the separators to be balanced (Observation 3) and small (Lemma 6).

▶ **Theorem 7.** *The hyperdisk decomposition of a hyperbolic random graph is expected to be balanced and for a separator S on level ℓ, the following holds.*

$$\mathbb{E}\left[|S|\right] = \begin{cases} O\left(n^{1-\alpha}\right) \cdot \left(2^{1-\alpha}\right)^{-\ell} & \text{for } \alpha < 1 \\ O\left(\log n\right) & \text{for } \alpha = 1 \\ O\left(1\right) & \text{for } \alpha > 1 \end{cases}$$

4 The Treewidth of Hyperbolic Random Graphs

The treewidth of a graph G is closely related to the size of separators in G. If G has treewidth k, it is known to have a balanced separator of size $k+1$. This follows from the fact that a tree (e.g., the decomposition tree of width k) has a (weighted) balanced separator of size 1. Conversely, if G can be recursively decomposed by small balanced separators, i.e., if it has a balanced separator hierarchy with small separators, its treewidth is also small.

The following statement is easy to prove and for example used to show that planar graphs have treewidth $\sqrt{n}$ based on the planar separator theorem.

▶ **Lemma 8.** *Let $(T, \mathcal{S})$ be a separator hierarchy of G. For each node i of T, let X_i be the union of S_i and all separators S_j for which j is an ancestor of i in T. Then $(T, \mathcal{X} = \{X_1, \dots, X_t\})$ is a tree decomposition of G.*

Proof. We have to show that for each vertex $v \in V$ the bags containing v form a subtree of T and that for each edge $uv \in E$, there exists a bag containing u and v. For the former, let v be a vertex and let i be the unique node of T such that $v \in S_i$. Then $v \in X_j$ if and only if $j = i$ or j is an descendant of i. The node i together with its descendants clearly is a subtree of T. For the second condition, let $uv \in E$, let $u \in S_i$ and $v \in S_j$. Due to their edge, u and v cannot be separated, which implies that i is an ancestor of j or vice versa. Without loss of generality, we assume i is an ancestor of j. Then X_j includes S_j and S_i, which proves the claim. ◀

Using the properties of the hyperdisk decomposition as stated in Lemma 5 and Theorem 7 together with Lemma 8, we obtain a tree decomposition with upper bounds on the expected size of each bag. Applying a Chernoff bound (Theorem 1) leads to the following theorem.

▶ **Theorem 9.** *For the treewidth* $\operatorname{tw}(G)$ *of a hyperbolic random graph* G*, the following holds with high probability.*

$$\operatorname{tw}(G) = \begin{cases} \Theta\left(n^{1-\alpha}\right) & \text{for } \alpha < 1 \\ O\left(\log^2 n\right) & \text{for } \alpha = 1 \\ O\left(\log n\right) & \text{for } \alpha > 1 \end{cases}$$

Note that the theorem states a matching lower bound if $\alpha < 1$. It follows from the fact that hyperbolic random graphs have clique number $\Theta(n^{1-\alpha})$ if $\alpha < 1$ [14]. For $\alpha \geq 1$, the clique number is $\Theta(\log n / \log\log n)$ [14]. Thus, our upper bounds for $\alpha = 1$ and $\alpha > 1$ differ from this lower bound by factors $\log n \log\log n$ and $\log\log n$, respectively.

5 Applications

Our results from the previous sections have several algorithmic implications. In particular, the logarithmic treewidth for $\alpha > 1$ leads to efficient algorithms for numerous NP-hard problems, e.g., Vertex Cover, Independent Set, Dominating Set, Odd Cycle Traversal, and Max Cut [9]. We note that the size of the largest connected component in a hyperbolic random graph is whp polynomial even for $\alpha > 1$ as the maximum degree [18] is a lower bound for the size of the largest component. Thus, for $\alpha > 1$, the treewidth is not only logarithmic in the size of the whole (potentially disconnected) hyperbolic random graph but also in the size of the largest component.

For $\alpha < 1$, the separators are larger and thus algorithmic applications are less obvious. Moreover, there exists a *giant component* [3], i.e., a connected component of linear size. In the following sections, we present several algorithmic applications for the case that G is the giant component of a hyperbolic random graph with $\alpha < 1$. We note that all results still hold when considering the whole hyperbolic random graph instead of only the giant component (in fact, some arguments actually get simpler).

We give an approximation scheme for Independent Set (Section 5.1) and a fast algorithm for computing maximum matchings (Section 5.2). Both results assume that the geometry of the hyperbolic random graph is known (which is a strong but not completely unreasonable assumption [4]). In Section 5.3 we give applications that do not rely on knowing the geometry.

5.1 An Approximation Scheme for Independent Set

As an independent set forms a clique in the complement graph and vice versa, it is NP-hard to approximate Independent Set with approximation ratio better than $O(n^{1-\varepsilon})$ for any $\varepsilon > 0$ [30]. However, based on the planar separator theorem (stating that planar graphs have balanced separators of size $O(\sqrt{n})$), Lipton and Tarjan [25] showed that Independent Set on planar graphs has a *PTAS (polynomial-time approximation scheme)*, i.e., for every constant $\varepsilon > 0$, it admits an efficient approximation algorithm with approximation ratio $1 - \varepsilon$. We adapt their approach to show that there is an approximation scheme for Independent Set if the input is a hyperbolic random graph given in its geometric representation.

The PTAS for planar graphs is based on two facts. First, planar graphs have a balanced separator hierarchy with separators of size $O(\sqrt{n})$. Second, planar graphs have independent sets of linear size. The latter follows directly from the fact that planar graphs have bounded chromatic number. For hyperbolic random graphs, this is not true, as they include comparatively large cliques [14]. However, as the degree sequence follows a power law, we find a subgraph of linear size whose vertices have bounded degree. This subgraph can then be colored with a constant number of colors which implies a large independent set. To obtain the following lemma, we have to apply this argument to the giant component of a hyperbolic random graph.

▶ **Lemma 10.** *The giant component of a hyperbolic random graph has whp independent sets of linear size.*

Following the approach of Lipton and Tarjan [25], we prove the following lemma. The rough idea is to choose V' to be the union of all separators of the hyperdisk decomposition with level at most $\lfloor \log_2(n/k) \rfloor$.

▶ **Lemma 11.** *Let $G = (V, E)$ be a hyperbolic random graph with $\alpha < 1$ and let $k \in \mathbb{N}$. Then there is a vertex set $V' \subseteq V$ of expected size $O(n)/k^\alpha$ such that the connected components of $G - V'$ have expected size at most k.*

Proof. Consider the hyperdisk decomposition $(T, \mathcal{S})$ of G and let i be a node on level ℓ. Recall that U_i denotes the set of vertices such that $G[U_i]$ is the subgraph whose separators are represented by the descendants of i in T. Due to the fact that the hyperdisk decomposition is balanced (Theorem 7), the expected size of U_i is at most $n2^{-\ell}$. We choose V' to be the union of all separators with level at most $\ell_{\max} = \lfloor \log_2(n/k) \rfloor$. Each connected component of $G - V'$ is then a subgraph of $G[U_i]$ for a vertex i with level $\ell_{\max} + 1$. Thus, the expected size of these components is at most k.

It remains to bound $\mathbb{E}[|V'|]$. Recall from Theorem 7 that a separator on level ℓ has expected size $O(n^{1-\alpha}) \cdot 2^{-\ell(1-\alpha)}$. Moreover, as T is a complete binary tree, there are 2^ℓ separators on level ℓ. Thus, the total size of separators on level ℓ is $O(n^{1-\alpha}) \cdot 2^{-\ell(1-\alpha)} 2^\ell = O(n^{1-\alpha}) \cdot 2^{\alpha\ell}$. We thus obtain the following for the expected size of V'.

$$\begin{aligned}\mathbb{E}\left[|V'|\right] &= \sum_{\ell=0}^{\ell_{\max}} O\left(n^{1-\alpha}\right) \cdot 2^{\alpha\ell} \\ &= O\left(n^{1-\alpha}\right) \cdot \sum_{\ell=0}^{\ell_{\max}} 2^{\alpha\ell}\end{aligned}$$

To conclude the proof, it remains to proof that the sum equals $(n/k)^\alpha \cdot O(1)$, which follows from the following calculation and from the fact that the geometric series converges.

$$\begin{aligned}\sum_{\ell=0}^{\ell_{\max}} 2^{\alpha\ell} &= \sum_{i=0}^{\ell_{\max}} 2^{\alpha(\ell_{\max}-i)} \\ &= 2^{\alpha\ell_{\max}} \sum_{i=0}^{\ell_{\max}} 2^{-\alpha i} \\ &= \left(n/k\right)^{\alpha} \cdot O(1)\end{aligned}$$ ◀

To approximate Independent Set, one can apply Lemma 11 to the given hyperbolic random graph G and then compute for each of the resulting connected components an optimal independent set in expected time $O(k2^k)$. For all $O(n/k)$ connected components, this takes expected $O(n2^k)$ time. The union of these independent sets is an independent set of G. Let I be this independent set, restricted to the giant component H of G. Comparing this to the size of an optimal independent set $I^\star$ of H, we miss at most the vertices in the separator V', thus $\mathbb{E}[|I^\star| - |I|] = O(n)/k^\alpha$. As $|I^\star|$ is linear with high probability (Lemma 10), dividing by $|I^\star|$ yields $\mathbb{E}[1 - |I|/|I^\star|] = k^{-\alpha} \cdot O(1)$ and thus $\mathbb{E}[|I|/|I^\star|] = 1 - k^{-\alpha} \cdot O(1)$.

This directly implies the claimed approximation scheme: for every given $\varepsilon > 0$, one can choose the k such that the expected approximation ratio is $1 - \varepsilon$. As the k we have to chose does not depend on n, the resulting running time is polynomial in n (but exponential in $1/\varepsilon$). Additionally applying concentration bounds if $k = \log n$ yields the following theorem.

▶ **Theorem 12.** *For the giant components of hyperbolic random graphs given in their geometric representation,* Independent Set *can be approximated in expected $O(n2^k)$ time with expected approximation ratio $1 - k^{-\alpha} \cdot O(1)$. If $k = \log n$, the algorithm runs whp in polynomial time and the bound on the approximation ratio holds whp.*

Proof. It remains to consider the case $k = \log n$. We apply Lemma 11 with $k = \log n$ leading to connected components of expected size $\log n$. Using a Chernoff bound (Theorem 1) shows that the size of the connected components is whp at most $c \log n$ for a constant c. Thus, the algorithm described above has whp polynomial run-time.

Concerning the approximation ratio, note that the separator V' has expected size $O(n)/\log^\alpha n$. Thus, there are constants c_1 and $n_{\min}$ such that the $\mathbb{E}[|V'|] \leq c_1 n/\log^\alpha n$ if $n > n_{\min}$. As we can brute-force smaller instances, we can assume the latter assumption to be true. Applying a Chernoff bound (Theorem 1) we get that V' includes whp at most $c_2 n/\log^\alpha n$ vertices for another constant c_2. As before let I be the independent set we computed and let $I^\star$ be an optimal independent set of the giant component. Then, $|I^\star| - |I| \leq c_2 n/\log^\alpha n$ holds whp. As whp $|I^\star| \geq c_3 n$ (at least for sufficiently large instances; see Lemma 10), the calculations from above show that $|I|/|I^\star| \geq 1 - c/\log^\alpha n$ holds whp for $c = c_2/c_3$. ◀

5.2 Computing Matchings in $O(n^{2-\alpha})$ Time

Lipton and Tarjan [25] also gave $O(n^{3/2})$ and $O(n^{3/2} \log n)$ algorithms that compute matchings of maximum cardinality and matchings of maximum weight, respectively, in a planar graph. Their algorithm uses the following divide-and-conquer strategy. Find a separator, recursively compute maximum matchings for both subgraphs, and finally combine these solutions by iteratively adding the vertices of the separator while maintaining a maximum matching. The latter can be done by finding a single augmenting path, which can be done in $O(m)$ and $O(m \log n)$ time for unweighted and weighted graphs, respectively (m is the number of edges).

To obtain the following theorem, we apply this divide-and-conquer strategy to hyperbolic random graphs, show that m is actually linear in n (even for the subgraphs for which we compute the augmenting paths), and apply concentration bounds.

▶ **Theorem 13.** *Let G be the giant component of a hyperbolic random graph given in its geometric representation. Whp, a maximum matching in G can be computed in $O(n^{2-\alpha})$ and $O(n^{2-\alpha} \log n)$ time if G is unweighted and weighted, respectively.*

5.3 In Case the Geometry is Unknown

The two previous applications relied on the fact that we know the geometric representation of the given hyperbolic random graph. However, even if the geometry is unknown, we can still benefit from the knowledge that hyperbolic random graphs have low treewidth by using an algorithm of Fomin et al. [13]. It takes a graph G and an integer k as input and either decides that $\operatorname{tw}(G) > k$ or returns a tree decomposition of width k^2. It runs in $O(k^7 n \log n)$

time. By spending an additional factor of $\log(\mathrm{tw}(G))$ we can actually compute the smallest k for which the algorithm succeeds to compute a tree decomposition.

Given a tree decomposition of width k, Fomin et al. [13] show (among other algorithms) how to compute a matching with maximum weight and a maximum vertex flow in a directed graph in $O(k^4 n \log^2 n)$ and $O(k^2 n \log n)$ time, respectively. For sufficiently large α, this leads to algorithms solving these problems faster than the best known algorithms for general graphs. E.g., for $\alpha = 31/32$, the treewidth of G is $O(n^{1/32})$ with high probability (Theorem 9). Thus, we get a tree decomposition of width $O(n^{1/16})$ in $O(n^{7/32} \cdot n \log^2 n)$ time. Computing a matching of maximum weight then takes $O(n^{1/4} \cdot n \log^2 n)$. Thus, the overall running time is $O(n^{5/4} \log^2 n)$. Also note that this approach leads to almost linear (i.e., linear up to a polylogarithmic factor) run-times if $\alpha \geq 1$.

6 Binomial Hyperbolic Random Graphs

So far, we considered the so-called *threshold* model of hyperbolic random graphs where vertices are connected if and only if they have distance at most R. A more realistic (but technically more difficult) model is the *binomial* model, in which longer edges and shorter non-edges are allowed with a certain probability. More precisely, two vertices with distance d are connected with the following probability $p(d)$ depending on the constant t (usually $0 < t < 1$). Note that we obtain the threshold model for $t \to 0$.

$$p(d) = \left(1 + e^{\frac{1}{2t}(d-R)}\right)^{-1}$$

As before, we start with a hyperdisk decomposition $(T, \mathcal{S})$ of the disk D and then transfer it to a separator hierarchy of a hyperbolic random graph. In the threshold model, separators in $(T, \mathcal{S})$ translated to separators in the graph if $\rho \geq R/2$. This is not true in the binomial model as vertices with distance greater than R are still connected with a certain probability.

However, we obtain separators as follows. Let k be a node of T with children i and j, i.e., S_k separates U_k into U_i and U_j. An edge of the graph G is *critical* with respect to S_k if it connects a vertex located in U_i with a vertex in U_j. As before, let $G[U_k]$ be the graph induced by vertices in U_k. Then the vertices located in S_k together with the endvertices of critical edges separate $G[U_k]$ into (subgraphs of) $G[U_i]$ and $G[U_j]$. In this way, we again obtain a separator hierarchy for G, which we call the *extended hyperdisk decomposition.*

To bound the size of the resulting separators, we have to bound the number of vertices in S_k and the number of critical edges. For the former, we can use the previous results (in particular Lemma 6). For the latter, note that the expected number of vertices in U_i is at most $n2^{-\ell}$ if i has level ℓ (as the hyperdisk decomposition is balanced). As the same holds for U_j, there are only $(n2^{-\ell})^2$ vertex pairs that potentially form critical edges, each with a probability of at most $p(2\rho)$ (as their distance is at least 2ρ). Thus, the expected number of critical edges is at most $n^2 p(2\rho) 2^{-2\ell}$. Plugging a carefully chosen value for ρ into this formula as well as into the formula given by Lemma 6 leads to the following theorem.

▶ **Theorem 14.** *Let G be a binomial hyperbolic random graph with $\alpha < 1$ and consider its extended hyperdisk decomposition with hyperdisks of radius $\rho = \frac{2\alpha t + t + 1}{2\alpha t + 2} R$. For a separator S on level ℓ, the following holds.*

$$\mathbb{E}(|S|) = O\left(n^{2 - \frac{\alpha+1}{\alpha t + 1}}\right) \cdot \left(2^{1-\alpha}\right)^{-\ell}$$

Proof. As mentioned above, the number of critical edges is bounded by the following term.

$$n^2 p(2\rho) 2^{-2\ell} = n^2 p\left(\frac{2\alpha t + t + 1}{\alpha t + 1} R\right) 2^{-2\ell}$$

$$
\begin{aligned}
&= n^2 \left(1 + e^{\frac{1}{2t}\left(\frac{2\alpha t+t+1}{\alpha t+1}R - R\right)}\right)^{-1} 2^{-2\ell} \\
&= n^2 \left(1 + e^{\frac{1}{2}\frac{\alpha+1}{\alpha t+1}R}\right)^{-1} 2^{-2\ell} \\
&= n^2 \left(1 + e^{\frac{1}{2}\frac{\alpha+1}{\alpha t+1}(2\log n + C)}\right)^{-1} 2^{-2\ell} \\
&\le n^2 \left(1 + e^{\frac{\alpha+1}{\alpha t+1}(\log n + C/2)}\right)^{-1} 2^{-(1-\alpha)\ell} \\
&= O\left(n^{2-\frac{\alpha+1}{\alpha t+1}}\right) \cdot \left(2^{1-\alpha}\right)^{-\ell}
\end{aligned}
$$

For the second part, we go one step back and assume that S is a separator in the hyperdisk decomposition of the disk D_R in the hyperbolic plane (instead of a separator in graph). By Lemma 6, we get the following bound on the measure of S.

$$
\begin{aligned}
\mu(S) &= O\left(e^{\alpha\rho - \alpha R}\right)\left(2^{1-\alpha}\right)^{-\ell} \\
&= O\left(e^{\alpha\frac{2\alpha t+t+1}{2\alpha t+2}R - \alpha R}\right)\left(2^{1-\alpha}\right)^{-\ell} \\
&= O\left(e^{\alpha\frac{t-1}{2\alpha t+2}R}\right)\left(2^{1-\alpha}\right)^{-\ell} \\
&= O\left(e^{\alpha\frac{t-1}{2\alpha t+2}(2\log n + C)}\right)\left(2^{1-\alpha}\right)^{-\ell} \\
&= O\left(n^{\frac{\alpha t-\alpha}{\alpha t+1}}\right)\left(2^{1-\alpha}\right)^{-\ell} \\
&= O\left(n^{\frac{\alpha t+1-\alpha-1}{\alpha t+1}}\right)\left(2^{1-\alpha}\right)^{-\ell} \\
&= O\left(n^{1-\frac{\alpha+1}{\alpha t+1}}\right)\left(2^{1-\alpha}\right)^{-\ell}
\end{aligned}
$$

Multiplying with n (as we have n vertices) leads to the claimed bound. ◀

Note that this bound coincides with our result for the threshold model when $t \to 0$. Moreover, for $t \in (0, 1)$, we obtain separators of sublinear size. As for the threshold model, we can use Lemma 8 to obtain bounds for the treewidth (compare Section 4).

7 Conclusion

We have shown that hyperbolic random graphs have small separators, as well as a small treewidth (with a phase transition from polynomial to logarithmic at $\beta = 3$). This stands in stark contrast to other popular random graph models like Erdős-Rényi [12] or Barabási-Albert [2] that have linear separators [17]. Beyond providing new insights on the structural properties of hyperbolic random graphs, our results give rise to several algorithmic applications.

To judge the practical merit of these algorithms, an interesting next step is therefore to compare separators on real graphs with predictions made by the various models. It is, however, a challenge to compute small separators in a given massive graph. Depending on the precise problem formulation, this is likely to be an NP-complete problem.

A more theoretical open question is whether our results are tight or can be improved to achieve even smaller separators. Of interest is especially the binomial model, since it allows for long range edges in the graph; and there exists no lower bound on the treewidth in the current literature.

Acknowledgements. We thank the anonymous reviewers for their useful comments.

References

1 Mohammed Amin Abdullah, Michel Bode, and Nikolaos Fountoulakis. Typical distances in a geometric model for complex networks. *CoRR*, abs/1506.07811:1–33, 2015.

2 Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

3 Michael Bode, Nikolaos Fountoulakis, and Tobias Müller. On the largest component of a hyperbolic model of complex networks. *The Electronic Journal of Combinatorics*, 22(3):1–46, 2015.

4 Marián Boguñá, Fragkiskos Papadopoulos, and Dmitri Krioukov. Sustaining the internet with hyperbolic mapping. *Nature Communications*, 1(62), 2010.

5 Béla Bollobás and Oliver M. Riordan. *Mathematical Results on Scale-Free Random Graphs*, chapter 1, pages 1–34. Wiley, 2005.

6 Karl Bringmann, Ralph Keusch, and Johannes Lengler. Geometric inhomogeneous random graphs. *CoRR*, abs/1511.00576:1–42, 2015.

7 Fan Chung and Linyuan Lu. The average distance in a random graph with given expected degrees. *Internet Mathematics*, 1(1):91–113, 2003.

8 Aaron Clauset, Cosma Rohilla Shalizi, and M. E. J. Newman. Power-law distributions in empirical data. *SIAM Review*, 51(4):661–703, 2009.

9 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer International Publishing, 2015.

10 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer-Verlag New York, 1999.

11 Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2012.

12 P. Erdős and A. Rényi. On random graphs. I. *Publicationes Mathematicae*, 6:290–297, 1959.

13 Fedor V. Fomin, Daniel Lokshtanov, Michał Pilipczuk, Saket Saurabh, and Marcin Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. *CoRR*, abs/1511.01379:1–44, 2015.

14 Tobias Friedrich and Anton Krohmer. Cliques in hyperbolic random graphs. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM'15)*, pages 1544–1552, 2015.

15 Tobias Friedrich and Anton Krohmer. On the diameter of hyperbolic random graphs. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP'15)*, pages 614–625, 2015.

16 Yong Gao. On the threshold of having a linear treewidth in random graphs. In *Proceedings of the 12th Annual International Conference on Computing and Combinatorics (COCOON'06)*, pages 226–234, 2006.

17 Yong Gao. Treewidth of Erdős-Rényi random graphs, random intersection graphs, and scale-free random graphs. *Discrete Applied Mathematics*, 160(4–5):566–578, 2012.

18 Luca Gugelmann, Konstantinos Panagiotou, and Ueli Peter. Random hyperbolic graphs: Degree sequence and clustering. In *Proceedings of the 39th International Colloquium on Automata, Languages, and Programming (ICALP'12)*, pages 573–585, 2012.

19 Michał Karoński, Edward R. Scheinerman, and Karen B. Singer-Cohen. On random intersection graphs: The subgraph problem. *Combinatorics, Probability and Computing*, 8(1–2):131–159, 1999.

20 Marcos Kiwi and Dieter Mitsche. A bound for the diameter of random hyperbolic graphs. In *Proceedings of the 12th Workshop on Analytic Algorithmics and Combinatorics (ANALCO'15)*, pages 26–39, 2015.

21 Dmitri Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguñá. Hyperbolic geometry of complex networks. *Physical Review E*, 82:036106, 2010.

22 Choongbum Lee, Joonkyung Lee, and Sang il Oum. Rank-width of random graphs. *Journal of Graph Theory*, 70(3):339–347, 2012.

23 Anshui Li and Tobias Müller. On the treewidth of random geometric graphs and percolated grids. Manuscript, 2015.

24 Richard J. Lipton and Robert E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

25 Richard J. Lipton and Robert E. Tarjan. Applications of a planar separator theorem. *SIAM Journal on Computing*, 9(3):615–627, 1980.

26 Silvio Micali and Vijay V. Vazirani. An $o(\sqrt{|V|}|e|)$ algorithm for finding maximum matching in general graphs. In *Proceedings of the 21st Annual Symposium on Foundations of Computer Science (FOCS'80)*, pages 17–27, 1980.

27 Dieter Mitsche and Guillem Perarnau. On the treewidth and related parameters of random geometric graphs. In *Proceedings of the 29th International Symposium on Theoretical Aspects of Computer Science (STACS'12)*, pages 408–419, 2012.

28 Fragkiskos Papadopoulos, Dmitri Krioukov, Marián Boguñá, and Amin Vahdat. Greedy forwarding in dynamic scale-free networks embedded in hyperbolic metric spaces. In *Proceedings of the 29th Conference on Information Communications (INFOCOM'10)*, pages 2973–2981, 2010.

29 Remco van der Hofstad. Random graphs and complex networks. Vol. II. `http://www.win.tue.nl/~rhofstad/NotesRGCNII.pdf`, 2014.

30 David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC'06)*, pages 681–690, 2006.

Efficient Embedding of Scale-Free Graphs in the Hyperbolic Plane

Thomas Bläsius[1], Tobias Friedrich[2], Anton Krohmer[3], and Sören Laue*[4]

1 **Hasso Plattner Institute, Potsdam, Germany**
`thomas.blaesius@hpi.de`
2 **Hasso Plattner Institute, Potsdam, Germany**
`tobias.friedrich@hpi.de`
3 **Hasso Plattner Institute, Potsdam, Germany**
`anton.krohmer@hpi.de`
4 **Friedrich Schiller University, Jena, Germany**
`soeren.laue@uni-jena.de`

Abstract

Hyperbolic geo metry appears to be intrinsic in many large real networks. We construct and implement a new maximum likelihood estimation algorithm that embeds scale-free graphs in the hyperbolic space. All previous approaches of similar embedding algorithms require a runtime of $\Omega(n^2)$. Our algorithm achieves quasilinear runtime, which makes it the first algorithm that can embed networks with hundreds of thousands of nodes in less than one hour. We demonstrate the performance of our algorithm on artificial and real networks. In all typical metrics like Log-likelihood and greedy routing our algorithm discovers embeddings that are very close to the ground truth.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases hyperbolic random graphs, embedding, power-law graphs, hyperbolic plane

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.16

1 Introduction

The study and analysis of complex real-world networks is a rapidly growing field. There are a number of commonly observed properties of complex networks like power-law degree distribution, small clustering coefficient, and small average distances. During the last decade, dozens of models for such *scale-free networks* have been proposed. The most popular model is the preferential attachment model by Barabási and Albert [5]. Most accessible for mathematical analysis is the inhomogeneous random graph model by van der Hofstad [33], which generalizes the models of Chung and Lu [10, 1, 2] and Norros and Reittu [26].

All aforementioned network models observe a power-law degree distribution, small diameter and average distances. However, all of them naturally also have a *small clustering coefficient*, that is, the number of triangles and small cliques in such artificial networks is magnitudes lower than observed in real-world networks. The reason is that in the standard definitions of these network models, the edges are (merely) independent, which is not true

* Sören Laue acknowledges the support of Deutsche Forschungsgemeinschaft (DFG) under grant GI-711/5-1 within the priority program "Algorithms for Big Data".

for real-world networks. For social networks the reason is easy to see: If someone is friends with two people, it is likelier that they know each other as well than it would be for two random strangers to forge a connection. There are a number of modifications of above models that incorporate this intuition [34, 25, 23], however, all these fixes introduce other artificial artifacts and can not explain *why* the clustering occurs in the first place.

Hyperbolic Random Graphs. A natural definition of a scale-free network model with all aforementioned properties emerges when adding an appropriate geometry. It is well studied that geometric random graphs with an Euclidean space result in a Poisson degree distribution [30]. Krioukov et al. [20] took a different approach by assuming an underlying *hyperbolic geometry* to the network. The most prominent feature of a hyperbolic space is its exponential expansion around a given point, in contrast to Euclidean space, which expands only polynomially. *Hyperbolic random graphs* are obtained by placing all nodes in the hyperbolic plane, and connecting two nodes whenever they are a small (hyperbolic) distance apart. The desired clustering then naturally emerges as a reflection of the geometric proximity. This model has been analyzed to have a power-law degree distribution and high clustering [16, 20], to have a polylogarithmic diameter and ultra-short average distances of order $\mathcal{O}(\log\log n)$ [15, 9], and allows fast bootstrap percolation [19].

Generating Hyperbolic Random Graphs. With most fundamental structural properties of hyperbolic random graphs settled, the next step is studying algorithms on the network model. The first addressed algorithmic problem is efficiently *generating* such a graph or, equivalently, *sampling* a graph from the probability distribution defined by hyperbolic random graphs. The naive generation of a hyperbolic random graph takes $\Theta(n^2)$ time [3]. Using a polar quadtree adapted to hyperbolic space, von Looz et al. [36] achieved a time complexity of $\mathcal{O}((n^{3/2} + m) \log n)$; and by a more sophisticated partitioning of the space, Bringmann et al. [9] obtained an optimal expected linear runtime for generation, which is crucial for large-scale experiments.

Visualizing Data in Hyperbolic Geometry. It is well known in the visualization community that hierarchical or tree-like structures can be well represented in a hyperbolic space [32]. There are three approaches to embed a network in the hyperbolic space:

- A popular way to obtain hyperbolic coordinates for the nodes of a network is embedding a spanning tree of the network in hyperbolic space [38, 37, 24]. As trees can be embedded perfectly, this is a very efficient way to map a network and has been used for interactive network browsers, which allow assigning more display space to the interesting portions of a network [21, 22]. The result might reduce visual clutter and help focus, but it ignores most structural details of the network. Nodes which are close in graph distance are not necessarily close in hyperbolic space. In fact, clusters and most local structures are not preserved.
- Another approach is determining shortest path distances and finding an embedding where metric distances match the graph distances. Computing the all-pair-shortest-path matrix can be done with the well established Euclidean data analysis method Multidimensional Scaling (MDS) [13], which has been translated to hyperbolic geometry [12]. Due to the quadratic size of the distance matrix, this approach only works for graphs with a few hundred nodes [4]. To reduce the runtime, it is possible to (randomly) select a small subset of the pairwise distances [31, 35, 42].

- Our objective is slightly different. Instead of preserving distances between nodes, we aim at inferring the *popularity* (reflected by radial coordinates) and *similarity* (reflected by angular coordinates) of all nodes [28]. The reason why a connection between vertices exist can be twofold: Either, the two vertices are similar, which holds e. g. for close friends in social networks; or for geographically close ASs in the Internet graph. On the other hand, a connection may be present due to the popularity of one end vertex: For instance, many people follow Lady Gaga on Twitter; but most are arguably not very similar to her. Embedded shortest path distances lose this information. Our goal is to recover this information using the most likely embedding assuming a hyperbolic nature of the graph in the first place. For this, we use the random network model of Krioukov et al. [20].

Maximum Likelihood Estimation Embedding of Graphs in Hyperbolic Space. We focus on the last-mentioned approach of maximum likelihood estimation (MLE) algorithms, i.e., we want to find the node coordinates in the network by maximizing the probability that the network is produced by some underlying hyperbolic model. Boguñá et al. [8] were the first to find such an embedding for the Internet graph ($m = 58\,416$ connections between $n = 23\,752$ autonomous systems) in the hyperbolic space. It is impressive that greedy navigation along these hyperbolic coordinates is almost maximally efficient, i.e., it almost always finds the shortest paths between almost any two pairs of vertices in the same component. However, the described method to discover the hyperbolic coordinates "require[s] substantial manual intervention and do[es] not scale to large networks" [20]. A general algorithm for embedding a network in a hyperbolic space was later presented by Papadopoulos et al. [29]. Their HyperMap algorithm is an approximate maximum likelihood estimation (MLE) algorithm. They demonstrate their algorithm on synthetic networks with $n = 5\,000$ nodes and $m = 20\,000$ edges and a subset of the aforementioned Internet graph with $n = 8\,220$ nodes. The asymptotic runtime was improved in a subsequent paper from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$ [27]. The authors present no runtime measurements [29, 27], but their HyperMap code on our machine requires more than 1.5 hours for a graph of size 2 000 (cf. Section 6.2). The algorithm was further refined in [39], who use a community detection algorithm for the coarse layout of the nodes; and an MLE to find precise positions. While their runtime is still $\Omega(n^2)$, our techniques extend to their case.

Our New Hyperbolic Embedder. We design and implement a new algorithm for computing hyperbolic MLE embeddings of massive networks (Section 5).[1] Compared to previous approaches that need $\Omega(n^2)$ runtime, our algorithm runs in quasilinear runtime. To this end, we developed several new techniques. First, we use an analytical approach to compute the expected angles between pairs of high-degree nodes based on their number of common neighbors. In contrast to [27], this approach does not rely on expensive numerical computations, making it fast in practice. The resulting angle distance matrix is then fed to a spring embedder that finds good positions for high-degree nodes in linear time. For small degree nodes, we substantially improve runtime by using the geometric data structure of Bringmann et al. [9] that allows traversing nodes of close proximity in expected amortized constant time.

This enables us to embed significantly larger graphs than before. For instance, we computed in under one hour a hyperbolic embedding of the Amazon product recommendation network which has over 300 000 nodes. To evaluate the quality of our embedding, we conduct large-scale experiments on 6 250 generated graphs and compare our embedding with the

[1] Our code will be made available at `https://hpi.de/friedrich/research/hyperbolic`.

ground truth data (Section 6). We observe that in typical metrics like Log-likelihood and greedy routing, our algorithm achieves embeddings that are competitive with the original.

Furthermore, we investigate the performance of two classical methods of embedding graphs in the Euclidean space, namely spring embedders and maximum variance unfolding, when applied to the hyperbolic space (Sections 3 and 4). We find that both of them can work under some strong assumptions, but generally fail to translate to large real-world graphs.

2 Preliminaries

In this section, we briefly introduce the hyperbolic random graph model. Due to space constraints, we keep the definitions concise and refer the reader to previous work for a more intuitive introduction, see e.g. [20, 16]. We use the native representation of the hyperbolic space [20] of curvature -1, where points are identified by radial coordinates (r, φ). The first coordinate describes the hyperbolic distance from the origin, and two points x, y have hyperbolic distance

$$\operatorname{dist}(x,y) := \cosh^{-1}(\cosh(r_x)\cosh(r_y) - \sinh(r_x)\sinh(r_y)\cos(\varphi_x - \varphi_y)).$$

The hyperbolic random graph model formally defines a probability distribution over the set of all graphs of size n. A graph G on n vertices is sampled from this distribution as follows. Consider a disc D_n of radius $R = 2\log n + C$ in the hyperbolic space, where C is a parameter adjusting the average degree of the resulting graph. Each vertex v is randomly equipped with hyperbolic coordinates (r_v, φ_v) sampled from the probability density function $f(r,\varphi) = \frac{\alpha \sinh(\alpha r)}{2\pi(\cosh(\alpha R)-1)}$, for a parameter α adjusting the power-law exponent $\beta = 2\alpha + 1$ of the resulting network. Then, every two vertices u, v are connected with probability

$$p(\operatorname{dist}(u,v)) := \left(1 + \exp(\tfrac{1}{2T} \cdot (\operatorname{dist}(u,v) - R))\right)^{-1}, \tag{2.1}$$

where T is a parameter regulating the importance of the underlying geometry: When $T \to 0$, we obtain the so-called *step model*, where an edge $\{u, v\}$ is present if and only if $\operatorname{dist}(u, v) \leqslant R$. For $T > 0$, we obtain the *binomial model*, where long-range edges are possible (but unlikely). Typically, one assumes $0 \leqslant T < 1$. This yields a random graph depending on 4 parameters: n, R (or C), α, and T. Following standard graph notation, we write $\Gamma(v)$ for the set of neighbors of v, and we use δ to refer to the average degree of G.

Further, given a graph $G = (V, E)$ and any mapping from nodes to hyperbolic coordinates $\{r_i, \varphi_i\}_{i=1}^n$, we define the *Log-likelihood* as

$$\mathcal{L}(\{r_i,\varphi_i\}_{i=1}^n \mid G) := \sum_{\{u,v\}\in E} \log(p(\operatorname{dist}(u,v))) + \sum_{\{u,v\}\notin E} \log(1 - p(\operatorname{dist}(u,v))),$$

where the hyperbolic distances dist are taken with respect to the coordinates $\{r_i, \varphi_i\}_{i=1}^n$. To simplify presentation, we write

$$\mathcal{L}(v) := \sum_{u\in\Gamma(v)} \log(p(\operatorname{dist}(u,v))) + \sum_{u\notin\Gamma(v)} \log(1 - p(\operatorname{dist}(u,v))), \tag{2.2}$$

so that we have $\mathcal{L}(\{r_i,\varphi_i\}_{i=1}^n \mid G) = \frac{1}{2}\sum_{v\in V} \mathcal{L}(v)$.

Our goal is to devise an algorithm which, given only the network structure (i.e. a list of edges) of a generated hyperbolic random graph, can re-infer the hyperbolic coordinates of the original embedding. As additional requirements, we would like that the algorithm is robust to noise (i.e. works reasonably well even if the supplied graph was not hyperbolic).

Before presenting our algorithm, we revisit two popular embedding techniques in the Euclidean plane and investigate their performance when applied to the hyperbolic setting.

3 Spring Embedder

A heavily used technique to embed graphs in the Euclidean plane is the force-directed method (also called spring embedder) [17], which works roughly as follows. For every edge one assumes an attractive force pulling its end vertices toward each other, and for every pair of vertices one assumes a repulsive force pushing them away. The algorithm starts with some initial drawing (e.g., by choosing random positions) and computes for each vertex the total force acting on it. Then, all vertices are moved by a small step according to these forces. This is iterated until a stable configuration is reached.

In a drawing generated by a spring embedder, edges are usually short and non-adjacent vertices are usually far away from each other. Moreover, the repulsive forces lead to a somewhat uniform distribution of the vertices in the available space. Note that these are exactly the properties we wish to obtain for our embeddings in the hyperbolic plane. It thus seems natural to adapt spring embedders to the hyperbolic geometry, which actually has been done before by Kobourov and Wampler [18]. In the following we discuss why the straight-forward way of implementing a spring embedder in the hyperbolic plane does not work in our setting. For several adaptations that lead to good results at least for smaller graphs, see the online version.

3.1 Difficulties in the Hyperbolic Plane

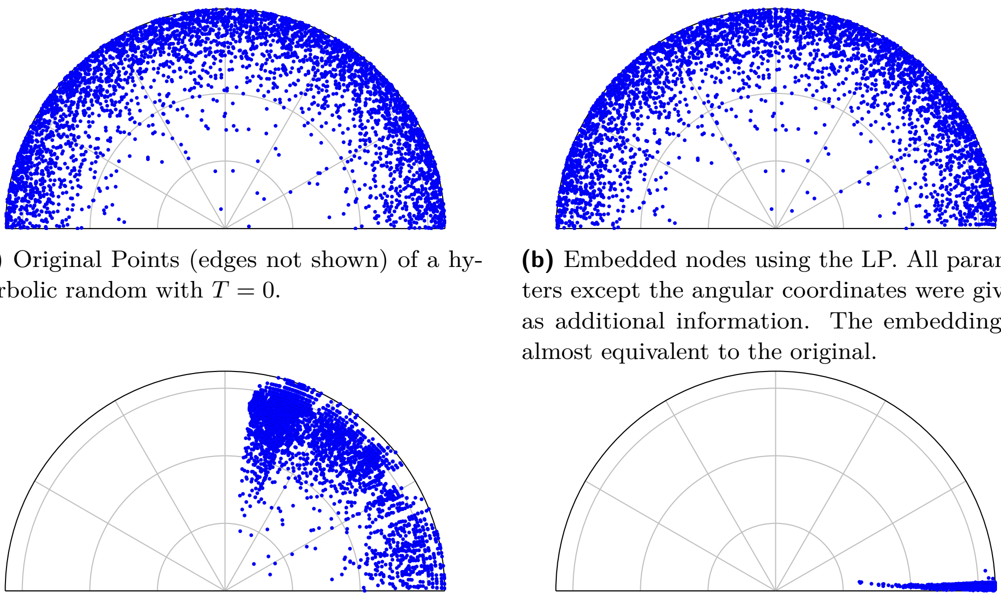
To understand the difficulties in the hyperbolic plane, first consider the following artificial situation in the Euclidean plane. Assume v is a vertex only connected to u; and assume the current drawing is already stable except that v is far away from u. Now when v moves towards u, it also gets closer to other vertices it is not connected to, which then push v back towards the direction where it came from. This is not a problem, however, as there are usually only few vertices close enough to v such that their force is noticeable. Moreover, vertices on the opposite side of v support the movement towards u.

In the hyperbolic plane, an analogous situation works out differently. The geodesic line between v and u contains points with smaller radius, such that v first moves almost directly towards the origin. In turn, the distance to *all* other nodes decreases, which immediately pushes v back to a position with larger radius. Thus, even bad embeddings are stable.

Judging from the pictures presented by Kobourov and Wampler [18], it seems that they did not encounter these issues in their spring embedder. This can be explained by the fact that the radii they use are all rather small, which can be deduced from the presented drawings by observing that the vertices are very well separated from the boundary of the Poincaré disk (which is only true for very small radii). However, for such small radii the hyperbolic plane behaves very similar to the Euclidean plane. We note that using small radii is reasonable for visualizing small graphs using a fish-eye view. However, as the radii in a hyperbolic random graph grow logarithmically with an increasing number of vertices, this is not suitable for our purpose.

4 Maximum Variance Unfolding

Another popular method for embedding graphs into the Euclidean plane is maximum variance unfolding (MVU) [40]. This is essentially a semidefinite program whose objective function spreads out nodes while using constraints to keep neighbors close together. In the one-dimensional case it is equivalent to an LP.


(a) Original Points (edges not shown) of a hyperbolic random with $T = 0$.

(b) Embedded nodes using the LP. All parameters except the angular coordinates were given as additional information. The embedding is almost equivalent to the original.

(c) Embedded nodes using the LP with estimated radial coordinates (See Section 5.1). The quality of the LP solution quickly degrades.

(d) Embedded nodes using the LP with all other parameters given. The graph was generated using $T = 0.5$. The embedding is essentially unusable.

Figure 1 First phase of the LP. Since nodes are placed in $[0, \pi]$, half of D_n is hidden.

The use-case in the hyperbolic geometry is similar: Nodes shall have distance $< R$ if they have an edge, and distance $\geqslant R$ otherwise. It is possible to encode this into the following LP:

$$\begin{array}{lll} \text{maximize} & \sum_{j=1}^{n} \varphi_j & \\ \text{subject to} & \varphi_i - \varphi_j \leqslant \theta(r_i, r_j), & i,j = 1,\dots,n, \quad i \neq j \\ & \varphi_j - \varphi_i \leqslant \theta(r_i, r_j), & i,j = 1,\dots,n, \quad i \neq j \\ & 0 \leqslant \varphi_i \leqslant \pi & i = 1,\dots,n \\ & \varphi_v = 0, & \text{for some starting node } v \end{array}$$

where $\theta(r_i, r_j)$ is the maximal angular distance such that nodes $\operatorname{dist}(i,j) \leqslant R$, i. e.

$$\theta(r_i, r_j) = \arccos\left(\frac{\cosh(r_i)\cosh(r_j) - \cosh(R)}{\sinh(r_i)\sinh(r_j)}\right). \tag{4.1}$$

The LP has a caveat: It is only able to spread nodes on the half circle $[0, \pi]$; since for larger angular coordinates the hyperbolic distances start decreasing again, which is not encodable in the LP. This can be fixed, however, using a small trick: First, embed all nodes on a half-circle with an arbitrary starting node v. Then, pick the node u in the embedding with angular coordinate closest to $\frac{\pi}{2}$; and embed the graph again using u as the starting node.

This yields all nodes that belong in the lower half of D_n: If w has angular distance at least $\frac{\pi}{2}$ from u in the second embedding, we set $\varphi_w = \varphi_w + \pi$ in the first embedding.

This simple method works surprisingly well on generated hyperbolic random graphs that are drawn from the step model, when given all global parameters and radial coordinates, see Figure 1a–b. It is, however, extremely volatile to the quality of the estimated parameters; and it fails completely when used on a real graph or even a graph generated by the binomial

Algorithm 1 Fast Embedding Algorithm

```
Input: Undirected connected Graph $G = (V, E)$
1: Estimate global parameters $n, R, \alpha, T$; and radial coordinates $r_i$    ▷ See Section 5.1
2: Partition nodes into layers such that $v \in L_i \Leftrightarrow \deg(v) \in [2^i, 2^{i+1} - 1]$
3: Embed Core (all nodes in layers $\geqslant \frac{\log n}{2}$)    ▷ See Section 5.2
4: for $i = \frac{\log n}{2} - 1 \ldots 0$ do
5:     for $r = 1 \ldots \log n$ do
6:         for all $v \in \bigcup_{j \geqslant i} L_j$ do
7:             Embed $v$ by optimizing its Log-likelihood    ▷ See Sections 5.3 and 5.4
```

model, see Figure 1c–d. The reason is that the LP has a constraint for each edge in the graph: If there is just one long-range edge, the MVU can no longer unfold the graph and all nodes are mapped to an extremely small range of angular coordinates. This behavior persists even after adding different error terms for edges; and we were not able to make this approach work on noisy data.

5 The Embedder

Our embedding algorithm is inspired by the Metropolis-Hastings Algorithm from [8]. Algorithm 1 contains a bird's eye view over the whole algorithm. Detailed description of individual steps follow in the next sections.

The algorithm proceeds in three phases: First, it estimates all parameters that are computationally easy to guess. This includes the radial coordinates of all nodes, see Section 5.1.

In the second phase, high-degree nodes are embedded by considering their common neighbors. Producing a good initial ordering of nodes in inner layers is crucial for the success of the algorithm since nodes in all subsequent layers are typically placed close to their neighbors in higher layers. This step is described in Section 5.2.

In the third phase, the algorithm embeds the rest of the graph layer-wise. To embed a layer L_i, we iterate over all nodes $v \in L_i$. In each iteration, $\mathcal{O}(\log n)$ angular coordinates for v are sampled; and v is moved to the position with the best Log-likelihood, see Sections 5.3 and 5.4. This is repeated $\log n$ times per layer. While this step is similar to HyperMap [8, 27, 29], we improve upon their algorithm by achieving an amortized polylogarithmic runtime per node as compared to their linear runtime. Our overall algorithm thus runs in $\mathcal{O}(n \cdot \mathrm{polylog}(n))$.

5.1 Parameter Estimation

To bootstrap the embedding algorithm, the global graph parameters have to be known: The original number of nodes n, the radius R of the disc D_n, the parameter α adjusting the power-law exponent; and the parameter T adjusting the clustering. These values are required for instance for evaluating the probability that two nodes are connected, see equation (2.1) which in turn is needed to produce the Log-likelihood. In the following, we give some brief explanations on how each parameter is guessed.

Estimating n. Algorithm 1 expects a connected graph as input, since disconnected components can be placed anywhere in the graph as there is no adjacency information.

The hyperbolic random graph, however, does typically not produce a connected graph. For power-law exponents $2 < \beta < 3$, its giant component is of size $\Theta(n)$ [6, 7]; and for $\beta \geqslant 3$ the graph breaks up into components of order $o(n)$. Unfortunately, the leading constant of the size of the giant component is yet unknown; and a numerical estimation is hard since it is governed by a non-linear system of equations together with other parameters [8].

We have found experimentally that the majority of nodes missing from the giant component are of degree 0. Surprisingly, the most effective and robust method for estimating the number of these nodes was by simply extrapolating from the number of 1- and 2-degree nodes. Let $\hat{n} \cdot F(k)$ be the number of nodes of degree k, where $\hat{n}$ is the total number of nodes in the input graph. Then, we estimate n simply by setting $n := \hat{n}(1 + \max\{0, 2F(1) - F(2)\})$.

Estimating α. The parameter α adjusts the power-law exponent β of the hyperbolic random graph via the functional behavior $\beta = 2\alpha + 1$ [16]. We estimate β from the cumulative degree distribution using the classical algorithm by Clauset et al. [11].

Estimating T. Recall that this parameter adjusts the importance of the underlying geometric structure. It has recently been observed, however, that T does not have a big influence on the quality of the embedding [27]. We found that setting T to a small fixed value like 0.1 produces good results. We investigate the role of T closer in the online version.

Estimating R and r_i. We estimate these values using the above determined parameters. Good analytical estimates have been derived in previous work [8]:

$$R = 2\log\left(\frac{4n^2\alpha^2 T}{|E| \cdot \sin(\pi T)(2\alpha - 1)^2}\right), \quad r_i = \min\left\{R,\ \ 2\log\left(\frac{2n\alpha T}{\deg(i) \cdot \sin(\pi T)(\alpha - \frac{1}{2})}\right)\right\}$$

5.2 Embedding the Core

Laying out the large-degree nodes (also called the *core* of the graph) has a huge impact on the overall performance of the embedding. We consider all nodes v with radial coordinates $r_v < R/2$ to be in the core, of which there are $\Theta(n^{1-\alpha})$ in expectation [14]. If the node ordering of the core is roughly correct, the algorithm will usually yield excellent embeddings. One the other hand, if the core was embedded poorly, the remaining steps can not salvage the poor initialization. Thus, we put considerable care into embedding the core correctly.

HyperMap [29] uses the number of common neighbors of large degree nodes to lay out the core: For two nodes u, v they compute the number $c_{uv} = |\Gamma(u) \cap \Gamma(v)|$, and numerically determined the angle $\varphi(c_{uv}, r_u, r_v)$ that maximizes the likelihood that the nodes u, v have c_{uv} common neighbors. This is a promising approach, as the common neighborhood of large nodes is tightly concentrated around its expected value. Determining the likelihood numerically, however, is a computationally expensive operation.

To overcome this, we analytically derive in Section 5.2.1 an approximate expression for the relative angle of two nodes up to constant factors. Using this, we present a spring embedder in Section 5.2.2 that embeds the core based on the estimated pair-wise angle differences.

5.2.1 Estimating the Angle-Differences

To estimate the relative angle between two nodes, we use their inferred radial coordinates and the number of their common neighbors. We perform this computation in the step model; however, we have experimentally found that our results hold up well in the binomial model.

Let u, v be the two nodes whose (expected number of) common neighbors we wish to compute. They have radii r_u and r_v, respectively, and a relative angle of $\Delta\theta(u, v)$. W. l. o. g., we assume that $r_u \leqslant r_v$. Consider now a third node w. We compute the probability that w

is connected to both u and v. Under the assumption that $r_u + r_w \geqslant R$ and $r_v + r_w \geqslant R$, we know from [16] that this only holds if

$$\begin{aligned}&\Delta\theta(u,w) \leqslant 2e^{\frac{1}{2}(R-r_u-r_w)}(1+\Theta(e^{R-r_u-r_w})), \quad \text{and}\\ &\Delta\theta(v,w) \leqslant 2e^{\frac{1}{2}(R-r_v-r_w)}(1+\Theta(e^{R-r_v-r_w})). \end{aligned} \tag{5.1}$$

Assume $r_v + r_w \geqslant R$ does not hold. In this case, the distance between v and w is obviously at most R and thus they are connected. Moreover, note that in this case the right hand side of the above formula increases with increasing R and thus the inequality is satisfied for any angle $\Delta\theta(v,w)$ if R is sufficiently large. Thus, under the assumption that R is sufficiently large, we may use equation (5.1).

Observe now that for large enough radii r_w, the node w is not connected to either u or v (unless $\Delta\theta(u,v) \leqslant \mathcal{O}(\frac{1}{n})$). On the other hand, when $R - r_v - r_w = \Omega(1)$, w is connected with constant probability to both u and v. Thus, depending on the radius r_w, there is a "good" fraction of the angular coordinates $[0,2\pi)$ where w will be connected to both nodes, and a "bad" fraction where it will be connected to only one or neither of u, v. We call the probability to be connected to both nodes $p_g(r_w)$.

We already know that $p_g(r_w) = 1 \Leftrightarrow r_w = R - r_v \pm \Theta(1)$. We label this critical value of r_w with r_1. On the other hand, $p_g(r_w) = 0$ holds when $\theta(r_u, r_w) + \theta(r_v, r_w) \leqslant \Delta\theta(u,v)$, since then there is no possible angle for φ_w where it is connected to both nodes u, v, see equation (4.1). The critical value r_0 for which this number becomes positive is when $\theta(r_u, r_w) + \theta(r_v, r_w) = \Delta\theta(u,v)$ and thereby

$$\begin{aligned}\Delta\theta(u,v) &= 2e^{\frac{1}{2}(R-r_u-r_0)}(1 \pm \Theta(e^{R-r_u-r_0})) + 2e^{\frac{1}{2}(R-r_v-r_0)}(1 \pm \Theta(e^{R-r_v-r_0}))\\ &= \Theta(1) \cdot e^{\frac{1}{2}(R-r_u-r_0)}.\end{aligned}$$

Solving for r_0, this holds whenever $r_0 = \min\{R, R - r_u - 2\log(\Delta\theta(u,v)) \pm \Theta(1)\}$.

For values $r_1 \leqslant r_w \leqslant r_0$, the regions in which w connects to u, v both increase as in equation (5.1). Thus, the intersection of these regions increases as $p_g(r_w) \sim e^{-r_w/2}$. To determine the function up to constants, we set

$$1 = p_g(r_1) = A \cdot e^{-r_1/2} + B, \quad \text{and} \quad 0 = p_g(r_0) = A \cdot e^{-r_0/2} + B.$$

Solving this system of equations, we obtain that $p_g(r_w) = \Theta(1) \cdot (e^{\frac{1}{2}(r_1-r_w)} - e^{\frac{1}{2}(r_1-r_0)})$. Thus, we may compute the probability that an arbitrary node is connected to both u and v using the cumulative distribution function and p_g. We thereby have

$$\begin{aligned}\Pr[w \sim u,v] &= \int_0^R \rho(r) \cdot p_g(r)\,\mathrm{d}r\\ &= \Pr[r_w \leqslant r_1] + \Theta(1) \cdot \int_{r_1}^{r_0} e^{\alpha r - \alpha R} \cdot (e^{\frac{1}{2}(r_1-r)} - e^{\frac{1}{2}(r_1-r_0)})\,\mathrm{d}r\\ &= e^{\alpha r_1 - \alpha R} + \Theta(1) \cdot \left[e^{\alpha r - \alpha R} \cdot (\tfrac{1}{\alpha - \frac{1}{2}} e^{\frac{1}{2}(r_1-r)} - \tfrac{1}{\alpha} e^{\frac{1}{2}(r_1-r_0)})\right]_{r_1}^{r_0}\\ &= \Theta(1) \cdot e^{\alpha r_0 - \alpha R + \frac{1}{2}(r_1-r_0)}.\end{aligned}$$

Hence, the expected number of common neighbors of u and v is

$$c_{uv} = \Theta(1) \cdot \exp(\tfrac{R}{2} + (\tfrac{1}{2} - \alpha)r_u - \tfrac{1}{2}r_v) \cdot \Delta\theta(u,v)^{1-2\alpha}.$$

To find the angle $\varphi(c_{uv}, r_u, r_v)$ maximizing the Log-likelihood in the step model, we observe that the number of common neighbors of u, v is a binomial random variable: There exists a set $S \subseteq D_n$ in which each node is connected to both u, v and each node in $D_n \setminus S$ connected to at most one of u, v. Since the maximum likelihood estimator for binomial random variables is the number of successes divided by the number of trials, we obtain the maximum likelihood for $\Delta\theta(u, v)$ by rearranging above equation.

$$\varphi(c_{uv}, r_u, r_v) = \Theta(1) \cdot c_{uv}^{\frac{1}{1-2\alpha}} \cdot \exp(-\tfrac{1}{2}r_u + (\tfrac{1}{2-4\alpha})(r_v - R)).$$

To obtain actual values for $\Delta\theta(u, v)$ we first simply omit the constant factor hidden by $\Theta(1)$ in the above expression. To obtain reasonable angles, observe that the largest angle should likely be π. To obtain this, one can simply rescale all values of $\varphi(c_{uv}, r_u, r_v)$ with the same constant factor such that the maximum is π. As this is prone to errors if outliers exist, we instead scale all angles by the same constant such that their median is $\pi/2$. Angles that are larger than π after this scaling are then set to π. Preliminary experiments showed that using the logarithm of the above expression for initially computing $\Delta\theta(u, v)$ (before the scaling) improved the robustness of our algorithm.

5.2.2 Embedding According to the Estimated Angles

In this section, we assume that we know the desired angle $\Delta\theta(u, v)$ between any pair of vertices u and v in the core. Our goal is to assign an angle to each vertex that realizes these differences as good as possible. To this end, we use a 1-dimensional spring embedder (see Section 3 for a short introduction to spring embedders) that basically works as follows. We start with random initial angles. Then in each iteration, we consider every pair u, v of vertices. If the the current angle between u and v is larger than $\Delta\theta(u, v)$ we get an attractive force, otherwise we get a repulsive force. W. l. o. g., we assume $0 \leqslant \varphi_u < \varphi_v \leqslant \pi$ (the other cases work symmetrically). Moreover, let $\mathrm{err}(u, v) = \varphi_v - \varphi_u - \varphi(c_{uv}, r_u, r_v)$. The force $F_u(v)$ acting on u due to v is then given by

$$F_u(v) = \begin{cases} -\operatorname{err}(u,v)^2 & \text{if } \operatorname{err}(u,v) \leqslant 0, \\ \operatorname{err}(u,v)^2 & \text{if } 0 < \operatorname{err}(u,v) \leqslant \frac{\pi}{2}, and \\ (\pi - \operatorname{err}(u,v))^2 & \text{if } \frac{\pi}{2} < \operatorname{err}(u,v) \leqslant \pi. \end{cases}$$

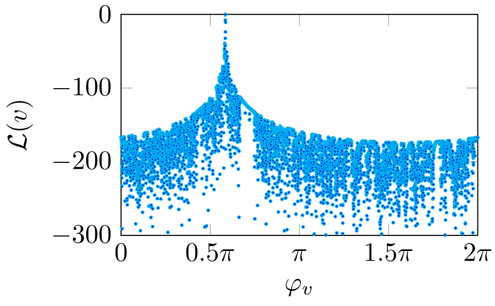
To interpret this formula, first note that $\mathrm{err}(u, v) < 0$ holds if the current angle is too small. Thus, $F_u(v)$ is negative (pushing u away from v) and the strength of the force increases quadratically in the distance to the desired angle. Conversely, if the current angle is too large, we get a repulsive force increasing quadratically in the distance to the desired angle as long as this distance is at most $\pi/2$. For larger distances, the strength of the force actually decreases again. This has the following reason. Imagine the extreme case that u and v have angle π between them but actually want to have a very small angle. Then it does not really matter whether the angle of u increases or decreases as it comes closer to v not matter what. Thus, we do not really want a very strong force in one of the two directions, which is the reason why we decrease the strength of attractive forces when $\mathrm{err}(u, v)$ becomes very large.

Similar to Section 3, the total force acting on u is defined as

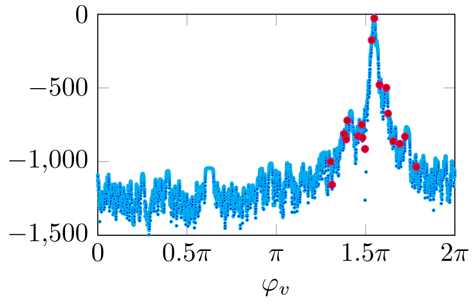
$$F_u = \sum_{v \in V \setminus u} F_u(v)$$

and the new angle of u is obtained by setting $\varphi_u = \varphi_u + cF_u$. The value for c is again chosen such that the maximum step size does not exceed a parameter $\theta_{\max} := \max_{u \in V}\{cF_u\}$.

Due to the 1-dimensionality of this spring embedder, we encounter a similar problem as for the hyperbolic spring embedder in Section 3: to move a vertex u to a specific position,



(a) Exemplary fitness landscape for a node v with 3 neighbors. Both methods for computing the fitness landscape exhibit no visible difference in the plot.



(b) Fitness landscape of a node v and the coordinates at which the efficient algorithm samples the fitness. Red points indicate the sampled angles.

Figure 2 Fitness landscape of a node v computed with the efficient algorithm.

it necessarily has to pass through all vertices in between and there is no second dimension that could be used to get around them. This leads to strong repulsive forces hindering u in getting to the desired position and we observed in our experiments that the algorithm often gets stuck in a local minimum. As before, we use velocity and a rather large step size $\theta_{\max}$ to circumvent this issue. Preliminary experiments showed that we obtain good results using the following parameters. We set $\theta_{\max} = 0.55\pi$ in the first iteration, decreasing it linearly down to 0 in the final iteration. For the velocity assume F_u is the force from iteration i. Then we add cF_u to the force in iteration $i+1$ where c is 1 in the first iteration and linearly decreases down to 0.5 in the last iteration. Since there are $\Theta(n^{1-\alpha})$ nodes in the core [14], the total runtime of the spring embedder is $\mathcal{O}(k \cdot n^{2-2\alpha})$, where k is the number of iterations. Choosing $k = \mathcal{O}(n^{2\alpha-1})$, we achieve a runtime of $\mathcal{O}(n)$.

The performance of this algorithm depends on the randomly chosen initial angles. To be able to compare core embeddings, we define a score S as

$$S = \sum_{u \in V} \sum_{v \in V \setminus u} |F_u(v)| \, .$$

A smaller score then indicates a better embedding. We define s_{opt} as the score that is obtained when the spring embedder is initialized with the original coordinates. We then say that a core embedding is *good*, if it has a score $s \leqslant 1.2 \cdot s_{\text{opt}}$. Each graph thus has a certain probability that the core embedding is good, depending on the randomly chosen initial positions. To further increase the probability of getting a good embedding for the core, we run the spring embedder 5 times with different initial angles and use the best result, which boosts the probability of getting a good embedding to 95% for the *worst* of over 3 000 randomly generated hyperbolic random graphs (see Section 6 for the experimental setup). This suggests that the spring embedder is rather robust, i.e. we rarely encounter initial drawings that lead to bad results.

5.3 Computing the Log-likelihood efficiently

A key ingredient to achieve a quasilinear runtime is to improve the runtime of the Log-likelihood computation $\mathcal{L}(v)$. By a naive implementation of the Log-likelihood $\mathcal{L}(v)$ (see equation (2.2)), one needs $\Omega(n)$ time to compute the Log-likelihood of a single node. A more careful inspection, however, allows for a significant speedup.

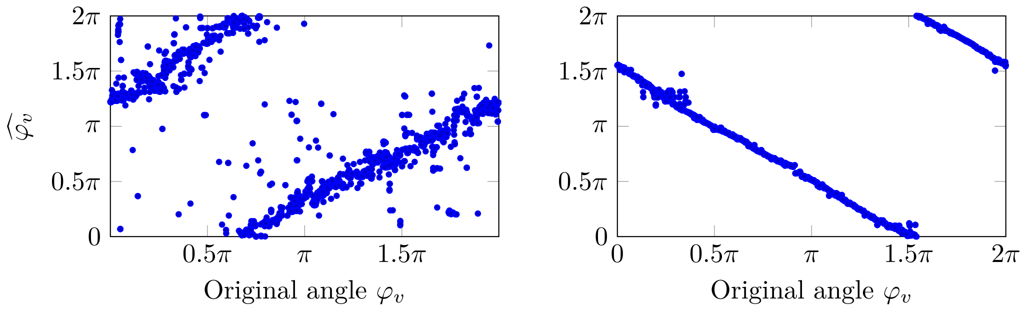


Figure 3 The plots correspond to embeddings with average squared deviation $\Delta\varphi_G = 0.44$ (left) and $\Delta\varphi_G = 0.01$ (right). For each vertex v the plot contains one point with x-coordinate φ_v (angle of v in the original embedding) and y-coordinate $\widehat{\varphi_v}$ (angle in the computed embedding).

First, observe that the total number of edges in a hyperbolic random graph is of order $\mathcal{O}(n)$; so the term $\sum_{u\in\Gamma(v)} \log(p_{uv})$ can be computed in amortized constant time. To speed up the computation of the second summand, we observe that the term $\log(1-p_{uv})$ is very close to 0 whenever $\mathrm{dist}(u,v) \gg R$, since

$$p_{uv} := (1+\exp(\tfrac{1}{2T}(\mathrm{dist}(u,v)-R)))^{-1} \approx \exp(-\tfrac{1}{2T}(\mathrm{dist}(u,v)-R)),$$

and by a Taylor series for $p_{uv} \to 0$ we get

$$\log(1-p_{uv}) = -p_{uv} - \mathcal{O}(p_{uv}^2) \approx -\exp(-\tfrac{1}{2T}(\mathrm{dist}(u,v)-R)).$$

This implies that non-neighbors that are far away from v barely contribute to its Log-likelihood. If, on the other hand, $\mathrm{dist}(u,v) \ll R$, we have $p_{uv} \to 1$, and thus

$$\log(1-p_{uv}) \approx \log(1-(1-\exp(\tfrac{1}{2T}(\mathrm{dist}(u,v)-R)))) = \tfrac{1}{2T}(\mathrm{dist}(u,v)-R).$$

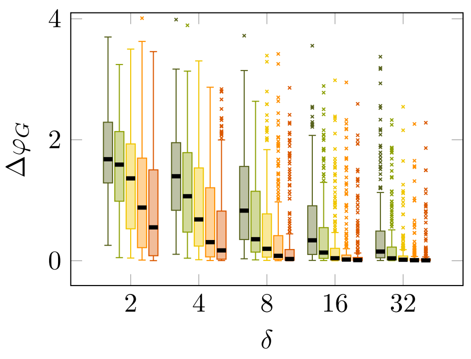
Thus, it suffices to take into account non-neighbors with low distance from u while either ignoring or coarsely approximating the influence of far away non-neighbors on the Log-likelihood. To this end, we implemented the geometric data structures introduced by Bringmann et al. [9]. These were originally used to generate hyperbolic random graphs in linear time by partitioning the disc D_n into suitably sized cells. To compute the Log-likelihood of a node, one can then compare it directly with nodes in neighboring cells (that have a big influence on the Log-likelihood); while averaging over all nodes in far away cells. As shown in [9], this runs in amortized time $\mathcal{O}(1)$. We need an extra $\mathcal{O}(\log n)$ factor to update the cells whenever a node is moved during the embedding algorithm.

Figure 2a shows the fitness landscapes of a node v; computed once via the classical exact $\Omega(n)$ method, and once using our amortized $\mathcal{O}(1)$ method. Both methods exhibit no visible differences in the plot; and we found that the relative error made by the fast Log-likelihood computation is $\leqslant 1.0025$ at all coordinates except one, where it was $\leqslant 1.02$.

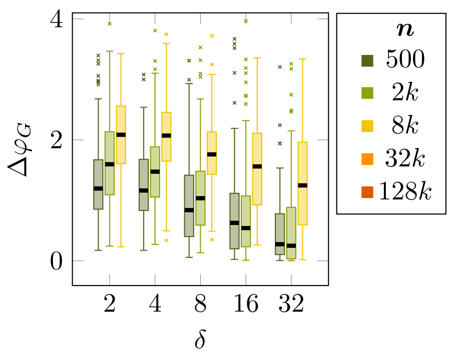
5.4 Finding the Optimal Angle

To find a good angular coordinate for a node v, previous algorithms typically scan the whole range $[0, 2\pi)$ at resolution $\frac{2\pi}{n}$; and evaluate at each angle the Log-likelihood $\mathcal{L}(v)$. This incurs another factor $\Omega(n)$ on the overall runtime.

To save on this, we sample only few points around a region where a node has their maximum likelihood. To this end, we observe that the coarse likelihood landscape for a node



(a) Our main algorithm.



(b) Our hyperbolic spring embedder.

Figure 4 Each data point in the box plot represents the value of $\Delta\varphi_G$ for a single graph G (y-axis) depending on the average degree a (x-axis). The graphs are grouped into small, medium, and large graphs.

v (for small T) is governed by the position of v's neighbors. Furthermore, neighbors with large radii have a larger influence on the fitness landscape, as the hyperbolic distance to these nodes increases more quickly than to neighbors with small radial coordinates. Hence, v needs to be placed close to its embedded low-degree neighbors.

Ignoring non-neighbors for now, we achieve this by computing a weighted average over the angles of all neighbors of v. Let $u_1, \dots, u_k$ be the embedded neighbors of v. Then, v's angle is computed as follows.

$$\varphi_v = \arctan\left(\frac{\sum_{i=1}^{k} \exp(r_{u_i}) \cdot \sin(\varphi_{u_i})}{\sum_{i=1}^{k} \exp(r_{u_i}) \cdot \cos(\varphi_{u_i})}\right)$$

To take non-neighbors into consideration, we then randomly sample $\mathcal{O}(\log(n))$ points around this angle and use the one with the smallest Log-likelihood. Figure 2b shows the fitness landscape of an exemplary node u, as well as the randomly sampled angles. As can be seen, the heuristic typically finds good candidates whose angles are close to the optimal angle.

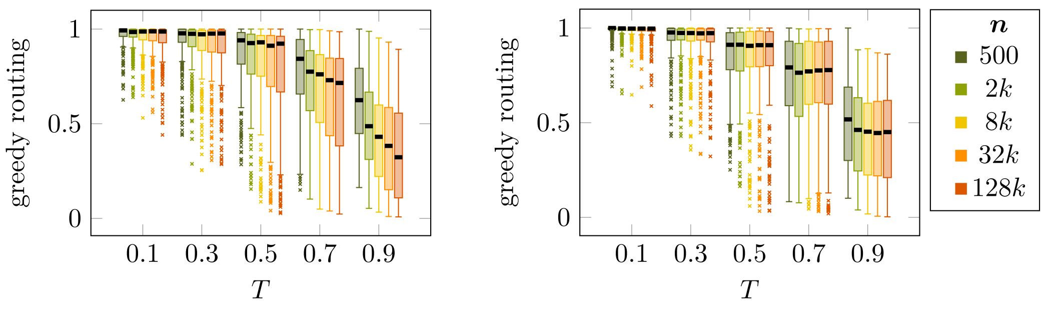
6 Experiments

To evaluate the quality of our algorithm, we sampled 10 different graphs for every combination of the following parameters: $\alpha \in \{0.55, 0.65, 0.75, 0.85, 0.95\}$, $T \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$, $\delta \in \{2, 4, 8, 16, 32\}$, $n \in \{500, 2\,000, 8\,000, 32\,000, 128\,000\}$. This results in a total of 6 250 graphs. For each of these graphs, we computed the following statistics: Log-likelihood, success ratio of greedy routing and the average squared deviation in the original angle vs. estimated angle plot. We present the most insightful statistics in standard box plot form.[2]

6.1 Quality

A popular way to judge whether an embedding makes sense is to plot the embedded angular coordinates against the original generated coordinates. If the result resembles a straight line

[2] A box contains 50% of all data points; the median is marked black. Points are considered outliers if they have distance more than 1.5× IQR to the box. The whiskers depict the closest data point to the box that is not an outlier.



(a) Our main algorithm. **(b)** Originally generated embeddings.

Figure 5 The success ratio of greedy routing (x-axis) depending on the value of T (y-axis) grouped with respect to the number of vertices (colors).

(that might have a cyclic shift), then the relative ordering of nodes has been reconstructed well in the embedding. Two examples for such plots are shown in Figure 3. To allow for comparisons that scale to a large amount of graphs, we derive the following quality measure. For a vertex v let $\Delta\varphi_v$ be the quadratic difference between φ_v in the original embedding and φ_v in the computed embedding. For a graph $G = (V, E)$, the value $\Delta\varphi_G = \sum_{v \in V} \Delta\varphi_v / n$ then describes the average squared deviation in G.

The box plot in Figure 4a plots $\Delta\varphi_G$ against the average degree δ; grouped by the size of the graph. In this and all other plots, we average over all parameters that are not explicitly grouped by. Observe that $\Delta\varphi_G$ is high if the average degree is small, as the few existing edges are not sufficient to uniquely determine the single best embedding. Thus, several embeddings may be equally good. In fact, for small δ, our algorithm finds an embedding with a Log-likelihood very close to the Log-likelihood of the original embedding (the mean values for large graphs with $\delta = 2$ are $-2.39 \cdot 10^5$ for the embedding and $-2.19 \cdot 10^5$ for the original, respectively, while the corresponding values for $\delta = 16$ are $-1.78 \cdot 10^6$ and $-1.16 \cdot 10^6$). For an average degree of 8, the mean value for $\Delta\varphi_G$ of all medium sized and large graphs is 0.2 and 0.04, respectively. For comparison, note that the plots in Figure 3 correspond to graphs with values 0.01 and 0.44. Also note that our algorithm performs particularly well on large graphs, which was the goal we aimed for.

For comparison with the spring embedder described in the online version, see Figure 4b. As the spring embedder is too slow on large graphs, we only ran the experiments on medium and small graphs. Note that the quality of the spring embedder decreases for increasing graph size. In contrast, it performs comparatively well on small graphs (and in some cases actually better than our main algorithm) while it is heavily outperformed on the medium sized graphs. Hence, the spring embedder is a reasonable option for graphs with up to 1 000 vertices, while our main algorithm is the better option for larger graphs.

A quality measure previously used for hyperbolic embeddings is the success ratio of greedy routing. Figure 5a shows this ratio for the embeddings generated by our algorithm depending on the parameter T, grouped by the size of the graph. Observe that the ratio is close to 100% for small values of T but drops significantly for larger values. This is unfortunate as real world graphs are considered to have fairly large values of T, e.g., $T = 0.7$ was used for the embedding of the Internet graph [8]. Though this particular embedding allows greedy routing with success ratio 97%, the ratios of around 80% we obtain for $T = 0.7$ seem to reflect the typical behavior of random hyperbolic graphs much better; see Figure 5b.

Note that these observations imply that maximizing the Log-likelihood will not necessarily lead to the desired result in terms of greedy routing. Conversely, optimizing the embedding

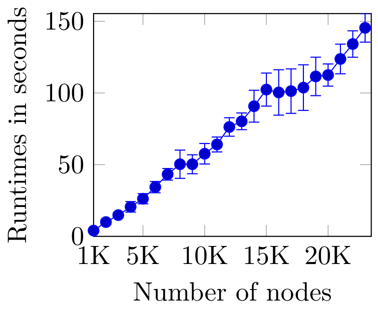


Figure 6 Runtimes for the embedding algorithm. Error bars show the standard deviation.

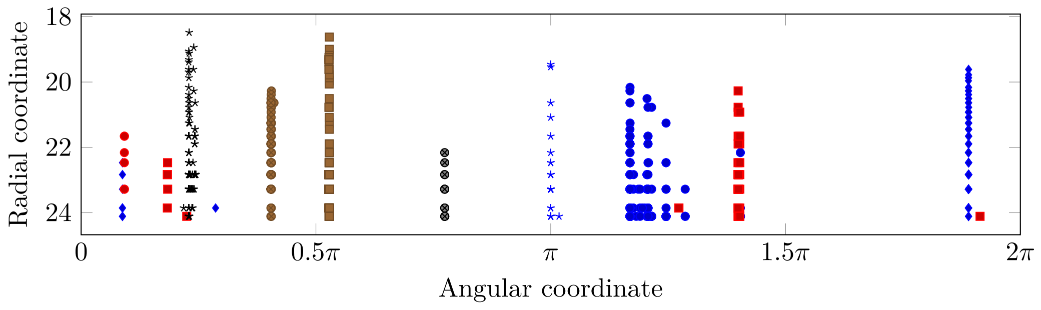


Figure 7 The nine largest communities in the amazon product recommendation network. For clarity, only nodes that belong to a single community are shown. Nodes belonging to the same community are typically placed nearby, even though the embedding algorithm had no knowledge of the ground truth communities.

for greedy routing will probably not lead to an embedding that is close to the original embedding of a hyperbolic random graph. Hence, we do not see the low success ratios our embeddings achieve for large T as a weakness but rather as a strength as it matches the behavior of the original embedding.

6.2 Runtime

A key contribution of our algorithm is its significant improvement on the runtimes compared to previous approaches. The runtime experiments were performed on commodity hardware, i.e. a 2.7 GHz Core i7 with 8 GB of RAM. Figure 6 shows the runtimes depending on n. Note that compared to available algorithms these are fairly quick: Graphs of size 20 000 can be embedded in under two minutes. We even embedded graphs of size 330 000 in under one hour, see Section 6.3. For comparison, the reference algorithm HyperMap [27, 29] needs over 1.5 hours for a graph of size 2 000.

6.3 Embedding a Real-World Graph

As a proof of concept, we embed the Amazon product recommendation network [41]. It has $n = 334\,863$ nodes with an average degree of 5.53, the degree distribution follows a power-law with exponent $\beta = 3.6$ and the average clustering coefficient is 0.4. The nodes represent products available on Amazon, and an edge $\{u, v\}$ is present if product u is recommended together with product v. Product categories define ground truth communities in this graph.

The embedding took 50 minutes on a single 2.7 GHz Core i7. While the number of nodes is too large to visually inspect the whole graph, we have plotted the nine largest communities in Figure 7. Most nodes belonging to a single community are mapped close together; which suggests that the hyperbolic embedding might be a useful tool in discovering hidden communities in a large network.

Acknowledgements. We thank the authors of [27] for their code and for helpful discussions; Christoph Kessler (HPI Potsdam) and Maximilian Katzmann (FSU Jena) for their help with the experiments; and Konrad Schöbel (FSU Jena) for fruitful discussions on hyperbolic variants of MDS.

References

1 William Aiello, Fan Chung, and Linyuan Lu. A random graph model for massive graphs. In *32nd Symp. Theory of Computing (STOC)*, pages 171–180, 2000.

2 William Aiello, Fan Chung, and Linyuan Lu. A random graph model for power law graphs. *Experimental Mathematics*, 10(1):53–66, 2001.

3 Rodrigo Aldecoa, Chiara Orsini, and Dmitri Krioukov. Hyperbolic graph generator. *Computer Physics Communications*, 196:492–496, 2015. `doi:10.1016/j.cpc.2015.05.028`.

4 Dena Marie Asta and Cosma Rohilla Shalizi. Geometric network comparisons. In *31st Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 102–110, 2015.

5 Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.

6 Michel Bode, Nikolaos Fountoulakis, and Tobias Müller. On the giant component of random hyperbolic graphs. In *7th European Conf. Combinatorics, Graph Theory and Applications*, pages 425–429, 2013.

7 Michel Bode, Nikolaos Fountoulakis, and Tobias Müller. The probability that the hyperbolic random graph is connected. `www.math.uu.nl/~Muell001/Papers/BFM.pdf`, 2014.

8 Marián Boguñá, Fragkiskos Papadopoulos, and Dmitri Krioukov. Sustaining the internet with hyperbolic mapping. *Nature Communications*, 1:62, 2010.

9 Karl Bringmann, Ralph Keusch, and Johannes Lengler. Geometric inhomogeneous random graphs. *arXiv preprint arXiv:1511.00576*, 2015.

10 F. Chung and L. Lu. Connected components in random graphs with given expected degree sequences. *Annals of Combinatorics*, 6(2):125–145, 2002.

11 Aaron Clauset, Cosma Rohilla Shalizi, and M. E. J. Newman. Power-law distributions in empirical data. *SIAM Review*, 51(4):661–703, 2009.

12 James R. Clough and Tim S. Evans. Embedding graphs in lorentzian spacetime. *arXiv 1602.03103*, 2016.

13 Trevor F. Cox and M.A.A. Cox. *Multidimensional Scaling, Second Edition.* Chapman and Hall/CRC, 2 edition, 2000.

14 Tobias Friedrich and Anton Krohmer. Cliques in hyperbolic random graphs. In *34th IEEE Conf. Computer Communications (INFOCOM)*, pages 1544–1552, 2015.

15 Tobias Friedrich and Anton Krohmer. On the diameter of hyperbolic random graphs. In *42nd Intl. Coll. Automata, Languages and Programming (ICALP)*, pages 614–625, 2015.

16 Luca Gugelmann, Konstantinos Panagiotou, and Ueli Peter. Random hyperbolic graphs: degree sequence and clustering. In *39th Intl. Coll. Automata, Languages and Programming (ICALP)*, pages 573–585, 2012.

17 Stephen G. Kobourov. *Handbook of Graph Drawing and Visualization*, chapter Force-Directed Drawing Algorithms, pages 383–408. Chapman and Hall/CRC, 2013.

18 Stephen G. Kobourov and Kevin Wampler. Non-eeuclidean spring embedders. *IEEE Transactions on Visualization and Computer Graphics*, 11(6):757–767, 2005.

19 Christoph Koch and Johannes Lengler. Bootstrap percolation on geometric inhomogeneous random graphs. In *43rd Intl. Coll. Automata, Languages and Programming (ICALP)*, 2016.

20 Dmitri Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguñá. Hyperbolic geometry of complex networks. *Phys. Rev. E*, 82:036106, 2010.

21 John Lamping, Ramana Rao, and Peter Pirolli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *13th ACM CHI*, pages 401–408, 1995. `doi:10.1145/223904.223956`.

22 Jonh Lamping and Ramana Rao. The hyperbolic browser: A focus+context technique for visualizing large hierarchies. *Journal of Visual Languages & Computing*, 7(1):33–55, 1996.

23 David Liben-Nowell and Jon M. Kleinberg. The link-prediction problem for social networks. *J. Am. Soc. Inf. Sci.*, 58(7):1019–1031, 2007. `doi:10.1002/asi.20591`.

24 Tamara Munzner. Exploring large graphs in 3d hyperbolic space. *IEEE Computer Graphics and Applications*, 18(4):18–23, 1998. `doi:10.1109/38.689657`.

25 M. E. J. Newman. Clustering and preferential attachment in growing networks. *Phys. Rev. E*, 64:025102, 2001.

26 Ilkka Norros and Hannu Reittu. On a conditionally Poissonian graph process. *Advances in Applied Probability*, 38(1):59–75, 2006.

27 Fragkiskos Papadopoulos, Rodrigo Aldecoa, and Dmitri Krioukov. Network geometry inference using common neighbors. *Phys. Rev. E*, 92:022807, Aug 2015. `doi:10.1103/PhysRevE.92.022807`.

28 Fragkiskos Papadopoulos, Maksim Kitsak, M Ángeles Serrano, Marián Boguñá, and Dmitri Krioukov. Popularity versus similarity in growing networks. *Nature*, 489(7417):537–540, 2012.

29 Fragkiskos Papadopoulos, Constantinos Psomas, and Dmitri V. Krioukov. Network mapping by replaying hyperbolic growth. *IEEE/ACM Trans. Netw.*, 23(1):198–211, 2015. `doi:10.1109/TNET.2013.2294052`.

30 Mathew D. Penrose. *Random Geometric Graphs.* Oxford University Press, 2003.

31 Yuval Shavitt and Tomer Tankel. Hyperbolic embedding of internet graph for distance estimation and overlay construction. *IEEE/ACM Trans. Netw.*, 16(1):25–36, 2008. `doi:10.1145/1373452.1373455`.

32 Eleni Stai, Vasileios Karyotis, and Symeon Papavassiliou. A hyperbolic space analytics framework for big network data and their applications. *IEEE Network*, 30(1):11–17, 2016. `doi:10.1109/MNET.2016.7389825`.

33 Remco van der Hofstad. Random graphs and complex networks. Available at `www.win.tue.nl/~rhofstad/NotesRGCN.pdf`, 2011.

34 Alexei Vázquez. Growing network with local rules: Preferential attachment, clustering hierarchy, and degree correlations. *Phys. Rev. E*, 67:056104, May 2003. `doi:10.1103/PhysRevE.67.056104`.

35 Kevin Verbeek and Subhash Suri. Metric embedding, hyperbolic space, and social networks. In *30th Annual Symposium on Computational Geometry (SOCG)*, page 501, 2014. `doi:10.1145/2582112.2582139`.

36 Moritz von Looz, Henning Meyerhenke, and Roman Prutkin. Generating random hyperbolic graphs in subquadratic time. In *26th Intl. Symp. Algorithms and Computation (ISAAC)*, pages 467–478. Springer, 2015.

37 Jörg A. Walter. H-MDS: a new approach for interactive visualization with multidimensional scaling in the hyperbolic space. *Inf. Syst.*, 29(4):273–292, 2004. `doi:10.1016/j.is.2003.10.002`.

38 Jörg A. Walter and Helge J. Ritter. On interactive visualization of high-dimensional data using the hyperbolic plane. In *8th ACM Intl. Conf. Knowledge Discovery and Data Mining (SIGKDD)*, pages 123–132, 2002. `doi:10.1145/775047.775065`.

39 Zuxi Wang, Qingguang Li, Fengdong Jin, Wei Xiong, and Yao Wu. Hyperbolic mapping of complex networks based on community information. *Physica A: Statistical Mechanics and its Applications*, 455:104–119, 2016.

40 Kilian Q Weinberger and Lawrence K Saul. Unsupervised learning of image manifolds by semidefinite programming. *International Journal of Computer Vision*, 70(1):77–90, 2006.

41 Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 42(1):181–213, 2015.

42 Xiaohan Zhao, Alessandra Sala, Haitao Zheng, and Ben Y. Zhao. Efficient shortest paths on massive social graphs. In *7th International Conference on Collaborative Computing (CollaborateCom)*, pages 77–86, 2011.

Fully Dynamic Spanners with Worst-Case Update Time*

Greg Bodwin[1] and Sebastian Krinninger[2]

1 **Stanford University, Stanford, USA**
gbodwin@cs.stanford.edu
2 **Max Planck Institute for Informatics, Saarbrücken, Germany**
skrinnin@mpi-inf.mpg.de

Abstract

An *α-spanner* of a graph G is a subgraph H such that H preserves all distances of G within a factor of α. In this paper, we give fully dynamic algorithms for maintaining a spanner H of a graph G undergoing edge insertions and deletions with worst-case guarantees on the running time after each update. In particular, our algorithms maintain:

- a 3-spanner with $\tilde{O}(n^{1+1/2})$ edges with worst-case update time $\tilde{O}(n^{3/4})$, or
- a 5-spanner with $\tilde{O}(n^{1+1/3})$ edges with worst-case update time $\tilde{O}(n^{5/9})$.

These size/stretch tradeoffs are best possible (up to logarithmic factors). They can be extended to the weighted setting at very minor cost. Our algorithms are randomized and correct with high probability against an oblivious adversary. We also further extend our techniques to construct a 5-spanner with suboptimal size/stretch tradeoff, but improved worst-case update time.

To the best of our knowledge, these are the *first* dynamic spanner algorithms with sublinear worst-case update time guarantees. Since it is known how to maintain a spanner using small *amortized* but large *worst-case* update time [Baswana et al. SODA'08], obtaining algorithms with strong worst-case bounds, as presented in this paper, seems to be the next natural step for this problem.

1998 ACM Subject Classification F.2.2 Computations on discrete structures, G.2.2 Graph algorithms

Keywords and phrases Dynamic graph algorithms, spanners

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.17

1 Introduction

An *α-spanner* of a graph G is a sparse subgraph that preserves all original distances within a multiplicative factor of α. Spanners are an extremely important and well-studied primitive in graph algorithms. They were formally introduced by Peleg and Schäfer [34] in the late eighties after appearing naturally in several network problems [36]. Today, they have been successfully applied in diverse fields such as routing schemes [16, 17, 36, 39, 43], approximate shortest paths algorithms [19, 20, 9], distance oracles [9, 14, 15, 37, 44], broadcasting [25], etc. A landmark upper bound result due to Awerbuch [6] states that for any integer k, every graph has a $(2k-1)$-spanner on $O(n^{1+1/k})$ edges. Moreover, the extremely popular *girth conjecture* of Erdős [24] implies the existence of graphs for which $\Omega(n^{1+1/k})$ edges are

* This work was partially done while the authors were visiting the Simons Institute for the Theory of Computing.

necessary in any $(2k-1)$-spanner. Thus, the primary question of the optimal sparsity of a graph spanner is essentially resolved.

The next natural question in the field of spanners is to obtain efficient algorithms for computing a sparse spanner of an input graph G. This problem is well understood in the static setting; notable results include [6, 11, 38, 43]. However, in many of the above applications of spanners, the underlying graph can experience minor changes and the application requires the algorithm designer to have a spanner available at all times. Here, it is very wasteful to recompute a spanner from scratch after every modification. The challenge is instead to *dynamically maintain* a spanner under edge insertions and deletions with only a small amount of time required per update. This is precisely the problem we address in this paper.

The pioneering work on dynamic spanners was by Ausiello et al. [5], who showed how to maintain a 3- or 5-spanner with amortized update time proportional to the maximum degree Δ of the graph, i.e. for any sequence of u updates the algorithm takes time $O(u \cdot \Delta)$ in total. In sufficiently dense graphs, Δ might be $\Omega(n)$. Elkin [22] showed how to maintain a $(2k-1)$ spanner of optimal size using $\tilde{O}(mn^{-1/k})$ expected update time; i.e. *super-linear* time for dense enough graphs. Finally, Baswana et al. [10] gave fully dynamic algorithms that maintain $(2k-1)$-spanners with essentially optimal size/stretch tradeoff using *amortized* $O(k^2 \log^2 n)$ or $O(1)^k$ time per update. Their *worst-case* guarantees are much weaker: any individual update in their algorithm can require $\Omega(n)$ time. It is very notable that *every* previously known fully dynamic spanner algorithm carries the drawback of $\Omega(n)$ worst-case update time. It is thus an important open question whether this update time is an intrinsic part of the dynamic spanner problem, or whether this linear time threshold can be broken with new algorithmic ideas.

There are concrete reasons to prefer worst-case update time bounds to their amortized counterparts. In real-time systems, hard guarantees on update times are often needed to serve each request before the next one arrives. Amortized guarantees, meanwhile, can cause undesirable behavior in which the system periodically stalls on certain inputs. Despite this motivation, good worst-case update times often pose a veritable challenge to dynamic algorithm designers, and are thus significantly rarer in the literature. Historically, the fastest dynamic algorithms usually first come with amortized time bounds, and comparable worst-case bounds are achieved only after considerable research effort. For example, this was the case for the dynamic connectivity problem on undirected graphs [31] and the dynamic transitive closure problem on directed graphs [41]. In other problems, a substantial gap between amortized and worst-case algorithms remains, despite decades of research. This holds in the cases of fully dynamically maintaining minimum spanning trees [30, 27, 23], all-pairs shortest paths [18, 42], and more. Thus, strong amortized update time bounds for a problem do not at all imply the existence of strong worst-case update time bounds, and once strong amortized algorithms are found it becomes an important open problem to discover whether or not there are interesting worst-case bounds to follow.

The main result of this paper is that highly nontrivial worst-case time bounds are indeed available for fully dynamic spanners. We present the first ever algorithms that maintain spanners with essentially optimal size/stretch tradeoff and *polynomially sublinear* (in the number of nodes in the graph) worst-case update time. Our main technique is a very general new framework for boosting the performance of an orientation-based algorithm, which we hope can have applications in related dynamic problems.

1.1 Our results

We obtain fully dynamic algorithms for maintaining spanners of graphs undergoing edge insertions and deletions. In particular, in the unweighted setting we can maintain:

- a 3-spanner of size $O(n^{1+1/2}\log^{1/2} n \log\log n)$ with worst-case update time $O(n^{3/4}\log^4 n)$, or
- a 5-spanner of size $O(n^{1+1/3}\log^{2/3} n \log\log n)$ with worst-case update time $O(n^{5/9}\log^4 n)$, or
- a 5-spanner of size $O(n^{1+1/2}\log^{1/2} n \log\log n)$ with worst-case update time $O(n^{1/2}\log^4 n)$.

Naturally, these results assume that the initial graph is empty; otherwise, a lengthy initialization step is unavoidable.

Using standard techniques, these results can be extended into the setting of arbitrary positive edge weights, at the cost of an increase in the stretch by a factor of $1+\epsilon$ and an increase in the size by a factor of $\log_{1+\epsilon} W$ (for any $\epsilon > 0$, where W is the ratio between the largest and smallest edge weights).

Our algorithms are randomized and correct with high probability against an *oblivious adversary* [12] who chooses its sequence of updates independently from the random choices made by the algorithm.[1] This adversarial model is the same one used in the previous randomized algorithms with amortized update time [10]. Since the girth conjecture has been proven unconditionally for $k = 2$ and $k = 3$ [46], the first two spanners have optimal size/stretch tradeoff (up to the log factor). The third result sacrifices a non-optimal size/stretch tradeoff in exchange for improved update time.

1.2 Technical Contributions

Our main new idea is a general technique for boosting the performance of orientation-based algorithms.

Our algorithm contains three new high-level ideas. First, let $\vec{G}$ be an arbitrary orientation of the input graph G; i.e. replace every undirected edge $\{u, v\}$ by a directed edge, either (u, v) or (v, u). We give an algorithm ALG for maintaining either a 3-spanner or a 5-spanner of G with update time proportional to the maximum *out-degree* of the oriented graph $\vec{G}$. This algorithm is based on the clustering approach used in [11]. For maintaining 3- and 5-spanners we only need to consider clusters of diameter at most 2 consisting of the set of neighbors of certain cluster centers.

This alone is of course not enough, as generally the maximum out-degree of $\vec{G}$ can be as large as $n - 1$. To solve this problem, we combine ALG with the following simple out-degree reduction technique. Partition outgoing edges of every node into at most $t \leq \lceil n/s \rceil$ groups of size at most s each. For any $1 \leq i \leq t$, we combine the edges of the i-th groups and on the corresponding subgraph G_i we run an instance of ALG to maintain a 3-spanner with update time $O(s)$, the maximum out-degree in $\vec{G}_i$. By the decomposability of spanners, the union of all these sub-spanners $H_1 \cup \ldots H_t$ is a 3-spanner of G. In this way we can obtain an algorithm for maintaining a 3-spanner of size $|H_1| + \ldots |H_t| = O(n^{5/2}/s)$ with worst-case update time $O(s)$ for any $1 \leq s \leq n$. We remark that the general technique of partitioning a graph into subgraphs of low out-degree has been used before, e.g. [7]; however, our recursive conversion of these subgraphs into spanners is original and an important technical contribution of this paper.

[1] In particular, this means that the adversary is *not* allowed to see the current edges of the spanner.

The partitioning is still not enough, as the optimal size of a 3-spanner is $O(n^{3/2})$, which would then require $s = \Omega(n)$ worst-case update time. However, we can improve upon this tradeoff once more with a more fine-grained application of ALG. In particular, on each subgraph $\vec{G}_i$, ALG maintains two subgraphs A_i^1 and $\vec{B}_i^1$, such that:

- A_i^1 is a 'partial' 3-spanner of G_i of size $\tilde{O}(n^{1+1/2} \cdot s/n)$, and
- The maximum out-degree in $\vec{B}_i^1$ is considerably smaller than the maximum out-degree in $\vec{G}_i$.

We then recursively apply ALG on $\vec{B}_1^1 \cup \cdots \cup \vec{B}_t^1$ to some depth ℓ at which the out-degree can no longer be reduced by a meaningful amount. Our final spanner is then the union of all the sets A_i^j, for $1 \leq i \leq t$ and $1 \leq j \leq \ell$, as well as the "remainder" graphs $\vec{B}_1^\ell \cup \cdots \cup \vec{B}_t^\ell$, which have low out-degree and are thus sparse.

In principle, the recursive application of ALG could be problematic, as one update in G could lead to several changes to the edges in the B_i^1 subgraphs, which then propagate as an increasing number of updates in the recursive calls of the algorithm. This places another constraint on ALG. We carefully design ALG in such a way that it performs only a constant number of changes to each B_i^1 with any update in G, and we only recurse to depth $\ell = o(\log n)$ so that the total number of changes at each level is subpolynomial.

Overall, we remark that our framework for performing out-degree reduction is fairly generic, and seems likely applicable to other algorithms that admit the design of an ALG with suitable properties. The main technical challenges are designing ALG with these properties, and performing some fairly involved parameter balancing to optimize the running time used by the recursive calls. However, we do not know how to extend our approach to sparser spanners with larger stretches since corresponding constructions usually need clusters of larger diameter and maintaining such clusters with update time proportional to the maximum (out)-degree of the graph seems challenging.

1.3 Other Related Work

There has been some related work attacking the spanner problem in other models of computation. Some of the work on streaming spanner algorithms, in particular [8, 26], was converted into purely *incremental* dynamic algorithms, which maintain spanners under edge insertions but cannot handle deletions. This line of research culminated in an incremental algorithm with worst-case update time $O(1)$ per edge insertion [22]. Elkin [21] also gave a near-optimal algorithm for maintaining spanners in the distributed setting.

A concept closely related to spanners are *emulators* [19], in which the graph H for approximately preserving the distances may contain arbitrary weighted edges and is not necessarily a subgraph of G. Dynamic algorithms for maintaining emulators have been commonly used as subroutines to obtain faster dynamic algorithms for maintaining (approximate) shortest paths or distances. Some of the work on this problem includes [40, 13, 28, 29, 2, 1].

As outlined above, one of the main technical contributions of this paper is a framework for exploiting orientations of undirected graphs. The idea of orienting undirected graphs has been key to many recent advances in dynamic graph algorithms. Examples include [33, 32, 35, 4, 3].

2 Preliminaries

We consider unweighted, undirected graphs $G = (V, E)$ undergoing edge insertions and edge deletions. For all pairs of nodes u and v we denote by $d_G(u, v)$ the distance between u and v in G. An *α-spanner* of a graph $G = (V, E)$ is a subgraph $H = (V, E') \subseteq G$ such that

$d_H(u,v) \leq \alpha \cdot d_G(u,v)$ for all $u, v \in V$.[2] The parameter α is called the *stretch* of the spanner. We will use the well-known fact that it suffices to only span distances over the edges of G.

▶ **Lemma 1** (Spanner Adjacency Lemma (Folklore)). *If $H = (V, E')$ is a subgraph of $G = (V, E)$ that satisfies $d_H(u,v) \leq \alpha \cdot d_G(u,v)$ for all $(u,v) \in E$, then H is an α-spanner of G.*

We will work with *orientations* of undirected graphs. We denote an undirected edge with endpoints u and v by $\{u,v\}$ and a directed edge from u to v by (u,v). An *orientation* $\vec{G} = (V, \vec{E})$ of an undirected graph $G = (V, E)$ is a directed graph on the same set of nodes such that for every edge $\{u,v\}$ of G, $\vec{G}$ either contains the edge (u,v) or the edge (v,u). Conversely, G is the *undirected projection* of $\vec{G}$. In an undirected graph G, we denote by $N(v) := \{w \mid \{v,w\} \in G\}$ the set of neighbors of v. In an oriented graph $\vec{G}$, we denote by $\mathit{Out}(v) := \{w \mid (v,w) \in \vec{G}\}$ the set of outgoing neighbors of v. Similarly, by $\mathit{In}(v) := \{u \mid (u,v) \in \vec{G}\}$ we denote the set of incoming neighbors of v. We denote by $\Delta^+(\vec{G})$ the maximum out-degree of $\vec{G}$.

Our algorithms can easily be extended to graphs with edge weights, via the standard technique of weight binning:

▶ **Lemma 2** (Weight Binning, e.g. [10]). *Suppose there is an algorithm that dynamically maintains a spanner of an arbitrary unweighted graph with some particular size, stretch, and update time. Then for any $\epsilon > 0$, there is an algorithm that dynamically maintains a spanner of an arbitrary graph with positive edge weights, at the cost of an increase in the stretch by a factor of $1 + \epsilon$ and an increase in the update time by a factor of $O(\log_{1+\epsilon} W)$ (and no change in update time). Here, W is the ratio between the largest and smallest edge weight in the graph.*

Since this extension is already well known, we will not discuss it further. Instead, we will simplify the rest of the paper by focusing only on the unweighted setting; that is, all further graphs in this paper are unweighted and undirected.

In our algorithms, we will use the well-known fact that good hitting sets can be obtained by random sampling. This technique was first used in the context of shortest paths by Ullman and Yannakakis [45]. A general lemma on the size of the hitting set can be formulated as follows.

▶ **Lemma 3** (Hitting Sets). *Let $a \geq 1$, let V be a set of size n and let $U_1, U_2, \ldots, U_r$, be subsets of V of size at least q. Let S be a subset of V obtained by choosing each element of V independently at random with probability $p = \min(x/q, 1)$ where $x = a \ln(rn) + 1$. Then, with high probability (whp), i.e. probability at least $1 - 1/n^a$, both the following two properties hold:*

1. *For every $1 \leq i \leq r$, the set S contains a node in U_i, i.e. $U_i \cap S \neq \emptyset$.*
2. *$|S| \leq 3xn/q = O(an \ln(rn)/q)$.*

A well-known property of spanners is *decomposability.* We will exploit this property to run our dynamic algorithm on carefully chosen subgraphs.

▶ **Lemma 4** (Spanner Decomposability, [10]). *Let $G = (V, E)$ be an undirected (possibly weighted) graph, let $E_1, \ldots, E_t$ be a partition of the set of edges E, and let, for every $1 \leq i \leq t$, H_i be an α-spanner of $G_i = (V, E_i)$ for some $\alpha \geq 1$. Then $H = \bigcup_{i=1}^{t} H_i$ is an α-spanner of G.*

[2] If u and v are disconnected in G, then $d_G(u,v) = \infty$ and so they may be disconnected in the spanner as well.

In our algorithms we use a reduction for getting a fully dynamic spanner algorithm for an arbitrarily long sequence of updates from a fully dynamic spanner algorithm that only works for a polynomially bounded number of updates. This is particularly useful for randomized algorithms whose high-probability guarantees are obtained by taking a union bound over a polynomially bounded number of events.

▶ **Lemma 5** (Update Extension, Implicit in [3]). *Assume there is a fully dynamic algorithm for maintaining an α-spanner (for some $\alpha \geq 1$) of size at most $S(m, n, W)$ with worst-case update time $T(m, n, W)$ for up to $4n^2$ updates in G. Then there also is a fully dynamic algorithm for maintaining an α-spanner of size at most $O(S(m, n, W))$ with worst-case update time $O(T(m, n, W))$ for an arbitrary number of updates.*

For completeness, we give the proof of this lemma in an appendix. We remark that is is entirely identical to the one given in [3].

3 Algorithms for Partial Spanner Computation

Our goal in this section is to describe fully dynamic algorithm for *partial* spanner computation. We prove lemmas that can informally be summarized as follows: given a graph G with an orientation $\vec{G}$, one can build a very sparse spanner that only covers the edges leaving nodes with large out-degree in $\vec{G}$. There is a smooth tradeoff between the sparsity of the spanner and the out-degree threshold beyond which edges are spanned.

As a crucial subroutine, our algorithms employ a fully dynamic algorithm for maintaining certain structural information related to a *clustering* of G. We will describe this subroutine first.

3.1 Maintaining a clustering structure

In the spanner literature, a *clustering* of a graph $G = (V, E)$ is a partition of the nodes V into *clusters* $C_1, \dots, C_k$, as well as a “leftover” set of *free* nodes F, with the following properties:

- For each cluster C_i, there exists a “center” node $x_i \in V$ such that all nodes in C_i are adjacent to x_i.
- The free nodes F are precisely the nodes that are not adjacent to any cluster center.

In this paper, we will represent clusterings with a vector c indexed by V, such that for any clustered $v \in V$ we have $c[v]$ equal to its cluster center, and for any free $v \in V$ we use the convention $c[v] = \infty$.

We will use the following subroutine in our main algorithms:

▶ **Lemma 6.** *Given an oriented graph $\vec{G} = (V, \vec{E})$ and a set of cluster centers $S = \{s_1, \dots, s_k\}$, there is a fully dynamic algorithm that simultaneously maintains:*

1. *A clustering c of $G = (V, E)$ with centers S*
2. *For each node v and each cluster index $i \in \{1, \dots, k\}$, the set*

$$In(v, i) := \{u \in In(v) \mid c[u] = i\}$$

(i.e. the incoming neighbors to v from cluster i)

3. *For every pair of cluster indices $i, j \in \{1, \dots, k\}$, the set*

$$In(i, j) := \{(u, v) \in \vec{E} \mid c[u] = j, c[v] = i\}$$

(i.e. the incoming neighbors to cluster i from cluster j).

This algorithm has worst-case update time $O(\Delta^+(\vec{G})\log n)$, where $\Delta^+(\vec{G})$ is the maximum out-degree of $\vec{G}$.

The second $In(v, i)$ sets will be useful for the 3-spanner, while the third $In(i, j)$ sets will be useful for the 5-spanner.

The implementation of this lemma is extremely straightforward; it is not hard to show that the necessary data structures can be maintained in the naive way by simply passing a message along the outgoing edges from u and v whenever an edge (u, v) is inserted or deleted. Due to space constraints, we defer full implementation details and pseudocode to Appendix A.

3.2 Maintaining a partial 3-spanner

We next show how to convert Lemma 6 into a fully dynamic algorithm for maintaining a partial 3-spanner of a graph, as described in the introduction. Specifically:

▶ **Lemma 7.** *For every integer $1 \leq d \leq n$, there is a fully dynamic algorithm that takes an oriented graph $\vec{G} = (V, \vec{E})$ on input and maintains subgraphs $A = (V, E_A), \vec{B} = (V, \vec{E}_B)$ (i.e. $\vec{B}$ is oriented but A is not) over a sequence of $4n^2$ updates with the following properties:*
- *$d_A(u, v) \leq 3$ for every edge $\{u, v\}$ in $E \setminus E_B$*
- *A has size $|A| = O(n^2(\log n)/d + n)$*
- *The maximum out-degree of $\vec{B}$ is $\Delta^+(\vec{B}) \leq d$.*
- *With every update in G, at most 4 edges are changed in $\vec{B}$.*

Further, this algorithm has worst-case update time $O(\Delta^+(\vec{G})\log n)$. The algorithm is randomized, and all of the above properties hold with high probability against an oblivious adversary.

Informally, this lemma states the following. Edges leaving nodes with high out-degree are easy for us to span; we maintain A as a sparse spanner of these edges. Edges leaving nodes with low out-degree are harder for us to span, and we maintain $\vec{B}$ as a collection of these edges.

Note that this lemma is considerably *stronger* than the existence of a 3-spanner. In particular, by setting $d = \sqrt{n \log n}$ and then using $A \cup \vec{B}$ as a spanner of G, we obtain a fully dynamic algorithm for maintaining a 3-spanner:

▶ **Corollary 8.** *There is a fully dynamic algorithm for maintaining a 3-spanner of size $O(n^{1+1/2}\sqrt{\log n})$ for an oriented graph $\vec{G}$ with worst-case update time $O(\Delta^+(\vec{G})\log n)$. The stretch and the size guarantee both hold with high probability against an oblivious adversary.*

The proof is essentially immediate from Lemma 7; we omit it because it is non-essential. The detail of handling only $4n^2$ updates is not necessary in this corollary, due to Lemma 5.

Looking forward, we will wait until Lemma 4 to show precisely how the extra generality in Lemma 7 is useful towards strong worst-case update time. The rest of this subsection is devoted to the proof of Lemma 7.

3.2.1 Algorithm

It will be useful in this algorithm to fix an arbitrary ordering of the nodes in the graph. This allows us to discuss the "smallest" or "largest" node in a list, etc.

We initialize the algorithm by determining a set of cluster centers S via random sampling. Specifically, every node of G is added to S independently with probability $p = \min(x/d, 1)$

where $x = a \ln(4n^5) + 1$ for some error parameter $a \geq 1$. We then use the algorithm of Lemma 6 above to maintain a clustering with $S = \{s_1, \dots, s_k\}$ as the set of cluster centers. The subgraphs A and $\vec{B}$ are defined according to the following three rules:

1. For every clustered node v (i.e. $c[v] \neq \infty$), A contains the edge $\{v, c[v]\}$ from v to its cluster center in S.
2. For every clustered node v (i.e. $c[v] \neq \infty$) and every cluster index $1 \leq i \leq k$, A contains the edge $\{u, v\}$ to the first node $u \in \mathit{In}(v, i)$ (unless $\mathit{In}(v, i) = \emptyset$).
3. For every node u and every node v among the *first* d neighbors of u in $N(u)$ (with respect to an arbitrary fixed ordering of the nodes), $\vec{B}$ contains the edge (u, v). Alternately, if $|N(u)| \leq d$, then $\vec{B}$ contains all such edges (u, v).

We maintain the subgraph $\vec{B}$ in the following straightforward way. For every node u we store $N(u)$, the set of neighbors of u, in two self-balancing binary search trees: $N_{\leq d}(u)$ for the first d neighbors and $N_{>d}(u)$ for the remaining neighbors. Every time an edge (u, v) or an edge (v, u) is inserted into $\vec{G}$, we add v to $N_{\leq d}(u)$ and we add (u, v) to $\vec{B}$. If $N_{\leq d}(u)$ now contains more than d nodes, we remove the largest element v', add it to $N_{>d}(u)$, and remove (u, v') from $\vec{B}$.[3] Similarly, every time an edge (u, v) or an edge (v, u) is deleted from $\vec{G}$, we first check if v is contained in $N_{>d}(u)$ and if so remove it from $N_{>d}(u)$. Otherwise, we first remove v from $N_{\leq d}(u)$ and (u, v) from $\vec{B}$. Then we find the smallest node v' in $N_{>d}(u)$, remove v' from $N_{>d}(u)$, add v' to $N_{\leq d}(u)$, and add (u, v) to $\vec{B}$.

We now explain how to maintain the subgraph A. As an underlying subroutine, we use the algorithm of Lemma 6 to maintain a clustering w.r.t. centers S. On each edge insertion/deletion, we first update the clustering, and then perform the following steps:

1. For every node v for which $c[v]$ has just changed from some center s_i to some other center s_j, we remove the edge $\{v, s_i\}$ from A (if $i \neq \infty$) and add the edge $\{v, s_j\}$ to A (if $j \neq \infty$).
2. For every node u that has been added to $\mathit{In}(v, i)$ for some node v and some $1 \leq i \leq k$, we check if u is now the first node in $\mathit{In}(v, i)$. If so, we add the edge $\{u, v\}$ to A and remove the edge $\{u', v\}$ for the previous first node u' of $\mathit{In}(v, i)$ (if $\mathit{In}(v, i)$ was previously non-empty).
3. For every node u that is removed from $\mathit{In}(v, i)$ for some node v and some $1 \leq i \leq k$, we check if u was the first node in $\mathit{In}(v, i)$ and if so remove the edge $\{u, v\}$ from A and add the edge $\{u', v\}$ for the new first node u' of $\mathit{In}(v, i)$ (if $\mathit{In}(v, i)$ is still non-empty).

3.2.2 Analysis

To bound the update time required by this algorithm, we will argue that we spend $O(\Delta^+(\vec{G}) \log n)$ time per update maintaining A, and $O(\log n)$ time per update maintaining $\vec{B}$ (which, in our applications, is always dominated by $O(\Delta^+(\vec{G}) \log n)$). By Lemma 6, the clustering structure can be updated in time $O(\Delta^+(\vec{G}) \log n)$. Each operation in steps 1, 2, and 3 above can be charged to the corresponding changes in s_i and $\mathit{In}(v, i)$ and thus can also be carried out within the same $O(\Delta^+(\vec{G}) \log n)$ time bound. Updating the subgraph $\vec{B}$ takes time $O(\log n)$, since we must perform a constant number of queries and updates in the corresponding self-balancing binary search trees.

We now show that the subgraphs A and $\vec{B}$ have all of the properties claimed in Lemma 7. First, we will discuss the sparsity bounds on A and $\vec{B}$. Observe that rule 1 contributes at

[3] Note that the node v' that is removed from $N_{\leq d}(u)$ might be the node v we have added in the first place.

most n edges to A, since every node is contained in at most one cluster. Next, recall that the number of cluster centers S is $|S| = k = O(n(\log n)/d)$ (by Lemma 3, with high probability). Thus, A contains only $O(nk) = O(n^2(\log n)/d)$ edges due to rule 2. As the only edges of $\vec{B}$ come from rule 3, the maximum out-degree in $\vec{B}$ is d. The claimed sparsity bounds therefore hold. Furthermore, with every insertion or deletion of an edge $\{u, v\}$ in G, at most one edge is added to or removed from the first d neighbors of u and v, respectively. This implies that there are at most 4 changes to $\vec{B}$ with every update in G. It now only remains to show that A is a 3-spanner of $G \setminus B$.

▶ **Lemma 9.** *For up to $4n^3$ updates, $d_A(u, v) \leq 3$ for every edge $\{u, v\}$ in $E \setminus E_B$ with high probability.*

Proof. Let $\{u, v\}$ be an edge of $E \setminus E_B$. Assume without loss of generality that the edge is oriented from u to v in $\vec{G}$. As $\{u, v\}$ is not contained in B, by rule 3 above we have $|N(u)| > d$. Thus, by Lemma 3, since the cluster centers S were chosen by random sampling, with high probability there exists a cluster center in the first d outgoing neighbors of each node in all of up to $4n^3$ different versions of G (i.e. one version for each of the $4n^3$ updates considered). Therefore $c[u] = i$ for some $1 \leq i \leq k$ and, by rule 1, A contains the edge $\{u, s_i\}$. Since $c[u] = i$, and u is an incoming neighbor of v in $\vec{G}$, we have $In(v, i) \neq \emptyset$, and thus, for the first element u' of $In(v, i)$, A contains the edge $\{u', v\}$ (by rule 2). As $c[u'] = i$, A contains the edge $\{s_i, u'\}$ by rule 1. This means that A contains the edges $\{u, s_i\}$, $\{s_i, u'\}$, and $\{u', v\}$, and thus there is a path from u to v of length 3 in A as desired. ◀

This now also completes the proof of Lemma 7.

3.3 5-spanner

The 5-spanner algorithm is very similar to the 3-spanner algorithm above, but we define the edges of the spanner in a slightly different way. Instead of including an edge from each node to each cluster, we have an edge between each *pair* of clusters. Thus, the subgraphs A and $\vec{B}$ are defined according to the following three rules:

1. For every clustered node v (i.e. $c[v] \neq \infty$), A contains the edge $\{v, c[v]\}$ from v to its cluster center in S.
2. For every pair of distinct cluster indices $1 \leq i, j \leq k$, A contains the edge $\{u, v\}$, where $\{u, v\}$ is the first element in $In(i, j)$ (unless $In(i, j) = \emptyset$).
3. For every node u and every node v among the *first* d neighbors of u in $N(u)$ (with respect to an arbitrary fixed ordering of the nodes), $\vec{B}$ contains the edge (u, v). Alternately, if $|N(u)| \leq d$, then $\vec{B}$ contains all such edges (u, v)..

Beyond this slightly altered definition, we use the same approach for maintaining A and $\vec{B}$ as in the 3-spanner. The guarantee on the stretch can be proved as follows.

▶ **Lemma 10.** *For up to $4n^3$ updates, $d_A(u, v) \leq 5$ for every edge $\{u, v\}$ in $E \setminus E_B$ with high probability.*

Proof. Let $\{u, v\}$ be an edge of $E \setminus E_B$. Assume without loss of generality that the edge is oriented from u to v in $\vec{G}$. As $\{u, v\}$ is not contained in B, by rule 3 above we have $|N(v)| > d$. We now apply Lemma 3 to argue that there is a cluster center in the first d outgoing neighbors of each node in up to $4n^3$ versions of the graph (one version for each update to be considered). Thus, $N(v)$ contains a cluster center from S with high probability. Therefore $c[v] = i$ for some $1 \leq i \leq k$ and, by rule 1, A contains the edge $\{v, s_i\}$. By the same argument, $N(u)$ contains a cluster center from S with high probability and thus A contains an edge $\{u, s_j\}$ where $c[u] = j$ for some $1 \leq j \leq k$. Since $c[v] = i$, $c[u] = j$, and u is an incoming neighbor of v in $\vec{G}$, we have $In(i, j) \neq \emptyset$, and thus, for the first element (u', v') of $In(i, j)$, A contains the edge $\{u', v'\}$ (by rule 2). As $c[v'] = i$, $c[u'] = j$, A contains the edges $\{v', s_i\}$ and $\{u', s_j\}$ by rule 1. This means that A contains the edges $\{u, s_i\}$, $\{s_i, u'\}$, $\{u', v'\}$, $\{v', s_i\}$ and $\{s_i, v\}$, and thus there is a path from u to v of length 5 in A as desired. ◀

Note that in this proof we exploit the fact that we have cluster centers for both u and v whenever the edge $\{u, v\}$ is missing. This motivates our design choice for considering the whole neighborhood of a node to determine its cluster. If we only considered cluster centers in the outgoing neighbors of a node, the resulting clustering would still be good enough for the 3-spanner, but the argument above for the 5-spanner would break down.

All other properties of the 5-spanner can be proved in an essentially identical manner to the 3-spanner. We can summarize the obtained guarantees as follows.

▶ **Lemma 11.** *For every integer $1 \leq d \leq n$, there is a fully dynamic algorithm that takes an oriented graph $\vec{G} = (V, \vec{E})$ on input and maintains subgraphs $A = (V, E_A), \vec{B} = (V, \vec{E}_B)$ (i.e. $\vec{B}$ is oriented but A is not) over a sequence of $4n^2$ updates with the following properties:*

- *$d_A(u, v) \leq 5$ for every edge $\{u, v\}$ in $E \setminus E_B$*
- *A has size $|A| = O((n^2 \log^2 n)/d^2 + n)$*
- *The maximum out-degree of $\vec{B}$ is $\Delta^+(\vec{B}) \leq d$.*
- *With every update in G, at most 4 edges are changed in $\vec{B}$.*

Further, this algorithm has worst-case update time $O(\Delta^+(\vec{G}) \log n)$. The algorithm is randomized, and all of the above properties hold with high probability against an oblivious adversary.

Once again, this lemma generalizes the construction of a sparse 5-spanner. By setting $d = (n \log n)^{2/3}$ we can obtain:

▶ **Corollary 12.** *There is a fully dynamic algorithm for maintaining a 5-spanner of size $O(n^{1+1/3} \log^{2/3} n)$ for an oriented graph $\vec{G}$ with worst-case update time $O(\Delta^+(\vec{G}) \log n)$. The stretch and the size guarantee both hold with high probability against an oblivious adversary.*

4 Out-degree Reduction for Improved Update Time

Our goal is now to use Lemmas 7 and 11 to obtain spanner algorithms with sublinear update time. Since we obtain our 3-spanner and 5-spanner in an essentially identical manner, we will explain only the 3-spanner in full detail, and then sketch the 5-spanner construction.

We next establish the following simple generalization of Lemma 7:

▶ **Lemma 13.** *For every integer $1 \leq s \leq n$ and $1 \leq d \leq n$, there is a fully dynamic algorithm that takes an oriented graph $\vec{G} = (V, \vec{E})$ on input and maintains subgraphs $A = (V, E_A), \vec{B} = (V, \vec{E}_B)$ (i.e. $\vec{B}$ is oriented but A is not) over a sequence of $4n^2$ updates with the following properties:*

- *$d_A(u, v) \leq 3$ for every edge $\{u, v\}$ in $E \setminus E_B$*
- *A has size $|A| = O(\Delta^+(\vec{G}) n^2 (\log n)/(sd))$*
- *The maximum out-degree of $\vec{B}$ is $\Delta^+(\vec{B}) \leq \Delta^+(\vec{G}) \cdot d/s$.*
- *With every update in G, at most 4 edges are changed in $\vec{B}$.*

Further, this algorithm has worst-case update time $O(s \log n)$. The algorithm is randomized, and all of the above properties hold with high probability against an oblivious adversary.

In particular, Lemma 7 is the special case of this lemma in which $s = \Delta^+(\vec{G})$.

Proof. We orient each incoming edge of G in an arbitrary way. We then maintain a partitioning of the (oriented) edges of G into $t := \lceil \Delta^+(\vec{G})/s \rceil$ groups, such that in each group each node has at most s outgoing edges. Specifically, we perform this partitioning by maintaining the current out-degree of each node u in $\vec{G}$, and we assign a new edge (u, v) which is the x^{th} edge leaving u in $\vec{G}$ to the subgraph $\vec{G}_{\lceil x/s \rceil}$. In this way, we form t subgraphs $\vec{G}_1, \dots \vec{G}_t$ of $\vec{G}$, each of which has $\Delta^+(\vec{G}_i) \leq s$.

We now run the algorithm of Lemma 7 on each $\vec{G}_i$ to maintain, for each $1 \leq i \leq t$, two subgraphs A_i and $\vec{B}_i$ as specified in the lemma. Let $A = \bigcup A_i$ and $\vec{B} = \bigcup \vec{B}_i$ denote the unions of these subgraphs.

Observe that every update in G only changes exactly one of the subgraphs $\vec{G}_i$ and thus only must be executed in one corresponding instance of the algorithm of Lemma 7. As we have "artificially" bounded the maximum out-degree of every subgraph $\vec{G}_i$ by s, the claimed bounds on the update time and the properties of A and $\vec{B}$ now follow simply from Lemma 7. ◀

We now recursively apply the "out-degree reduction" of the previous lemma to obtain subgraphs $\vec{B}$ of smaller and smaller out-degree. Finally, at bottom level, the maximum out-degree is small enough that we can apply a "regular" spanner algorithm to it.

▶ **Theorem 14.** *There is a fully dynamic algorithm for maintaining a 3-spanner of size $O(n^{1+1/2} \log^{1/2} n \log \log n)$ with worst-case update time $O(n^{3/4} \log^4 n)$.*

Proof. Our spanner construction is as follows (we temporarily omit details related to parameter choices, which influence the resulting update time). Apply Lemma 13 to obtain subgraphs $A_1, \vec{B}_1$. Include all edges in A in the spanner, and then recursively apply Lemma 13 to $\vec{B}$ to obtain $A_2, \vec{B}_2$. Repeat to depth ℓ (for some parameter ℓ that will be chosen later). At bottom level, instead of recursing, we apply the algorithm from Corollary 8 to obtain a 3-spanner of $\vec{B}$.

More formally, we set $\vec{B}_0 = \vec{G}_0$, and for every $1 \leq j \leq \ell$ we let A_j and $\vec{B}_j$ be the graphs maintained by the algorithm of Lemma 13 on input $\vec{B}_{j-1}$ using parameters s and d_j to be chosen later.[4] Further, we let H' be the spanner maintained by the algorithm of Corollary 8 on input $\vec{B}_\ell$. The resulting graph maintained by our algorithm is $H = \bigcup_{1 \leq j \leq \ell} A_j \cup H'$. Then, by Lemma 13, we have the following properties for every $1 \leq j \leq \ell$:

- $d_{A_j}(u, v) \leq 3$ for every edge $\{u, v\}$ in $B_{j-1} \setminus B_j$
- A_j has size $|A_j| = O(\Delta^+(\vec{B}_{j-1}) n^2 (\log n)/(s d_j))$
- The maximum out-degree of $\vec{B}_j$ is $\Delta^+(\vec{B}_j) \leq \Delta^+(\vec{B}_{j-1}) \cdot d_j/s$.
- With every update in $\vec{B}_{j-1}$, at most 4 edges are changed in $\vec{B}_j$.

It is straightforward to see that the resulting graph H is a 3-spanner of G: At each level j of the recursion, A_j spans all edges of B_{j-1} *except* those that appear in the current subgraph $\vec{B}_j$. Thus, at bottom level, the only non-spanned edges of G are those in the final subgraph $\vec{B}_\ell$. For these edges we explicitly add a 3-spanner H' of $\vec{B}_\ell$ to H. By Lemma 1, this suffices to produce a 3-spanner of all of G.

Now that we have correctness of the construction, it remains to bound the number of edges in the output spanner. First, observe that, by induction,

$$\Delta^+(\vec{B}_j) \leq n \cdot \prod_{1 \leq j' \leq j} d_{j'}/s^j$$

[4] Note that the parameter s is the same for all levels of the recursion, whereas the parameter d_j is not.

for all $1 \leq j \leq \ell$. Since additionally H' has size $O(n^{1+1/2}\log^{1/2} n)$ by Corollary 8, the total number of edges in H is

$$|H| = \sum_{1\leq j\leq \ell} |A_j| + |H'| \leq \sum_{1\leq j\leq \ell} O\left(\frac{\Delta^+(B_{j-1})n^2\log n}{sd_j}\right) + O(n^{1+1/2}\log^{1/2} n)$$
$$\leq \sum_{1\leq j\leq \ell} O\left(\frac{\left(\prod_{1\leq j'\leq j-1} d_{j'}\right) n^3 \log n}{s^j d_j}\right) + O(n^{1+1/2}\log^{1/2} n)\,.$$

Thus, our spanner satisfies the claimed sparsity bound so long as the union of all ℓ of the A_j subgraphs fit within the claimed sparsity bound; this will be the case if we balance all summands.

We next bound the update time of our algorithm. Each change to some $\vec{B}_j$ causes at most 4 changes in the next level $\vec{B}_{j+1}$, and thus the number of changes to $\vec{B}_j$ can propagate exponentially. Thus, for every $0 \leq j \leq \ell - 1$, a single update in $\vec{G}$ could cause at most 4^j changes to $\vec{B}_j$. Each of the ℓ instances of the algorithm of Lemma 13 has a worst-case update time of $O(s\log n)$ and the algorithm of Corollary 8 has a worst-case update time of $\Delta^+(\vec{B}_\ell \log n)$. Since

$$\Delta^+(\vec{B}_\ell) \leq n \cdot \prod_{1\leq j\leq \ell} d_j/s^\ell$$

the worst-case update time of our overall algorithm is

$$O\left(\left(\sum_{j=0}^{\ell-1} 4^j s + 4^\ell \Delta^+(\vec{B}_\ell)\right)\cdot \log n\right) \leq O\left(\left(s + \frac{n\cdot \prod_{1\leq j\leq \ell} d_j}{s^\ell}\right)\cdot 4^\ell \log n\right)\,.$$

Our goal is now to choose parameters s_j, d, ℓ to minimize this expression subject to the constraint on spanner size given above. To achieve this, we set parameters as follows:

$$\begin{aligned} \ell &= \log\log n\,, \\ s &= n^{(3\cdot 2^\ell - 1)/(2^{\ell+2}-2)}\log n\,,\ \text{and} \\ d_j &= n^{(3\cdot 2^\ell - 2^{j-1} - 1)/(2^{\ell+2}-2)}\log n\,. \end{aligned}$$

These values were obtained with the help of a computer algebra solver, so we do not have explicit computations to show for them. ◀

We now turn our attention to the 5-spanner. Similar to Lemma 13 above, we can use Lemma 11 to perform a similar out-degree reduction step for our dynamic 5-spanner algorithm.

▶ **Lemma 15.** *For every integer $1 \leq s \leq n$ and $1 \leq d \leq n$, there is a fully dynamic algorithm that takes an oriented graph $\vec{G} = (V, \vec{E})$ on input and maintains subgraphs $A = (V, E_A), \vec{B} = (V, \vec{E}_B)$ (i.e. $\vec{B}$ is oriented but A is not) over a sequence of $4n^2$ updates with the following properties:*

- *$d_A(u, v) \leq 5$ for every edge $\{u, v\}$ in $E \setminus E_B$*
- *A has size $|A| = O(\Delta^+(\vec{G})n^2(\log^2 n)/(sd^2))$*
- *The maximum out-degree of $\vec{B}$ is $\Delta^+(\vec{B}) \leq \Delta^+(\vec{G})\cdot d/s$.*
- *With every update in G, at most 4 edges are changed in $\vec{B}$.*

Further, this algorithm has worst-case update time $O(s \log n)$. The algorithm is randomized, and all of the above properties hold with high probability against an oblivious adversary.

The proof of this lemma is essentially identical to the proof of Lemma 13 and has thus been omitted.

Just as in the case of the 3-spanner, we use this lemma to show:

▶ **Theorem 16.** *There is a fully dynamic algorithm for maintaining a 5-spanner of size $O(n^{1+1/3} \log^{2/3} n \log\log n)$ with worst-case update time $O(n^{5/9} \log^4 n)$.*

Proof. The proof is identical to the proof of Theorem 14, except that the proper parameter balance is now:

$$\begin{aligned} \ell &= \log\log n\,, \\ s &= n^{(5\cdot 3^{\ell} - 2^{\ell+1})/(3^{\ell+2} - 3\cdot 2^{\ell+1})} \log n\,, \text{ and} \\ d_j &= n^{(5\cdot 3^{\ell} - 3^{j-1} 2^{\ell-j+2} - 2^{\ell+1})/(3^{\ell+2} - 3\cdot 2^{\ell+1})} \log n\,. \end{aligned}$$

◀

Finally, we can also show:

▶ **Theorem 17.** *There is a fully dynamic algorithm for maintaining a 5-spanner of size $O(n^{1+1/2} \log^{1/2} n \log\log n)$ with worst-case update time $O(n^{1/2} \log^4 n)$.*

Proof. The proof is identical to the proof of Theorems 14 and 16, except that we now use the parameter balance

$$\begin{aligned} \ell &= \log\log n\,, \\ s &= n^{(3^{\ell+1} - 2^{\ell})/(2\cdot 3^{\ell+1} - 2^{\ell+2})} \log n\,, \text{ and} \\ d_j &= n^{(3^{\ell+1} - 3^{j}\cdot 2^{\ell-j} - 2^{\ell})/(2\cdot 3^{\ell+1} - 2^{\ell+2})} \log n\,. \end{aligned}$$

and we maintain the dynamic 3-spanner H' of size $O(n^{1+1/2} \log^{1/2} n)$ from Corollary 12 at bottom level. ◀

This spanner has non-optimal size/stretch tradeoff, but enjoys the best worst-case update time that we are currently able to construct.

Acknowledgements. We want to thank Seeun William Umboh for many fruitful discussions at Simons.

References

1. Ittai Abraham and Shiri Chechik. Dynamic decremental approximate distance oracles with $(1+\epsilon, 2)$ stretch. *CoRR*, abs/1307.1516, 2013. URL: `http://arxiv.org/abs/1307.1516`.
2. Ittai Abraham, Shiri Chechik, and Kunal Talwar. Fully dynamic all-pairs shortest paths: Breaking the $o(n)$ barrier. In *International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 1–16, 2014. `doi:10.4230/LIPIcs.APPROX-RANDOM.2014.1`.
3. Ittai Abraham, David Durfee, Ioannis Koutis, Sebastian Krinninger, and Richard Peng. On fully dynamic graph sparsifiers. *CoRR*, abs/1604.02094, 2015. URL: `http://arxiv.org/abs/1604.02094`.
4. Amihood Amir, Tsvi Kopelowitz, Avivit Levy, Seth Pettie, Ely Porat, and B. Riva Shalom. Online dictionary matching with one gap. *CoRR*, abs/1503.07563, 2015. URL: `http://arxiv.org/abs/1503.07563`.

5 Giorgio Ausiello, Paolo Giulio Franciosa, and Giuseppe F. Italiano. Small stretch spanners on dynamic graphs. *Journal of Graph Algorithms and Applications*, 10(2):365–385, 2006. Announced at ESA'05. doi:10.7155/jgaa.00133.

6 Baruch Awerbuch. Complexity of network synchronization. *Journal of the ACM*, 32(4):804–823, 1985. doi:10.1145/4221.4227.

7 Leonid Barenboim and Michael Elkin. *Distributed Graph Coloring: Fundamentals and Recent Developments*, chapter Arbdefective Coloring. Morgan and Claypool, 2013.

8 Surender Baswana. Streaming algorithm for graph spanners—single pass and constant processing time per edge. *Information Processing Letters*, 106(3):110–114, 2008. doi:10.1016/j.ipl.2007.11.001.

9 Surender Baswana and Telikepalli Kavitha. Faster algorithms for all-pairs approximate shortest paths in undirected graphs. *SIAM Journal on Computing*, 39(7):2865–2896, 2010. Announced at FOCS'10. doi:10.1137/080737174.

10 Surender Baswana, Sumeet Khurana, and Soumojit Sarkar. Fully dynamic randomized algorithms for graph spanners. *ACM Transactions on Algorithms*, 8(4):35:1–35:51, 2012. Announced at ESA'06, and SODA'08. doi:10.1145/2344422.2344425.

11 Surender Baswana and Sandeep Sen. A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. *Random Structures & Algorithms*, 30(4):532–563, 2007. Announced at ICALP'03. doi:10.1002/rsa.20130.

12 Shai Ben-David, Allan Borodin, Richard M. Karp, Gábor Tardos, and Avi Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11(1):2–14, 1994. Announced at STOC'90. doi:10.1007/BF01294260.

13 Aaron Bernstein and Liam Roditty. Improved dynamic algorithms for maintaining approximate shortest paths under deletions. In *Symposium on Discrete Algorithms (SODA)*, pages 1355–1365, 2011. doi:10.1137/1.9781611973082.104.

14 Shiri Chechik. Approximate distance oracles with constant query time. In *Symposium on Theory of Computing (STOC)*, pages 654–663, 2014. doi:10.1145/2591796.2591801.

15 Shiri Chechik. Approximate distance oracles with improved bounds. In *Symposium on Theory of Computing (STOC)*, pages 1–10, 2015. doi:10.1145/2746539.2746562.

16 Lenore Cowen. Compact routing with minimum stretch. *Journal of Algorithms*, 38(1):170–183, 2001. Announced at SODA'99. doi:10.1006/jagm.2000.1134.

17 Lenore Cowen and Christopher G. Wagner. Compact roundtrip routing in directed networks. *Journal of Algorithms*, 50(1):79–95, 2004. Announced at PODC'00. doi:10.1016/j.jalgor.2003.08.001.

18 Camil Demetrescu and Giuseppe F. Italiano. A new approach to dynamic all pairs shortest paths. *Journal of the ACM*, 51(6):968–992, 2004. Announced at STOC'03. doi:10.1145/1039488.1039492.

19 Dorit Dor, Shay Halperin, and Uri Zwick. All-pairs almost shortest paths. *SIAM Journal on Computing*, 29(5):1740–1759, 2000. Announced at FOCS'96. doi:10.1137/S0097539797327908.

20 Michael Elkin. Computing almost shortest paths. *ACM Transactions on Algorithms*, 1(2):283–323, 2005. Announced at PODC'01. doi:10.1145/1103963.1103968.

21 Michael Elkin. A near-optimal distributed fully dynamic algorithm for maintaining sparse spanners. In *Symposium on Principles of Distributed Computing (PODC)*, pages 185–194, 2007. doi:10.1145/1281100.1281128.

22 Michael Elkin. Streaming and fully dynamic centralized algorithms for constructing and maintaining sparse spanners. *ACM Transactions on Algorithms*, 7(2):20:1–20:17, 2011. Announced at ICALP'07. doi:10.1145/1921659.1921666.

23 David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Amnon Nissenzweig. Sparsification—a technique for speeding up dynamic graph algorithms. *Journal of the ACM*, 44(5):669–696, 1997. Announced at FOCS'92. doi:10.1145/265910.265914.

24 Paul Erdős. Extremal problems in graph theory. *Theory of graphs and its applications*, pages 29–36, 1964.

25 Arthur M. Farley, Andrzej Proskurowski, Daniel Zappala, and Kurt Windisch. Spanners and message distribution in networks. *Discrete Applied Mathematics*, 137(2):159–171, 2004.

26 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. Graph distances in the data-stream model. *SIAM Journal on Computing*, 38(5):1709–1727, 2008. doi:10.1137/070683155.

27 Greg N. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM Journal on Computing*, 14(4):781–798, 1985. Announced at STOC'83. doi:10.1137/0214055.

28 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Decremental single-source shortest paths on undirected graphs in near-linear total update time. In *Symposium on Foundations of Computer Science (FOCS)*, pages 146–155, 2014. doi:10.1109/FOCS.2014.24.

29 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. A subquadratic-time algorithm for dynamic single-source shortest paths. In *Symposium on Discrete Algorithms (SODA)*, pages 1053–1072, 2014. doi:10.1137/1.9781611973402.79.

30 Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *Journal of the ACM*, 48(4):723–760, 2001. Announced at STOC'98. doi:10.1145/502090.502095.

31 Bruce M. Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *Symposium on Discrete Algorithms (SODA)*, pages 1131–1142, 2013. doi:10.1137/1.9781611973105.81.

32 Tsvi Kopelowitz, Robert Krauthgamer, Ely Porat, and Shay Solomon. Orienting fully dynamic graphs with worst-case time bounds. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 532–543, 2014. doi:10.1007/978-3-662-43951-7_45.

33 Ofer Neiman and Shay Solomon. Simple deterministic algorithms for fully dynamic maximal matching. In *Symposium on Theory of Computing (STOC)*, pages 745–754, 2013. doi:10.1145/2488608.2488703.

34 David Peleg and Alejandro A. Schäffer. Graph spanners. *Journal of Graph Theory*, 13(1):99–116, 1989. doi:10.1002/jgt.3190130114.

35 David Peleg and Shay Solomon. Dynamic $(1+\epsilon)$-approximate matchings: A density-sensitive approach. In *Symposium on Discrete Algorithms (SODA)*, pages 712–729, 2016. doi:10.1137/1.9781611974331.ch51.

36 David Peleg and Jeffrey D. Ullman. An optimal synchronizer for the hypercube. *SIAM Journal of Computing*, 18(4):740–747, 1989. Announced at PODC'87. doi:10.1137/0218050.

37 Mihai Pătraşcu and Liam Roditty. Distance oracles beyond the Thorup-Zwick bound. *SIAM Journal on Computing*, 43(1):300–311, 2014. Announced at FOCS'10. doi:10.1137/11084128X.

38 Liam Roditty, Mikkel Thorup, and Uri Zwick. Deterministic constructions of approximate distance oracles and spanners. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 261–272, 2005. doi:10.1007/11523468_22.

39 Liam Roditty, Mikkel Thorup, and Uri Zwick. Roundtrip spanners and roundtrip routing in directed graphs. *ACM Transactions on Algorithms*, 4(3), 2008. Announced at SODA'02. doi:10.1145/1367064.1367069.

40 Liam Roditty and Uri Zwick. Dynamic approximate all-pairs shortest paths in undirected graphs. *SIAM Journal on Computing*, 41(3):670–683, 2012. Announced at FOCS'04. doi:10.1137/090776573.

41 Piotr Sankowski. Dynamic transitive closure via dynamic matrix inverse. In *Symposium on Foundations of Computer Science (FOCS)*, pages 509–517, 2004. doi:10.1109/FOCS.2004.25.

42 Mikkel Thorup. Worst-case update times for fully-dynamic all-pairs shortest paths. In *Symposium on Theory of Computing (STOC)*, pages 112–119, 2005. doi:10.1145/1060590.1060607.

43 Mikkel Thorup and Uri Zwick. Compact routing schemes. In *Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 1–10, 2001. doi:10.1145/378580.378581.

44 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *Journal of the ACM*, 52(1):74–92, 2005. Announced at STOC'01. doi:10.1145/1044731.1044732.

45 Jeffrey D. Ullman and Mihalis Yannakakis. High-probability parallel transitive-closure algorithms. *SIAM Journal on Computing*, 20(1):100–125, 1991. Announced at SPAA'90. doi:10.1137/0220006.

46 Rephael Wenger. Extremal graphs with no C^4's, C^6's, or C^{10}'s. *Journal of Combinatorial Theory B, Series B*, 52(1):113–116, 1991. doi:10.1016/0095-8956(91)90097-4.

A Updating the Clustering

In the following we give the straightforward algorithm for maintaining the clustering with worst-case update time proportional to the maximum out-degree of the original graph mentioned in Section 3.1. To make the presentation of this algorithm more succinct we assume that there is some (arbitrary, but fixed) ordering on the nodes. Furthermore, we assume that the nodes of S are given according to this order, i.e. $s_1 \leq s_2 \leq \cdots \leq s_k$. For every node v, we maintain $c[v]$ as the smallest i such that s_i is a neighbor of v (or ∞ if no such neighbor exists). Additionally, we naturally extend the sets $In(v, i)$ and $In(i, j)$ to the case $i, j \in \{1, \ldots, k, \infty\}$.

We begin with an empty graph $G = (V, \emptyset)$, a cluster vector c with $c[v] = \infty$ for all $v \in V$, and empty sets $In(i, j)$ and $In(i, v)$ for all cluster indices $1 \leq i, j \leq k$ and $v \in V$. We then modify these data structures under edge insertions and deletions as follows.

Correctness of the algorithms that follow is immediate, and is not shown formally.

A.1 Insertion of an edge (u, v)

- Add u to $In(v, i)$ for $i = c[u]$.
- Add (u, v) to $In(j, i)$ for $i = c[u]$ and $j = c[v]$
- If $u = s_i$ for some $1 \leq i \leq k$:
  - Set $j = c[v]$ (might be ∞)
  - Add u to $C[v]$.
  - If $i < j$:
    * Set $c[v] = i$
    * For every outgoing neighbor v' of v:
      Remove v from $In(v', j)$ and add v to $In(v', i)$.
      Remove (v, v') from $In(i', j)$ and add (v, v') to $In(i', i)$ where $i' = c[v']$.
- If $v = s_i$ for some $1 \leq i \leq k$:
  - Set $j = c[u]$ (might be ∞)
  - Add v to $C[u]$.

  - If $i < j$:
    * Set $c[u] = i$
    * For every outgoing neighbor v' of u:
      Remove u from $In(v', j)$ and add u to $In(v', i)$.
      Remove (u, v') from $In(i', j)$ and add (u, v') to $In(i', i)$ where $i' = c[v']$.

A.2 Deletion of an edge (u, v):

- Remove u from $In(v, i)$ for $i = c[u]$.
- Remove (u, v) from $In(j, i)$ for $i = c[u]$ and $j = c[v]$
- If $u = s_i$ for some $1 \leq i \leq k$:
  - Remove u from $C[v]$.
  - If $c[v] = s_i$:
    * Let j be minimal such that s_j is in $C[v]$ (might be ∞)
    * Set $c[v] = j$
    * For every outgoing neighbor v' of v:
      Remove v from $In(v', i)$ and add v to $In(v', j)$.
      Remove (v, v') from $In(i', j)$ and add (v, v') to $In(i', i)$ where $i' = c[v']$
- If $v = s_i$ for some $1 \leq i \leq k$:
  - Remove v from $C[u]$.
  - If $c[u] = s_i$:
    * Let j be minimal such that s_j is in $C[u]$ (might be ∞)
    * Set $c[u] = j$
    * For every outgoing neighbor v' of u:
      Remove u from $In(v', i)$ and add u to $In(v', j)$.
      Remove (u, v') from $In(i', j)$ and add (u, v') to $In(i', i)$ where $i' = c[v']$

B Proof of Lemma 5

We exploit the decomposability of spanners. We maintain a partition of G into two disjoint subgraphs G_1 and G_2 and run two instances A_1 and A_2 of the dynamic algorithm on G_1 and G_2, respectively. These two algorithms maintain a t-spanner of H_1 of G_1 and a t-spanner H_2 of G_2. By Lemma 4, the union $H = H_1 \cup H_2$ is a t-spanner of $G = G_1 \cup G_2$.

We divide the sequence of updates into phases of length n^2 each. In each phase of updates one of the two instances A_1, A_2 is in the state *growing* and the other one is in the state *shrinking*. A_1 and A_2 switch their states at the end of each phase. In the following we describe the algorithm's actions during one phase. Assume without loss of generality that, in the phase we are fixing, A_1 is growing and A_2 is shrinking.

At the beginning of the phase we restart the growing instance A_1. We will orchestrate the algorithm in such a way that at the beginning of the phase G_1 is the empty graph and $G_2 = G$. After every update in G we execute the following steps:

1. If the update was the insertion of some edge e, then e is added to the graph G_1 and this insertion is propagated to the *growing* instance A_1.
2. If the update was the deletion of some edge e, then e is removed from the graph G_i it is contained in and this deletion is propagated to the corresponding instance A_i.
3. In addition to processing the update in G, if G_2 is non-empty, then one arbitrary edge e is first removed from G_2 and deleted from instance A_2 and then added to G_1 and inserted into instance A_1.

Observe that these rules indeed guarantee that G_1 and G_2 are disjoint and together contain all edges of G. Furthermore, since the graph G_2 of the shrinking instance has at most n^2 edges at the beginning of the phase, the length of n^2 updates per phase guarantees that G_2 is empty at the end of the phase. Thus, the growing instance always starts with an empty graph G_1.

As both H_1 and H_2 have size at most $S(n, m, W)$, the size of $H = H_1 \cup H_2$ is $O(S(n, m, W))$. With every update in G we perform at most 2 updates in each of A_1 and A_2. It follows that the worst-case update time of our overall algorithm is $O(T(m, n, W))$. Furthermore since each of the instances A_1 and A_2 is restarted every other phase, each instance of the dynamic algorithm sees at most $4n^2$ updates before it is restarted.

Fixed-Parameter Approximability of Boolean MinCSPs[*]

Édouard Bonnet[1], László Egri[†2], and Dániel Marx[3]

1 Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), Budapest, Hungary
bonnet.edouard@sztaki.mta.hu

2 School of Computing Science, Simon Fraser University, Burnaby, Canada
laszlo.egri@mail.mcgill.ca

3 Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), Budapest, Hungary
dmarx@cs.bme.hu

Abstract

The minimum unsatisfiability version of a constraint satisfaction problem (MinCSP) asks for an assignment where the number of unsatisfied constraints is minimum possible, or equivalently, asks for a minimum-size set of constraints whose deletion makes the instance satisfiable. For a finite set Γ of constraints, we denote by MinCSP(Γ) the restriction of the problem where each constraint is from Γ. The polynomial-time solvability and the polynomial-time approximability of MinCSP(Γ) were fully characterized by Khanna et al. [33]. Here we study the fixed-parameter (FP-) approximability of the problem: given an instance and an integer k, one has to find a solution of size at most $g(k)$ in time $f(k) \cdot n^{O(1)}$ if a solution of size at most k exists. We especially focus on the case of constant-factor FP-approximability. Our main result classifies each finite constraint language Γ into one of three classes: (1) MinCSP(Γ) has a constant-factor FP-approximation; (2) MinCSP(Γ) has a (constant-factor) FP-approximation if and only if Nearest Codeword has a (constant-factor) FP-approximation; (3) MinCSP(Γ) has no FP-approximation, unless FPT = W[P]. We show that problems in the second class do not have constant-factor FP-approximations if both the Exponential-Time Hypothesis (ETH) and the Linear PCP Conjecture (LPC) hold. We also show that such an approximation would imply the existence of an FP-approximation for the k-Densest Subgraph problem with ratio $1 - \epsilon$ for any $\epsilon > 0$.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases constraint satisfaction problems, approximability, fixed-parameter tractability

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.18

1 Introduction

Satisfiability problems and, more generally, Boolean constraint satisfaction problems (CSPs) are basic algorithmic problems arising in various theoretical and applied contexts. An instance of a Boolean CSP consists of a set of Boolean variables and a set of constraints; each

[*] This work was supported by the European Research Council (ERC) starting grant "PARAMTIGHT: Parameterized complexity and the search for tight complexity results" (reference 280152) and OTKA grant NK105645.

[†] The second author was supported by NSERC.

constraint restricts the allowed combination of values that can appear on a certain subset of variables. In the decision version of the problem, the goal is to find an assignment that simultaneously satisfies every constraint. One can also define optimization versions of CSPs: the goal can be to find an assignment that maximizes the number of satisfied constraints, minimizes the number of unsatisfied constraints, maximizes/minimizes the weight (number of 1s) of the assignment, etc. [19].

Since these problems are usually NP-hard in their full generality, a well-established line of research is to investigate how the complexity of the problem changes for restricted versions of the problem. A large body of research deals with language-based restrictions: given any finite set Γ of Boolean constraints, one can consider the special case where each constraint is restricted to be a member of Γ. The ultimate research goal of this approach is to prove a *dichotomy theorem*: a complete classification result that specifies for each finite constraint set Γ whether the restriction to Γ yields and easy or hard problem. Numerous classification theorems of this form have been proved for various decision and optimization versions for Boolean and non-Boolean CSPs [46, 13, 10, 11, 9, 12, 8, 26, 32, 34, 47, 38]. In particular, for MinCSP(Γ), which is the optimization problem asking for an assignment minimizing the number of unsatisfied constraints, Creignou et al. [19] obtained a classification of the approximability for every finite Boolean constraint language Γ. The goal of this paper is to characterize the approximability of Boolean MinCSP(Γ) with respect to the more relaxed notion of fixed-parameter approximability.

Parameterized complexity [27, 29, 23] analyzes the running time of a computational problem not as a univariate function of the input size n, but as a function of both the input size n and a relevant parameter k of the input. For example, given a MinCSP instance of size n where we are looking for a solution satisfying all but k of the constraints, it is natural to analyze the running time of the problem as a function of both n and k. We say that a problem with parameter k is *fixed-parameter tractable (FPT)* if it can be solved in time $f(k) \cdot n^{O(1)}$ for some computable function f depending only on k. Intuitively, even if f is, say, an exponential function, this means that problem instances with “small” k can be solved efficiently, as the combinatorial explosion can be confined to the parameter k. This can be contrasted with algorithms with running time of the form $n^{O(k)}$ that are highly inefficient even for small values of k. There are hundreds of parameterized problems where brute force gives trivial $n^{O(k)}$ algorithms, but the problem can be shown to be FPT using nontrivial techniques; see the recent textbooks by Downey and Fellows [27] and by Cygan et al. [23]. In particular, there are fixed-parameter tractability results and characterization theorems for various CSPs [38, 13, 35, 36].

The notion of fixed-parameter tractability has been combined with the notion of approximability [16, 17, 28, 14, 18]. Following [16, 39], we say that a minimization problem is *fixed-parameter approximable (FPA)* if there is an algorithm that, given an instance and an integer k, in time $f_1(k) \cdot n^{O(1)}$ either returns a solution of cost at most $f_2(k) \cdot k$, or correctly states that there is no solution of cost at most k. The two crucial differences compared to the usual setup of polynomial-time approximation is that (1) the running time is not polynomial, but can have an arbitrary factor $f(k)$ depending only on k and (2) the approximation ratio is defined not as a function of the input size n but as a function of k. In this paper, we mostly focus on the case of constant-factor FPA, that is, when $f_2(k) = c$ for some constant c.

Schaefer’s Dichotomy Theorem [46] identified six classes of finite Boolean constraint languages (0-valid, 1-valid, Horn, dual-Horn, bijunctive, affine) for which the decision CSP is polynomial-time solvable, and shows that every language Γ outside these classes yields NP-hard problems. Therefore, one has to study MinCSP only within these six classes, as it is otherwise already NP-hard to decide if the optimum is 0 or not, making approximation or fixed-parameter tractability irrelevant. Within these classes, polynomial-time approximability and fixed-parameter tractability seem to appear in orthogonal ways: the classes where we have positive results for one approach is very different from the classes where the other approach helps. For example, 2CNF Deletion (also called Almost 2SAT) is fixed-parameter tractable [45, 37], but has no polynomial-time approximation algorithm with constant approximation ratio, assuming the Unique Games Conjecture [15]. On the other hand, if Γ consists of the three constraints (x), $(\bar{x})$, and $(a \to b) \wedge (c \to d)$, then the problem is W[1]-hard [41], but belongs to the class IHS-B[1] and hence admits a constant-factor approximation in polynomial time [33].

By investigating constant-factor FP-approximation, we are identifying a class of tractable constraints that unifies and generalizes the polynomial-time constant-factor approximable and fixed-parameter tractable cases. We observe that if each constraint in Γ can be expressed by a 2SAT formula (i.e., Γ is bijunctive), then we can treat the MinCSP instance as an instance of 2SAT Deletion, at the cost of a constant-factor loss in the approximation ratio. Thus the fixed-parameter tractability of 2SAT Deletion implies MinCSP has a constant-factor FP-approximation if the finite set Γ is bijunctive. If Γ is in IHS-B, then MinCSP is known to have a constant-factor approximation in polynomial time, which clearly gives another class of constant-factor FP-approximable constraints. Our main results show that probably these two classes cover all the easy cases with respect to FP-approximation (see Section 2 for the definitions involving properties of constraints).

▶ **Theorem 1.** *Let Γ be a finite Boolean constraint language.*
1. *If Γ is bijunctive or IHS-B, then* MinCSP(Γ) *has a constant-factor FP-approximation.*
2. *Otherwise, if Γ is affine, then* MinCSP(Γ) *has an FP-approximation (resp., constant-factor FP-approximation) if and only if* Nearest Codeword *has an FP-approximation (resp., constant-factor FP-approximation).*
3. *Otherwise,* MinCSP(Γ) *has no fixed-parameter approximation, unless* FPT = W[P].

Given a linear code over $GF[2]$ and a vector, the Nearest Codeword (NC) problem asks for a codeword in the code that has minimum Hamming distance to the given vector. There are various equivalent formulations of this problem: Odd Set is a variant of Hitting Set where one has to select at most k elements to hit each set exactly an odd number of times, and it is also possible to express the problem as finding a solution to a system of linear equations over $GF[2]$ that minimizes the number of unsatisfied equations. Arora et al. [2] showed that, assuming NP $\not\subseteq$ DTIME($n^{\text{polylog}\, n}$), it is not possible to approximate NC within ratio $2^{\log^{1-\epsilon} n}$ for any $\epsilon > 0$. In particular, this implies that a constant-factor polynomial-time approximation is unlikely. We give some evidence that even constant-factor FP-approximation is unlikely. First, we rule out this possibility under the assumption that the Linear PCP Conjecture (LPC) and the Exponential-Time Hypothesis (ETH) both hold.

▶ **Theorem 2.** *Assuming LPC and ETH, for any constant r,* NC *has no factor-r FP-approximation.*

Second, we connect the FP-approximability of NC with the k-Densest Subgraph problem, where the task is to find k vertices that induce the maximum number of edges.

▶ **Theorem 3.** *If* NC *has a factor-r FP-approximation for some constant r, then for every $\epsilon > 0$, there is a factor-$(1-\epsilon)$ FP-approximation for k-*Densest Subgraph.

[1] IHS-B stands for Implicative Hitting Set-Bounded, see definition in Section 2.

Thus a constant-factor FP-approximation for NC implies that k-DENSEST SUBGRAPH can be approximated arbitrarily well, which seems unlikely. Note that Theorems 2 and 3 remain valid for the other equivalent versions of NC, such as ODD SET. These theorems form the technically more involved parts of the paper.

Post's lattice is a very useful tool for classifying the complexity of Boolean CSPs (see e.g., [1, 20, 3]). A (possibly infinite) set Γ of constraints is a co-clone if it is closed under pp-definitions, that is, whenever a relation R can be expressed by relations in Γ using only equality, conjunctions, and projections, then relation R is already in Γ. Post's co-clone lattice characterizes every possible co-clone of Boolean constraints. From the complexity-theoretic point of view, Post's lattice becomes very relevant if the complexity of the CSP problem under study does not change by adding new pp-definable relations to the set Γ of allowed relations. For example, this is true for the decision version of Boolean CSP. In this case, it is sufficient to determine the complexity for each co-clone in the lattice, and a complete classification for every finite set Γ of constraints follows. For MINCSP, neither the polynomial-time solvability nor the fixed-parameter tractability of the problem is closed under pp-definitions, hence Post's lattice cannot be used directly to obtain a complexity classification. However, as observed by Khanna et al. [33] and subsequently exploited by Dalmau et al. [24, 25], the constant-factor approximability of MINCSP is closed under pp-definitions (modulo a small technicality related to equality constraints). We observe that the same holds for constant-factor FP-approximability and hence Post's lattice can be used for our purposes. Thus, the classification result amounts to identifying the maximal easy and the minimal hard co-clones.

The paper is organized as follows. Sections 2 and 3 contain preliminaries on CSPs, approximability, Post's lattice, and reductions. A more technical restatement of Theorem 1 in terms of co-clones is stated at the end of Section 3. Section 4 gives FPA algorithms, Section 5 establishes the equivalence of some CSPs with ODD SET, and Section 6 proves inapproximability results for CSPs. Section 7 proves Theorems 2 and 3, the conditional hardness results for ODD SET. Due to space restrictions, less difficult proofs appear only in the arxiv version [6].

2 Preliminaries

A subset R of $\{0,1\}^n$ is called an n-ary *Boolean relation.* If $n = 2$, relation R is *binary.* In this paper, a *constraint language* Γ is a finite collection of finitary Boolean relations. When a constraint language Γ contains only a single relation R, i.e., $\Gamma = \{R\}$, we write R instead of $\{R\}$. The decision version of CSP, restricted to finite constraint language Γ is defined as:

CSP(Γ)
Input: A pair $\langle V, \mathcal{C}\rangle$, where
- V is a set of variables,
- $\mathcal{C}$ is a multiset of constraints $\{C_1, \dots, C_q\}$, i.e., $C_i = \langle s_i, R_i\rangle$, where s_i is a tuple of variables of length n_i, and $R_i \in \Gamma$ is an n_i-ary relation.

Question: Does there exist a solution, that is, a function $\varphi : V \to \{0,1\}$ such that for each constraint $\langle s, R\rangle \in \mathcal{C}$, with $s = \langle v_1, \dots, v_n\rangle$, the tuple $\varphi(v_1), \dots, \varphi(v_n)$ belongs to R?

Note that we can alternatively look at a constraint as a Boolean function $f : \{0,1\}^n \to \{0,1\}$, where n is a non-negative integer called the arity of f. We say that f is satisfied by an assignment $s \in \{0,1\}^n$ if $f(s) = 1$. For example, if $f(x, y) = x + y \mod 2$, then the corresponding relation is $\{(0,1), (1,0)\}$; we also denote addition modulo 2 with $x \oplus y$.

We recall the definition of a few well-known classes of constraint languages. A Boolean constraint language Γ is:

- *0-valid* (*1-valid*), if each $R \in \Gamma$ contains a tuple in which all entries are 0 (1);
- *k-IHS-B+* (*k-IHS-B–*), where $k \in \mathbb{Z}^+$, if each $R \in \Gamma$ can be expressed by a conjunction of clauses of the form $\neg x$, $\neg x \vee y$, or $x_1 \vee \cdots \vee x_k$ (x, $\neg x \vee y$, $\neg x_1 \vee \cdots \vee \neg x_k$); *IHS-B+* (*IHS-B–*) stands for k-IHS-B+ (k-IHS-B–) for some k; *IHS-B* stands for IHS-B+ or IHS-B–;
- *bijunctive*, if each $R \in \Gamma$ can be expressed by a conjunction of binary clauses;
- *Horn* (*dual-Horn*), if each $R \in \Gamma$ can be expressed by a conjunction of Horn (dual-Horn) clauses, i.e., clauses that have at most one positive (negative) literal;
- *affine*, if each relation $R \in \Gamma$ can be expressed by a conjunction of relations defined by equations of the form $x_1 \oplus \cdots \oplus x_n = c$, where $c \in \{0, 1\}$;
- *self-dual* if for each relation $R \in \Gamma$, $(a_1, \ldots, a_n) \in R \Rightarrow (\neg a_1, \ldots, \neg a_n) \in R$.

MinCSP(Γ)
Input: An instance $\langle V, \mathcal{C} \rangle$ of CSP(Γ), and an integer k.
Question: Is there a deletion set $W \subseteq \mathcal{C}$ such that $|W| \leq k$, and the CSP(Γ)-instance $\langle V, \mathcal{C} \setminus W \rangle$ has a solution?

MinCSP*(Γ)
Input: An instance $\langle V, \mathcal{C} \rangle$ of CSP(Γ), a subset $\mathcal{C}^* \subseteq \mathcal{C}$ of undeletable constraints, and an integer k.
Question: Is there a deletion set $W \subseteq \mathcal{C} \setminus \mathcal{C}^*$ such that $|W| \leq k$ and the CSP(Γ)-instance $\langle V, \mathcal{C} \setminus W \rangle$ has a solution?

For every finite constraint language Γ, we consider the problem MinCSP above. For technical reasons, it will be convenient to work with a slight generalization of the problem, MinCSP*(defined above), where we can specify that certain constraints are "undeletable." For these two problems, a set of potentially more than k constraints whose removal yields a satisfiable instance is called a *feasible solution*. Note that, contrary to MinCSP for which removing all the constraints constitute a trivially feasible solution, it is possible that an instance of MinCSP* has no feasible solution. A *feasible instance* is an instance that admits at least one feasible solution. We will use two types of reductions to connect the approximability of optimization problems. The first type perfectly preserves the optimum value (or cost) of instances.

▶ **Definition 4.** An optimization problem A has a *cost-preserving reduction* to problem B if there are two polynomial-time computable functions F and G such that

1. For any feasible instance I of A, $F(I)$ is a feasible instance of B having the same optimum cost as I.
2. For any feasible instance I of A, if S' is a feasible solution for $F(I)$, then $G(I, S')$ is a feasible solution of I having cost at most the cost of $F(I)$.

The following easy lemma shows that the existence of undeletable constraints does not make the problem significantly more general. Note that, in the previous definition, if instance I has no feasible solution, then the behavior of F on I is not defined.

▶ **Lemma 5.** *There is a cost-preserving reduction from* MinCSP* *to* MinCSP.

The second type of reduction that we use is the standard notion of A-reductions [21], which preserve approximation ratios up to constant factors. We slightly deviate from the standard definition by not requiring any specific behavior of F when I has no feasible solution.

▶ **Definition 6.** A minimization problem A is *A-reducible* to problem B if there are two polynomial-time computable functions F and G and a constant α such that

1. For any feasible instance I of A, $F(I)$ is a feasible instance of B.
2. For any feasible instance I of A, and any feasible solution S' of $F(I)$, $G(I, S')$ is a feasible solution for I.
3. For any feasible instance I of A, and any $r \geq 1$, if S' is an r-approximate feasible solution for $F(I)$, then $G(I, S')$ is an (αr)-approximate feasible solution for I.

▶ **Proposition 7.** *If optimization problem A is A-reducible to optimization problem B and B admits a constant-factor FPA algorithm, then A also has a constant-factor FPA algorithm.*

3 Post's lattice, co-clone lattice, and a simple reduction

A *clone* is a set of Boolean functions that contains all projections (that is, the functions $f(a_1, \ldots, a_n) = a_k$ for $1 \leq k \leq n$) and is closed under arbitrary composition. *All* clones of Boolean functions were identified by Post [44], and he also described their inclusion structure, hence the name Post's lattice. To make use of this lattice for CSPs, Post's lattice can be transformed to another lattice whose elements are not sets of functions closed under composition, but sets of relations closed under the following notion of definability.

▶ **Definition 8.** Let Γ be a constraint language over some domain A. We say that a relation R is *pp-definable* from Γ if there exists a (primitive positive) formula $\varphi(x_1, \ldots, x_k) \equiv \exists y_1, \ldots, y_l \psi(x_1, \ldots, x_k, y_1, \ldots, y_l)$, where ψ is a conjunction of atomic formulas with relations in Γ and EQ_A (the binary relation $\{(a, a) : a \in A\}$) such that for every $(a_1, \ldots, a_k) \in A^k$ $(a_1, \ldots, a_k) \in R$ if and only if $\varphi(a_1, \ldots, a_k)$ holds. If ψ does not contain EQ_A, then we say that R is *pp-definable* from Γ *without equality*. For brevity, we often write "$\exists\wedge$-definable" instead of "pp-definable without equality". If S is a set of relations, S is *pp-definable* ($\exists\wedge$-definable) from Γ if every relation in S is *pp*-definable ($\exists\wedge$-definable) from Γ.

For a set of relations Γ, we denote by $\langle\Gamma\rangle$ the set of all relations that can be pp-defined over Γ. We refer to $\langle\Gamma\rangle$ as the *co-clone* generated by Γ. The set of all co-clones forms a lattice. To give an idea about the connection between Post's lattice and the co-clone lattice, we briefly mention the following theorem, and refer the reader to, for example, [5] for more information. Roughly speaking, the following theorem says that the co-clone lattice is essentially Post's lattice turned upside down, i.e., the inclusion between neighboring nodes are inverted.

▶ **Theorem 9** ([43], Theorem 3.1.3). *The lattices of Boolean clones and Boolean co-clones are anti-isomorphic.*

Using the above comments, it can be seen (and it is well known) that the lattice of Boolean co-clones has the structure shown in Figure 1. In the figure, if co-clone C_2 is above co-clone C_1, then $C_2 \supset C_1$. The names of the co-clones are indicated in the nodes[2], where we follow the notation of Böhler et al [5].

[2] If the name of a clone is L_3, for example, then the corresponding co-clone is $\mathrm{Inv}(\mathrm{L}_3)$ (Inv is defined, for example, in [5]), which is denoted by IL_3.

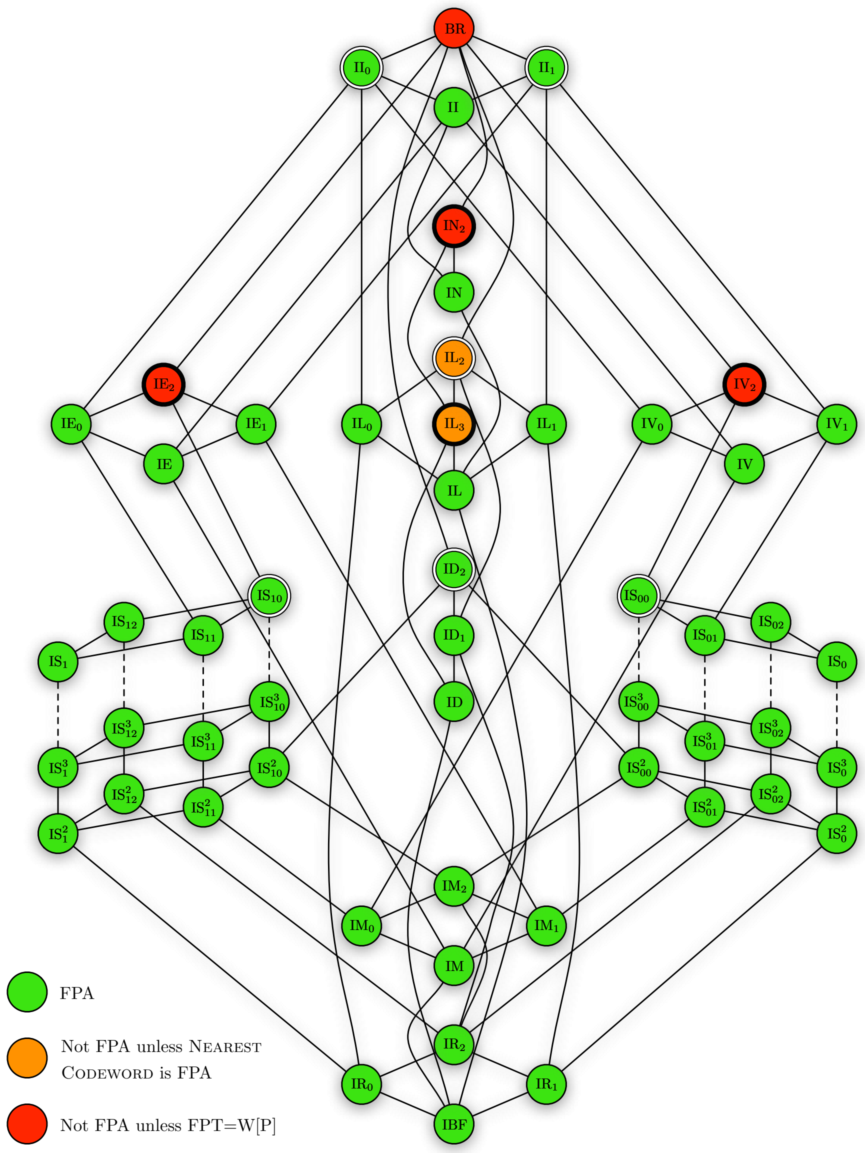


Figure 1 Classification of Boolean CSPs according to constant ratio fixed-parameter approximability. (*We thank Heribert Vollmer and Yuichi Yoshida for giving us access to their Post's lattice diagrams.*)

For a co-clone C we say that a set of relations Γ is a *base* for C if $C = \langle \Gamma \rangle$, that is, any relation in C can be pp-defined using relations in Γ. Böhler et al. give bases for all co-clones in [5], and the reader can consult this paper for details. We reproduce this list in Table 1.[3]

It is well-known that pp-definitions preserve the complexity of the decision version of CSP: if $\Gamma_2 \subseteq \langle \Gamma_1 \rangle$ for two finite languages Γ_1 and Γ_2, then there is a natural polynomial-time

[3] We note that EVEN^4 can be pp-defined using DUP^3. Therefore the base $\{\mathrm{DUP}^3, \mathrm{EVEN}^4, x \oplus y\}$ given by Böhler et al. [5] for IN_2 can be actually simplified to $\{\mathrm{DUP}^3, x \oplus y\}$.

Table 1 Bases for all Boolean co-clones. (See [5] for a complete definition of relations that appear.) The order of a co-clone is the minimum over all bases of the maximum arity of a relation in the base. The order is defined to be infinite if there is no finite base for that co-clone.

Co-clone	Order	Base	Co-clone	Order	Base
IBF	0	$\{=\}$, $\{\emptyset\}$	IS_{10}	∞	$\{\mathrm{NAND}^m \mid m \geq 2\} \cup \{x, \bar{x}, x \to y\}$
IR_0	1	$\{\bar{x}\}$	ID	2	$\{x \oplus y\}$
IR_1	1	$\{x\}$	ID_1	2	$\{x \oplus y, x\}$, every $R \in \{\{(a_1, a_2, a_3), (b_1, b_2, b_3)\} \mid \exists c \in \{1, 2\}$ such that $\sum_{i=1}^{3} a_i = \sum_{i=i}^{3} b_i = c\}$
IR_2	1	$\{x, \bar{x}\}, \{x\bar{x}\}$	ID_2	2	$\{x \oplus y, x \to y\}, \{x\bar{y}, \bar{x}yz\}$
IM	2	$\{x \to y\}$	IL	4	$\{\mathrm{EVEN}^4\}$
IM_1	2	$\{x \to y, x\}, \{x \wedge (y \to z)\}$	IL_0	3	$\{\mathrm{EVEN}^4, \bar{x}\}, \{\mathrm{EVEN}^3\}$
IM_0	2	$\{x \to y, \bar{x}\}, \{\bar{x} \wedge (y \to z)\}$	IL_1	3	$\{\mathrm{EVEN}^4, x\}, \{\mathrm{ODD}^3\}$
IM_2	2	$\{x \to y, x, \bar{x}\}, \{x \to y, \overline{x \to y}\}, \{x\bar{y} \wedge (u \to v)\}$	IL_2	3	$\{\mathrm{EVEN}^4, x, \bar{x}\}$, every $\{\mathrm{EVEN}^n, x\}$ where $n \geq 3$ is odd
IS_0^m	m	$\{\mathrm{OR}^m\}$	IL_3	4	$\{\mathrm{EVEN}^4, x \oplus y\}, \{\mathrm{ODD}^4\}$
IS_1^m	m	$\{\mathrm{NAND}^m\}$	IV	3	$\{x \vee y \vee \bar{z}\}$
IS_0	∞	$\{\mathrm{OR}^m \mid m \geq 2\}$	IV_0	3	$\{x \vee y \vee \bar{z}, \bar{x}\}$
IS_1	∞	$\{\mathrm{NAND}^m \mid m \geq 2\}$	IV_1	3	$\{x \vee y \vee \bar{z}, x\}$
IS_{02}^m	m	$\{\mathrm{OR}^m, x, \bar{x}\}$	IV_2	3	$\{x \vee y \vee \bar{z}, x, \bar{x}\}$
IS_{02}	∞	$\{\mathrm{OR}^m \mid m \geq 2\} \cup \{x, \bar{x}\}$	IE	3	$\{\bar{x} \vee \bar{y} \vee z\}$
IS_{01}^m	m	$\{\mathrm{OR}^m, x \to y\}$	IE_1	3	$\{\bar{x} \vee \bar{y} \vee z, x\}$
IS_{01}	∞	$\{\mathrm{OR}^m \mid m \geq 2\} \cup \{x \to y\}$	IE_0	3	$\{\bar{x} \vee \bar{y} \vee z, \bar{x}\}$
IS_{00}^m	m	$\{\mathrm{OR}^m, x, \bar{x}, x \to y\}$	IE_2	3	$\{\bar{x} \vee \bar{y} \vee z, x, \bar{x}\}$
IS_{00}	∞	$\{\mathrm{OR}^m \mid m \geq 2\} \cup \{x, \bar{x}, x \to y\}$	IN	3	$\{\mathrm{DUP}^3\}$
IS_{12}^m	m	$\{\mathrm{NAND}^m, x, \bar{x}\}$	IN_2	3	$\{\mathrm{DUP}^3, x \oplus y\}, \{\mathrm{NAE}^3\}$
IS_{12}	∞	$\{\mathrm{NAND}^m \mid m \geq 2\} \cup \{x, \bar{x}\}$	II	3	$\{\mathrm{EVEN}^4, x \to y\}$
IS_{11}^m	m	$\{\mathrm{NAND}^m, x \to y\}$	II_0	3	$\{\mathrm{EVEN}^4, x \to y, \bar{x}\}, \{\mathrm{DUP}^3, x \to y\}$
IS_{11}	∞	$\{\mathrm{NAND}^m \mid m \geq 2\} \cup \{x \to y\}$	II_1	3	$\{\mathrm{EVEN}^4, x \to y, x\}, \{x \vee (x \oplus z)\}$
IS_{10}^m	m	$\{\mathrm{NAND}^m, x, \bar{x}, x \to y\}$	BR	3	$\{\mathrm{EVEN}^4, x \to y, x, \bar{x}\}$, $\{$1-IN-3$\}, \{x \vee (x \oplus z)\}$

reduction from $\mathrm{CSP}(\Gamma_2)$ to $\mathrm{CSP}(\Gamma_1)$. The same is not true for MinCSP: the approximation ratio can change in the reduction. However, it has been observed that this change of the approximation ratio is at most a constant (depending on Γ_1 and Γ_2) [33, 24, 25]; we show the same here in the context of parameterized reductions.

▶ **Lemma 10.** *Let Γ be a constraint language, and R be a relation that is pp-definable over Γ without equality. Then there is an A-reduction from* MinCSP$(\Gamma \cup \{R\})$ *to* MinCSP(Γ).

By repeated applications of Lemma 10, the following corollary establishes that we need to provide approximation algorithms only for a few MinCSPs, and these algorithms can be used for other MinCSPs associated with the same co-clone.

▶ **Corollary 11.** *Let C be a co-clone and B be a base for C. If the equality relation can be $\exists\wedge$-defined from B, then for any finite $\Gamma \subseteq C$, there is an A-reduction from* MinCSP(Γ) *to* MinCSP(B).

For hardness results, we wish to argue that if a co-clone C is hard, then any constraint language Γ generating the co-clone is hard. However, there are two technical issues. First, co-clones are infinite and our constraint languages are finite. Therefore, we formulate this requirement instead by saying that a finite base B of the co-clone C is hard. Second, pp-definitions require equality relations, which may not be expressible by Γ. However, as the following theorem shows, this is an issue only if B contains relations where the coordinates are always equal (which will not be the case in our proofs). A k-ary relation R is *irredundant* if for every two different coordinates $1 \leq i < j \leq k$, R contains a tuple $(a_1, \dots, a_k)$ with $a_i \neq a_j$. A set of relations S is *irredundant* if any relation in S is irredundant.

▶ **Theorem 12** ([30, 4]). *If $S \subseteq \langle \Gamma \rangle$ and S is irredundant, then S is $\exists\wedge$-definable from Γ.*

Thus, considering an irredundant base B of co-clone C, we can formulate the following result.

▶ **Corollary 13.** *Let B be an irredundant base for some co-clone C. If Γ is a finite constraint language with $C \subseteq \langle \Gamma \rangle$, then there is an A-reduction from* MinCSP(B) *to* MinCSP(Γ).

By the following lemma, if the constraint language is self-dual, then we can assume that it also contains the constant relations.

▶ **Lemma 14.** *Let Γ be a self-dual constraint language. Assume that $x \oplus y \in \Gamma$. Then there is a cost-preserving reduction from* MinCSP($\Gamma \cup \{x, \bar{x}\}$) *to* MinCSP($\Gamma$).

The following theorem states our trichotomy classification in terms of co-clones.

▶ **Theorem 15.** *Let Γ be a finite set of Boolean relations.*

1. *If $\langle \Gamma \rangle \subseteq C$ (equivalently, if $\Gamma \subseteq C$), with $C \in \{\mathrm{II}_0, \mathrm{II}_1, \mathrm{IS}_{00}, \mathrm{IS}_{10}, \mathrm{ID}_2\}$, then* MinCSP($\Gamma$) *has a constant-factor* FPA *algorithm. (Note in these cases Γ is 0-valid, 1-valid, IHS-B+, IHS-B−, or bijunctive, respectively.)*
2. *If $\langle \Gamma \rangle \in \{\mathrm{IL}_2, \mathrm{IL}_3\}$, then* MinCSP($\Gamma$) *is equivalent to* Nearest Codeword *and to* Odd Set *under A-reductions (Note that these constraint languages are affine.)*
3. *If $C \subseteq \langle \Gamma \rangle$, where $C \in \{\mathrm{IE}_2, \mathrm{IV}_2, \mathrm{IN}_2\}$, then* MinCSP($\Gamma$) *does not have a constant-factor* FPA *algorithm unless* FPT = W[P]. *(Note that in these cases Γ can $\exists\wedge$-define either arbitrary Horn relations, or arbitrary dual Horn relations, or the relation $\mathrm{NAE}^3 = \{0,1\}^3 \setminus \{(0,0,0),(1,1,1)\}$.)*

Looking at the co-clone lattice, it is easy to see that Theorem 15 covers all cases. It is also easy to check that Theorem 1 formulated in the introduction follows from Theorem 15. Theorem 15 is proved the following way. Statement 1 is proved in Section 4 (Lemma 16, and Corollaries 18 and 21). Statement 2 is proved in Section 5 (Theorem 23). Statement 3 is proved in Sections 6 (Corollary 27 and Lemma 28).

4 CSPs with FPA algorithms

We prove the first statement of Theorem 15 by going through co-clones one by one. As every relation of a 0-valid MinCSP is always satisfied by the all 0 assignment, and every relation of a 1-valid MinCSP is always satisfied by the all 1 assignment, we have a trivial algorithm for these problems.

▶ **Lemma 16.** *If $\langle \Gamma \rangle \subseteq \mathrm{II}_0$ or $\langle \Gamma \rangle \subseteq \mathrm{II}_1$, then* MinCSP($\Gamma$) *is polynomial-time solvable.*

Consider now the co-clone ID_2. Almost 2-SAT is defined as MinCSP(Γ(2-SAT)), where $\Gamma(\text{2-SAT}) = \{x \vee y, x \vee \neg y, \neg x \vee \neg y\}$.

▶ **Theorem 17** ([45]). Almost 2-SAT *is fixed-parameter tractable.*

Since every bijunctive relation can be pp-defined by 2-SAT, the constant-factor FP-approximability of bijunctive languages easily follows from the FPT algorithm for Almost 2-SAT and from Corollary 11.

▶ **Corollary 18.** *If* $\langle\Gamma\rangle \subseteq \mathrm{ID}_2$*, then* MinCSP($\Gamma$) *has a constant-factor* FPA *algorithm.*

Proof. We check in Table 1 that $B = \{x \oplus y, x \to y\}$ is a base for the co-clone ID_2. Relations in B (and equality) can be $\exists\wedge$-defined over Γ(2-SAT), so the result follows from Corollary 11. ◀

We consider now IS_{00} and IS_{10}. We first note that if $\langle\Gamma\rangle$ is in IS_{00} or IS_{10}, then the language is k-IHS-B+ or k-IHS– for some $k \geq 2$.

▶ **Lemma 19.** *If* $\langle\Gamma\rangle \subseteq \mathrm{IS}_{00}$*, then there is an integer* $k \geq 2$ *such that* Γ *is k-IHS-B+. If* $\langle\Gamma\rangle \subseteq \mathrm{IS}_{10}$*, then there is an integer* $k \geq 2$ *such that* Γ *is k-IHS-B–.*

By Lemma 19, if $\langle\Gamma\rangle \subseteq \mathrm{IS}_{00}$, then Γ is generated by the relations $\neg x, x \to y, x_1 \vee \cdots \vee x_k$ for some $k \geq 2$. The MinCSP problem for this set of relations is known to admit a constant-factor approximation.

▶ **Theorem 20** ([19], Lemma 7.29)**.** MinCSP($\neg x, x \to y, x_1 \vee \cdots \vee x_k$) *has a* $(k+1)$*-factor approximation algorithm (and hence has a constant-factor FPA algorithm).*

Now Theorem 20 and Corollary 11 imply that there is a constant-factor FPA algorithm for MinCSP(Γ) whenever $\langle\Gamma\rangle$ is in the co-clone IS_{00} or IS_{10} (note that equality can be $\exists\wedge$-defined using $x \to y$). In fact, the resulting algorithm is a polynomial-time approximation algorithm: Theorem 20 gives a polynomial-time algorithm and this is preserved by Corollary 11.

▶ **Corollary 21.** *If* $\langle\Gamma\rangle \subseteq \mathrm{IS}_{00}$ *or* $\langle\Gamma\rangle \subseteq \mathrm{IS}_{10}$*, then* MinCSP($\Gamma$) *has a constant-factor* FPA *algorithm.*

Note that Theorem 7.25 in [19] gives a complete classification of Boolean MinCSPs with respect to constant-factor approximability. As mentioned, these MinCSPs also admit a constant-factor approximation algorithm. The reason we need Corollary 21 is to have the characterization in terms of the co-clone lattice.

5 CSPs equivalent to Odd Set

In this section we show the equivalence of several problems under A-reductions. We identify CSPs that are equivalent to the following well-known combinatorial problems. In the Nearest Codeword (NC) problem, the input is an $m \times n$ matrix A, and an m-dimensional vector b. The output is an n dimensional vector x that minimizes the Hamming distance between Ax and b. In the Odd Set problem, the input is a set-system $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ over universe U. The output is a subset $T \subseteq U$ of minimum size such that every set of $\mathcal{S}$ is hit an odd number of times by T, that is, $\forall i \in [m]$, $|S_i \cap T|$ is odd.

Even/Odd Set is the same problem as Odd Set, except that we can specify whether a set should be hit an even or odd number of times (the objective is the same as in Odd Set: find a subset of minimum size satisfying the requirements). We show that there is a parameter preserving reduction from Even/Odd Set to Odd Set.

▶ **Lemma 22.** *There is a cost-preserving reduction from* Even/Odd Set *to* Odd Set*.*

We define the relations $\mathrm{EVEN}^m = \{(a_1, \ldots, a_m) \in \{0,1\}^m : \sum_{i=1}^m a_i \text{ is even}\}$, $\mathrm{ODD}^m = \{(a_1, \ldots, a_m) \in \{0,1\}^m : \sum_{i=1}^m a_i \text{ is odd}\}$, and the languages $B_2 = \{\mathrm{EVEN}^4, x, \bar{x}\}$, $B_3 = \{\mathrm{EVEN}^4, x \oplus y\}$. Note that B_2 and B_3 are bases for the co-clones IL_2 and IL_3, respectively.

▶ **Theorem 23.** [4] *The following problems are equivalent under cost-preserving reductions:* (1) Nearest Codeword, (2) Odd Set, (3) MinCSP(B_2), *and* (4) MinCSP(B_3).

6 Hard CSPs: Horn (IV_2), dual-Horn (IE_2) and IN_2

In this section, we establish statement 3 of Theorem 15 by proving the inapproximability of MinCSP(Γ) if Γ generates one of the co-clones IE_2, IV_2, or IN_2. The inapproximability proof uses previous results on the inapproximability of circuit satisfiability problems.

A *Boolean circuit* is a directed acyclic graph, where each node with in-degree at least 2 is labeled as either an AND node or as an OR node, each node of in-degree 1 is labeled as a negation node, and each node of in-degree 0 is an input node. Furthermore, there is a node with out-degree 0 that is the output node. Given an assignment φ from the input nodes of circuit C to $\{0, 1\}$, we say that assignment φ satisfies C if the value of the output node (computed in the obvious way) is 1. The *weight* of an assignment is the number of input nodes with value 1. Circuit C is k-satisfiable if there is a weight-k assignment satisfying C. A circuit is *monotone* if it contains no negation gates. The problem Monotone Circuit Satisfiability (MCS) takes as input a monotone circuit C and an integer k, and the task is to decide if there is a satisfying assignment of weight at most k. The following theorem is a restatement of a result of Marx [40]. We use this to show that Horn-CSPs are hard.

▶ **Theorem 24** ([40]). Monotone Circuit Satisfiability *does not have an* FPA *algorithm, unless* FPT $=$ W[P].

▶ **Corollary 25.** Monotone Circuit Satisfiability, *where circuits are restricted to have gates of in-degree at most* 2, *does not have an FPA algorithm, unless* FPT $=$ W[P].

We use Corollary 25 to establish the inapproximability of Horn-SAT and dual-Horn-SAT, assuming that FPT $\neq$ W[P]. Using the co-clone lattice, this will show hardness of approximability of MinCSP(Γ) if $\langle\Gamma\rangle \in \{IV_2, IE_2\}$.

▶ **Lemma 26.** *If there is an* FPA *algorithm for* MinCSP($\{x \vee y \vee \bar{z}, x, \bar{x}\}$) *or* MinCSP($\{\bar{x} \vee \bar{y} \vee z, x, \bar{x}\}$) *with constant approximation ratio, then* FPT $=$ W[P].

Proof. We prove that there is a parameter preserving polynomial-time reduction from Monotone Circuit Satisfiability to MinCSP*($\{x \vee y \vee \bar{z}, x, \bar{x}\}$). This is sufficient by Corollary 25. Let C be the MCS instance. We produce an instance I of MinCSP* as follows. We think of inputs of C as gates, and we refer to these as "input gates". This will simplify the discussion. For each gate of C, we introduce a new variable into I, and we let f denote the natural bijection from the gates and inputs of C to the variables of the instance I.

We add constraints to simulate each AND gate of C as follows. Observe first that the implication relation $x \to y$ can be expressed as $y \vee y \vee \bar{x}$. For each AND gate $G_\wedge$ such that G_1 and G_2 are the gates feeding into $G_\wedge$ (note that G_1 and G_2 are allowed to be input gates), we add two constraints to I as follows. Let $y = f(G_\wedge)$, $x_1 = f(G_1)$, and $x_2 = f(G_2)$. We place the constraints $y \to x_1, y \to x_2$ into I. We observe that the only way variable y could take on value 1 is if both x_1 and x_2 are assigned 1. (In this case, note that y could also be assigned 0 but that will be easy to fix.)

Similarly, we add constraints to simulate each OR gate of C as follows. For each OR gate $G_\vee$ such that G_1 and G_2 are the gates feeding into $G_\vee$, we add a constraint to I, we add

[4] Note that Lemma 1 in [22] can be adapted to obtain the reduction from Odd Set to MinCSP(B_2).

the constraint $x_1 \vee x_2 \vee \bar{y}$ to I, where $y = f(G_\vee)$, $x_1 = f(G_1)$, and $x_2 = f(G_2)$. Note that if both x_1 and x_2 are 0, than y is forced to have value 0. (Otherwise y can take on either value 0 or 1, but again, this difference between an OR gate and our gadget will be easy to handle.)

In addition, we add a constraint $x_o = 1$, where x_o is the variable such that $x_o = f(G)$, where G is the output gate. We define all constraints that appeared until now to be undeletable, so that they cannot be removed in solution of the MinCSP* instance. To finish the construction, for each variable x such that $x = f(G)$ where G is an input gate, we add a constraint $x = 0$ to I. We call these constraints *input constraints*. Note that only input constraints can be removed.

If there is a satisfying assignment φ_C of C (from gates of C to $\{0, 1\}$) of weight k, then we remove the input constraints $x = 0$ of I such that $\varphi_C(G) = 1$, where $f(G) = x$. Clearly, the map $\varphi_C \circ f^{-1}$ is a satisfying assignment for I, where we needed k deletions.

For the other direction, assume that we have a satisfying assignment φ_I for I after removing some k input constraints (note that if any other constraints are removed, we can simply ignore those deletions). We repeatedly change φ_I as long as either of the following conditions apply. If x_1, x_2 and y are such that $f^{-1}(x_1)$ and $f^{-1}(x_2)$ are gates feeding into gate $f^{-1}(y)$ where $f^{-1}(y)$ is an AND gate, and $\varphi_I(x_1) = 1, \varphi_I(x_2) = 1, \varphi_I(y) = 0$, then we change $\varphi_I(y)$ to 1. Similarly, if $f^{-1}(y)$ is an OR gate, $1 \in \{\varphi_I(x_1), \varphi_I(x_2)\}, \varphi_I(y) = 0$, then we change $\varphi_I(y)$ to 1. It follows form the definition of the constraints we introduced for AND and OR gates that once we finished modifying φ_I, the resulting assignment φ'_I is still a satisfying assignment. Now it follows that $\varphi'_I \circ f$ is a weight k satisfying assignment for C.

To show the inapproximability of MinCSP$(\{\bar{x} \vee \bar{y} \vee z, x, \bar{x}\})$, we note that there is a parameter preserving bijection between instances of MinCSP$(\{\bar{x} \vee \bar{y} \vee z, x, \bar{x}\})$ and MinCSP$(\{x \vee y \vee \bar{z}, x, \bar{x}\})$: given an instance I of either problem, we obtain an equivalent instance of the other problem by replacing every literal ℓ with $\neg\ell$. Satisfying assignments are converted by replacing 0-s with 1-s and vice versa. ◀

As $\{x \vee y \vee \bar{z}, x, \bar{x}\}$ (resp., $\{\bar{x} \vee \bar{y} \vee z, x, \bar{x}\}$) is an irredundant base of IV_2 (resp., IE_2), Corollary 13 implies hardness if $\langle\Gamma\rangle$ contains IV_2 or IE_2.

▶ **Corollary 27.** *If Γ is a (finite) constraint language with $\mathrm{IV}_2 \subseteq \langle\Gamma\rangle$ or $\mathrm{IE}_2 \subseteq \langle\Gamma\rangle$, then* MinCSP$(\Gamma)$ *is not FP-approximable, unless* FPT = W[P].

▶ **Lemma 28.** *If Γ is a (finite) constraint language with $\mathrm{IN}_2 \subseteq \langle\Gamma\rangle$ then* MinCSP(Γ) *is not FP-approximable, unless* P = NP.

7 Odd Set is probably hard

We provide evidence that problems equivalent to NC and Odd Set (in particular, problems in Theorem 15(2)) are hard, i.e., they are unlikely to have a constant-factor FPA algorithm.

In the k-Densest Subgraph problem, we are given a graph $G = (V, E)$ and an integer k; the task is to find a set S of k vertices that maximizes the number of edges in the induced subgraph $G[S]$. Note that an exact algorithm for k-Densest Subgraph would imply an exact algorithm for Clique. Due to its similarity to Clique, it is reasonable to assume that k-Densest Subgraph is even hard to approximate. We formulate the following specific hardness assumption.

▶ **Assumption 29.** *There is an $\varepsilon > 0$ such that for any function f, one cannot approximate k-*Densest Subgraph *within ratio $1 - \varepsilon$ in time $f(k) \cdot n^{O(1)}$.*

It will be more convenient to work with a slightly different version of k-Densest Subgraph. In the Multicolored k-Densest Subgraph problem, we are given a graph $G = (V, E)$ whose vertex-set V is partitioned into k classes $C_1, \ldots, C_k$, and the goal is to find a set $S = \{v_1, \ldots, v_k\}$ of k vertices satisfying $v_i \in C_i$ for each $i \in [k]$, and maximizing the number of edges in the induced subgraph $G[S]$. We argue in the arxiv version that Assumption 29 implies Assumption 30 [6].

▶ **Assumption 30.** *There is an $\varepsilon > 0$ such that for any function f, one cannot approximate* Multicolored k-Densest Subgraph *within ratio $1 - \varepsilon$ in time $f(k) \cdot n^{O(1)}$.*

Odd Set has the so-called *self-improvement* property. Informally, a polynomial time (resp. fixed-parameter time) approximation within some ratio r can be turned into a polynomial time (resp. fixed-parameter time) approximation within some ratio close to $\sqrt{r}$.

▶ **Lemma 31.** *If there is an r-approximation for* Odd Set *running in time $f(n, m, k)$ where n is the size of the universe, m the number of sets, and k the size of an optimal solution, then for any $\varepsilon > 0$, there is a $(1+\varepsilon)\sqrt{r}$-approximation running in time* $\max(f(1+n+n^2, 1+m+nm, 1+k+k^2), O(n^{1+\frac{1}{\varepsilon}}m))$.

Proof. The following reduction is inspired by the one showing the self-improvement property of NC [2]. Let $\mathcal{S} = \{S_1, \ldots, S_m\}$ be any instance over universe $U = \{x_1, \ldots, x_n\}$. Let $\varepsilon > 0$ be any real positive value and k be the size of an optimal solution. We can assume that $k \geqslant \frac{1}{\varepsilon}$ since one can find an optimal solution by exhaustive search in time $O(n^{1+\frac{1}{\varepsilon}}m)$. We build the set-system $\mathcal{S}' = \mathcal{S} \cup \bigcup_{i \in [n], j \in [m]} S^i_j \cup \{\{e\}\}$ over universe $U' = U \cup \bigcup_{i,h \in [n]} \{x^i_h\} \cup \{e\}$ such that $S^i_j = \{e, x_i\} \cup \{x^i_h \mid x_h \in S_j\}$. Note that the size of the new instance is squared. We show that there is a solution of size at most k to instance $\mathcal{S}$ if and only if there is a solution of size at most $1 + k + k^2$ to instance $\mathcal{S}'$.

If T is a solution to $\mathcal{S}$, then $T' = \{e\} \cup T \cup \{x^i_h \mid x_i, x_h \in T\}$ is a solution to $\mathcal{S}'$. Indeed, sets in $\mathcal{S} \cup \{\{e\}\}$ are obviously hit an odd number of times. And, for any $i \in [n]$ and $j \in [m]$, set S^i_j is hit exactly once (by e) if $x_i \notin T$, and is hit by e, x_i, plus as many elements as S_j is hit by T; so again an odd number of times. Finally, $|T'| = 1 + |T| + |T|^2$.

Conversely, any solution to $\mathcal{S}'$ should contain element e (to hit $\{e\}$), and should intersect U in a subset T hitting an odd number of times each set S_i ($\forall i \in [m]$). Then, for each $x_i \in T$, each set S^i_j with $j \in [m]$ is hit exactly twice by e and x_i. Thus, one has to select a subset of $\{x^i_1, \ldots, x^i_n\}$ to hit each set of the family $\{S^i_1, \ldots, S^i_m\}$ an odd number of times. Again, this needs as many elements as a solution to $\mathcal{S}$ needs. So, if there is a solution to $\mathcal{S}'$ of size at most $1 + k + k^2$, then there is a solution to $\mathcal{S}$ of size at most k. In fact, we will only use the weaker property that if there is a solution to $\mathcal{S}'$ of size at most k, then there is a solution to $\mathcal{S}$ of size at most $\sqrt{k}$.

Now, assuming there is an r-approximation for Odd Set running in time $f(n, m, k)$, we run that algorithm on the instance $\mathcal{S}'$ produced from $\mathcal{S}$. This takes time $f(1+n+n^2, 1+m+nm, 1+k+k^2)$ and produces a solution of size $r(1+k+k^2)$. From that solution, we can extract a solution T to $\mathcal{S}$ by taking its intersection with U. And T has size smaller than $\sqrt{r(1+k+k^2)} \leqslant \sqrt{r}(k+1) = (1+\frac{1}{k})\sqrt{r}k \leqslant (1+\varepsilon)\sqrt{r}k$. ◀

Repeated application of the self-improvement in Lemma 31 shows that any constant-ratio approximation implies the existence of $(1+\varepsilon)$-approximation for arbitrary small $\varepsilon > 0$.

▶ **Corollary 32.** *If* Odd Set *admits an FPA algorithm with some ratio r, then, for any $\varepsilon > 0$, it also admits an FPA algorithm with ratio $1 + \varepsilon$.*

Now we show that an approximation for Odd Set with ratio $1 + \frac{\varepsilon}{3}$ implies the existence of a $(1 - \epsilon)$-approximation for k-Densest Subgraph. In light of Corollary 32, this means that any constant-factor approximation for Odd Set would violate Assumption 29.

▶ **Theorem 33.** *For any ratio r,* Odd Set *does not have an FPA algorithm with ratio r, unless Assumption 29 fails.*

Proof. Let $\varepsilon > 0$ be such that k-Densest Subgraph and therefore Multicolored k-Densest Subgraph do not admit a fixed-parameter $(1 - \varepsilon)$-approximation. We show that an FPA algorithm with ratio $1 + \frac{\varepsilon}{3}$ for Odd Set would contradict Assumption 29, and we conclude with Corollary 32. Let $G = (V = C_1 \uplus \ldots \uplus C_k, E)$ be an instance of Multicolored k-Densest Subgraph, and let X be an optimal solution inducing m edges. For any $\{i, j\} \in \binom{[k]}{2}$, we let $E_{\{i,j\}}$ be the set of edges between C_i and C_j.

We build $2^{\binom{k}{2}}$ instances of Odd Set: one for each subset of $\binom{[k]}{2}$. One such subset is $\mathcal{P} := \{\{i, j\} \mid E_{\{i,j\}} \cap E(X) \neq \emptyset\}$. In words, $\mathcal{P}$ is a correct guess of which $E_{\{i,j\}}$ are inhabited by the edges induced by the optimal solution X. Let $\mathcal{V}$ be the subset of indices $i \in [k]$ such that i appears in at least one pair of $\mathcal{P}$, and let $k' = |\mathcal{V}|$. Informally, $\mathcal{V}$ corresponds to the color classes of the vertices which are not isolated in the subgraph induced by X.

The universe U consists of an element x_v per vertex v of C_i such that $i \in \mathcal{V}$ and an element x_e per edge e in $E_{\{i,j\}}$ such that $\{i, j\} \in \mathcal{P}$. For any vertex $u \in C_i$ and any $j \in [k]$ such that $\{i, j\} \in \mathcal{P}$, we set $S_{u,j} = \{x_v \mid v \in C_i \text{ and } v \neq u\} \cup \{x_{vw} \mid vw \in E_{\{i,j\}} \text{ and } u \in \{v, w\}\}$, and for each $i \in \mathcal{V}$, $S_i = \{x_v \mid v \in C_i\}$. The set-system is $\mathcal{I} = (U, \mathcal{S} = \bigcup_{u \in C_i, \{i,j\} \in \mathcal{P}} S_{u,j} \uplus \bigcup_{i \in \mathcal{V}} S_i)$.

First, we show that the instance of Odd Set built for subset $\mathcal{P}$ admits a solution of size $k' + m$. Let $X' = \{a_1, \ldots, a_{k'}\} \subseteq X$ be the k' vertices which are not isolated in $G[X]$. We claim that $Z = \{x_{a_i}\}_{i=1}^{k'} \cup \{x_e\}_{e \in E(X')}$ is an odd set of $\mathcal{I}$. Each S_i with $i \in \mathcal{V}$ is hit by exactly one element of Z since no two a_p's can come from the same color class. Each $S_{u,j}$ with $u \in C_i$, $\{i, j\} \in \mathcal{P}$, and $u \notin X'$ is hit exactly once by x_{a_p} where the color class of a_p is C_i. Each $S_{u,j}$ with $u = a_p \in C_i \cap X'$, $\{i, j\} \in \mathcal{P}$ is hit exactly once by $x_{a_p a_q}$ where the color class of a_q is C_j. Finally, Z has the desired size $|X'| + |E(X')| = k' + |E(X)| = k' + m$.

Since we have established that $\mathcal{I}$ has an odd set of size $k' + m$, our supposed $1 + \frac{\varepsilon}{3}$-approximation would return a solution Z of at most $(1 + \frac{\varepsilon}{3})(k' + m)$ elements. We now show how to obtain a good approximation for Multicolored k-Densest Subgraph from such a solution to Odd Set. By construction, for each $i \in \mathcal{V}$, the set S_i should be hit an odd number of times, that is $|Z \cup S_i|$ is odd. In particular, $Z \cup S_i$ is non-empty. So, we can build a set $\{x_{u_i} | i \in \mathcal{V}\}$ where x_{u_i} is an arbitrary element of $Z \cup S_i$.

Let $\mathcal{E} = \{S_{u_i,j} \mid \{i, j\} \in \mathcal{P}\}$. Each of the $2|\mathcal{P}|$ sets of $\mathcal{E}$ (note that if, say, $\{1, 2\} \in \mathcal{P}$, then both $S_{u_1,2}$ and $S_{u_2,1}$ become a member of $\mathcal{E}$) are hit an even number of times by $Z \cap \bigcup_{i \in \mathcal{V}} S_i$. Indeed, $|(Z \cap \bigcup_{i \in \mathcal{V}} S_i) \cap S_{u_i,j}| = |Z \cap S_i \setminus \{x_{u_i}\}| = |Z \cap S_i| - 1$ which is even. We observe that each $S_{u_i,j} \in \mathcal{E}$ intersects with only one other set of $\mathcal{E}$, namely, $S_{u_j,i}$. So, we need at least $|\mathcal{P}|$ elements to hit the sets in $\mathcal{E}$. If there is an edge between u_i and u_j, both $S_{u_i,j}$ and $S_{u_j,i}$ can be hit at the same time by including element $x_{u_i u_j}$ into the solution. Otherwise $S_{u_i,j}$ and $S_{u_j,i}$ are disjoint and at least two elements are necessary to hit them. As there are at least $\frac{k'}{2}$ edges on k' non isolated vertices, we have $y \geq \frac{k'}{2}$. The set $Z \setminus (S_1 \cup \cdots \cup S_k)$ contains at most $|Z| - k \leq (1 + \frac{\varepsilon}{3})m + \frac{\varepsilon}{3}k' \leq (1 + \frac{\varepsilon}{3})m + \frac{2\varepsilon}{3}m = (1 + \varepsilon)m$ elements and these elements hit every set in $\mathcal{E}$. Thus, it can be true only for at most εm of the m pairs in $\mathcal{P}$ that the two sets $S_{u_i,j}, S_{u_j,i} \in \mathcal{E}$ cannot be hit by a single element of Z. Equivalently, it is true for at least $(1 - \varepsilon)m$ of the m pairs in $\mathcal{P}$ that the two sets $S_{u_i,j}, S_{u_j,i} \in \mathcal{E}$ are hit by a single element of Z. As mentioned previously, that element can only be $x_{u_i u_j}$. The fact that such an element actually exists means that there is an edge between u_i and u_j. Therefore, $\{u_i\}_{i \in \mathcal{V}}$

induces at least $(1-\varepsilon)m$ edges. It follows that Z is an $(1-\varepsilon)$-approximate solution for the instance of MULTICOLORED k-DENSEST SUBGRAPH; a contradiction to Assumption 29. ◀

▶ **Assumption 34** (Linear PCP Conjecture). *There exist constants $0 < \alpha < 1$, $A, B > 0$, such that* MAX 3-SAT *on n variables can be decided with completeness 1 and error α by a verifier using $\log n + A$ random bits and reading B bits of the proof.*

LPC is probably better thought of as an open problem rather than a conjecture. In previous work, LPC has almost always proved to be a necessary hypothesis in showing that a specific problem cannot admit an FPA algorithm [7]. If LPC turns out to be true, the consequence for approximation is that there is a linear reduction introducing a constant gap from 3-SAT to MAX 3-SAT. Thus, if we combine this fact with the sparsification lemma of Impagliazzo et al. [31], we may observe the following result:

▶ **Lemma 35** (Lemma 2, [7]). *Under LPC and ETH, there are two constants $r < 1$ and $\delta > 0$ such that one cannot distinguish satisfiable instances of* MAX 3-SAT *with m clauses from instances where at most rm clauses are satisfied in time $2^{\delta m}$.*

The previous result was in fact stated slightly more generally allowing a weaker form of LPC where the completeness is not 1 but $1-\varepsilon$. We re-stated the lemma this way since we will need perfect completeness. The state-of-the-art PCP concerning the inapproximability of MAX 3-SAT only implies the following:

▶ **Theorem 36** ([42]). *Under ETH, one cannot distinguish satisfiable instances of* MAX 3-SAT *from instances where at most $(\frac{7}{8}+o(1))m$ clauses are satisfied in time $2^{m^{1-o(1)}}$.*

Now, we are set for the following result:

▶ **Theorem 37.** *Under LPC and ETH, for any ratio r,* ODD SET *does not have an FPA algorithm with ratio r.*

Proof. Again, the idea is to assume an FPA algorithm with ratio $1+\varepsilon$ for ODD SET (with parameter k), and show that it would imply a too good approximation for MAX 3-SAT in subexponential time, therefore contradicting Lemma 35, and then conclude with Lemma 31.

Let $\phi = \bigwedge_{1 \leqslant i \leqslant m} C_i$ be any instance of MAX 3-SAT, where the C_is are 3-clauses over the set of n variables V. We partition the clauses arbitrarily into k sets $A_1, A_2, \dots, A_k$ of size roughly $\frac{m}{k}$. We denote by V_i the set of all the variables appearing in at least one clause of A_i; each V_i has size at most $\frac{3m}{k}$. Of course, while the A_i's are a partition of the clauses, the V_i's can intersect with each other. We build an instance $\mathcal{I} = (U, \mathcal{S})$ of ODD SET the following way. For each $i \in [k]$, set U_i contains one element $x(\mathcal{A}, i)$ per assignment $\mathcal{A}$ of V_i that satisfies all the clauses inside A_i. The universe U is $\bigcup_i U_i$. For each $i \neq j \in [k]$, for each variable $y \in V_i \cap V_j$, we set $S_{y,i,j} = \{x(\mathcal{A}, i) \mid y \text{ is set to true by } \mathcal{A}\} \cup \{x(\mathcal{A}, j) \mid y \text{ is set to false by } \mathcal{A}\}$. Observe that $S_{y,i,j}$ and $S_{y,j,i}$ are two different sets. Finally, $\mathcal{S} = \bigcup_{i \in [k]} \{U_i\} \cup \bigcup_{i \neq j \in [k], y \in V_i \cap V_j} S_{y,i,j}$.

If ϕ is satisfiable, we fix a (global) satisfying assignment $\mathcal{A}_g$. We claim that $S = \{x(\mathcal{A}, i) \mid \mathcal{A} \text{ agrees with } \mathcal{A}_g \text{ in the entire } V_i\}$ is a solution of size k to the ODD SET instance. Set S is of size k since for each $i \in [k]$ exactly one element $x(\mathcal{A}, i)$ can be such $\mathcal{A}$ agrees with $\mathcal{A}_g$. This also shows that each set U_i is hit exactly once by S. Finally, for each $i \neq j \in [k]$ and for each variable $y \in V_i \cap V_j$, sets $\{x(\mathcal{A}, i) \mid y \text{ is set to true by } \mathcal{A}\}$ and $\{x(\mathcal{A}, j) \mid y \text{ is set to false by } \mathcal{A}\}$ can be hit at most once. Besides, $\{x(\mathcal{A}, i) \mid y \text{ is set to true by } \mathcal{A}\}$ is hit exactly once by S if and only if $\{x(\mathcal{A}, j) \mid y \text{ is set to false by } \mathcal{A}\}$ is not hit by S, since the partial assignments mapped to the elements in S necessarily agree. Therefore, $S_{y,i,j}$ is hit exactly once by S, and S is a solution.

Now, we assume that Odd Set admits an FPA algorithm with ratio $1+\varepsilon$ for a small ε that we will fix later. If the solution S returned by this algorithm on instance $\mathcal{I}$ is of size greater than $(1+\varepsilon)k$, then we know that an optimal odd set has more than k elements, so we know that ϕ is not satisfiable. So, we can assume that $|S| \leqslant (1+\varepsilon)k$. Each U_i has to be hit at least once and U_is are pairwise disjoint, so we can arbitrarily decompose S into $P \uplus R$, where P is of size k and hits each U_i exactly once, and therefore $|R| \leqslant \varepsilon k$. Thus, at least $(1-\varepsilon)k$ sets U_is are hit exactly once by S. We denote by $\mathcal{U}$ the set of such sets U_i. Let $\mathcal{A}_g$ be the assignment of V agreeing on each assignment $\mathcal{A}$ of V_i such that $x(\mathcal{A}, i)$ is the only element hitting $U_i \in \mathcal{U}$ (and setting the potential remaining variable arbitrarily). Assignment $\mathcal{A}_g$ is well defined since if $x(\mathcal{A}, i)$ is the only element hitting $U_i \in \mathcal{U}$ and $x(\mathcal{A}', j)$ is the only element hitting $U_j \in \mathcal{U}$, and assignments $\mathcal{A}$ and $\mathcal{A}'$ disagree on a variable y, then $S_{y,i,j}$ would be hit an even number of times (0 or 2). By construction, $\mathcal{A}_g$ satisfies all the clauses in the A_is such that $U_i \in \mathcal{U}$, that is at least $(1-\varepsilon)k \times \frac{m}{k} = (1-\varepsilon)m$ clauses. Let r and δ be two constants satisfying Lemma 35. If we choose $\varepsilon = \frac{1-r}{2}$, this number of clauses exceed rm, so we would know that the instance is satisfiable.

Say, the running time of the FPA algorithm is $f(k)(|U| + |\mathcal{S}|)^c$ for some constant c. We may observe that $|U| \leqslant k2^{\frac{3m}{k}}$ and $|\mathcal{S}| \leqslant k + 2\binom{k}{2}n$. Thus, the running time is $g(k)n^c 2^{\frac{3mc}{k}}$. Setting $k = \frac{6c}{\delta}$, this running time would be better than $2^{\delta m}$, contradicting LPC or ETH. ◀

References

1 Eric Allender, Michael Bauland, Neil Immerman, Henning Schnoor, and Heribert Vollmer. The complexity of satisfiability problems: Refining Schaefer's theorem. *J. Comput. Syst. Sci.*, 75(4):245–254, 2009.

2 Sanjeev Arora, László Babai, Jacques Stern, and Z. Sweedyk. The hardness of approximate optima in lattices, codes, and systems of linear equations. *J. Comput. Syst. Sci.*, 54(2):317–331, 1997. doi:10.1006/jcss.1997.1472.

3 Arnab Bhattacharyya and Yuichi Yoshida. An algebraic characterization of testable boolean CSPs. In *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, pages 123–134, 2013.

4 V. G. Bodnarchuk, L. A. Kaluzhnin, V. N. Kotov, and B. A. Romov. Galois theory for Post algebras. *Kibernetika*, 5(3):1–10, 1969.

5 Elmar Böhler, Steffen Reith, Henning Schnoor, and Heribert Vollmer. Bases for boolean co-clones. *Inf. Process. Lett.*, 96(2):59–66, 2005.

6 Édouard Bonnet, László Egri, and Dániel Marx. Fixed-parameter approximability of boolean minCSPs. *CoRR*, abs/1601.04935, 2016. URL: http://arxiv.org/abs/1601.04935.

7 Édouard Bonnet, Bruno Escoffier, Eun Jung Kim, and Vangelis Th. Paschos. On subexponential and fpt-time inapproximability. *Algorithmica*, 71(3):541–565, 2015. doi:10.1007/s00453-014-9889-1.

8 Andrei A. Bulatov. A dichotomy theorem for constraint satisfaction problems on a 3-element set. *J. ACM*, 53(1):66–120, 2006. doi:10.1145/1120582.1120584.

9 Andrei A. Bulatov. Complexity of conservative constraint satisfaction problems. *ACM Trans. Comput. Log.*, 12(4):24, 2011. doi:10.1145/1970398.1970400.

10 Andrei A. Bulatov. The complexity of the counting constraint satisfaction problem. *J. ACM*, 60(5):34, 2013. doi:10.1145/2528400.

11 Andrei A. Bulatov, Martin E. Dyer, Leslie Ann Goldberg, Markus Jalsenius, Mark Jerrum, and David Richerby. The complexity of weighted and unweighted #CSP. *J. Comput. Syst. Sci.*, 78(2):681–688, 2012. doi:10.1016/j.jcss.2011.12.002.

12 Andrei A. Bulatov and Dániel Marx. The complexity of global cardinality constraints. *Logical Methods in Computer Science*, 6(4), 2010. doi:10.2168/LMCS-6(4:4)2010.

13 Andrei A. Bulatov and Dániel Marx. Constraint satisfaction parameterized by solution size. *SIAM J. Comput.*, 43(2):573–616, 2014. doi:10.1137/120882160.

14 Liming Cai and Xiuzhen Huang. Fixed-parameter approximation: Conceptual framework and approximability results. *Algorithmica*, 57(2):398–412, 2010. doi:10.1007/s00453-008-9223-x.

15 Shuchi Chawla, Robert Krauthgamer, Ravi Kumar, Yuval Rabani, and D. Sivakumar. On the hardness of approximating multicut and sparsest-cut. *Computational Complexity*, 15(2):94–114, 2006. doi:10.1007/s00037-006-0210-9.

16 Yijia Chen, Martin Grohe, and Magdalena Grüber. On parameterized approximability. In *Parameterized and Exact Computation, Second International Workshop, IWPEC 2006, Zürich, Switzerland, September 13-15, 2006, Proceedings*, pages 109–120, 2006.

17 Yijia Chen, Martin Grohe, and Magdalena Grüber. On parameterized approximability. *Electronic Colloquium on Computational Complexity (ECCC)*, 14(106), 2007. URL: http://eccc.hpi-web.de/eccc-reports/2007/TR07-106/index.html.

18 Rajesh Hemant Chitnis, MohammadTaghi Hajiaghayi, and Guy Kortsarz. Fixed-parameter and approximation algorithms: A new look. In *Parameterized and Exact Computation - 8th International Symposium, IPEC 2013, Sophia Antipolis, France, September 4-6, 2013, Revised Selected Papers*, pages 110–122, 2013. doi:10.1007/978-3-319-03898-8_11.

19 Nadia Creignou, Sanjeev Khanna, and Madhu Sudan. *Complexity Classifications of Boolean Constraint Satisfaction Problems.* Society for Industrial and Applied Mathematics, 2001.

20 Nadia Creignou and Heribert Vollmer. Parameterized complexity of weighted satisfiability problems: Decision, enumeration, counting. *Fundam. Inform.*, 136(4):297–316, 2015.

21 Pierluigi Crescenzi and Alessandro Panconesi. Completeness in approximation classes. *Inf. Comput.*, 93(2):241–262, 1991. doi:10.1016/0890-5401(91)90025-W.

22 Robert Crowston, Gregory Gutin, Mark Jones, and Anders Yeo. Parameterized complexity of satisfying almost all linear equations over $\mathbb{F}_2$. *Theory Comput. Syst.*, 52(4):719–728, 2013. doi:10.1007/s00224-012-9415-2.

23 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.

24 Víctor Dalmau and Andrei A. Krokhin. Robust satisfiability for csps: Hardness and algorithmic results. *TOCT*, 5(4):15, 2013. doi:10.1145/2540090.

25 Víctor Dalmau, Andrei A. Krokhin, and Rajsekar Manokaran. Towards a characterization of constant-factor approximable min csps. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 847–857, 2015. doi:10.1137/1.9781611973730.58.

26 Vladimir G. Deineko, Peter Jonsson, Mikael Klasson, and Andrei A. Krokhin. The approximability of MAXCSP with fixed-value constraints. *J. ACM*, 55(4), 2008. doi:10.1145/1391289.1391290.

27 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity.* Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

28 Rodney G. Downey, Michael R. Fellows, Catherine McCartin, and Frances Rosamond. Parameterized approximation of dominating set problems. *Inform. Process. Lett.*, 109(1):68–70, 2008. doi:10.1016/j.ipl.2008.09.017.

29 J. Flum and M. Grohe. *Parameterized Complexity Theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer, Berlin, 2006.

30 D. Geiger. Closed systems of functions and predicates. *Pacific Journal of Mathematics*, 27:95–100, 1968.

31 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. `doi:10.1006/jcss.2001.1774`.

32 Peter Jonsson, Mikael Klasson, and Andrei A. Krokhin. The approximability of three-valued MAXCSP. *SIAM J. Comput.*, 35(6):1329–1349, 2006. `doi:10.1137/S009753970444644X`.

33 Sanjeev Khanna, Madhu Sudan, Luca Trevisan, and David P. Williamson. The approximability of constraint satisfaction problems. *SIAM J. Comput.*, 30(6):1863–1920, 2000. `doi:10.1137/S0097539799349948`.

34 Vladimir Kolmogorov and Stanislav Zivny. The complexity of conservative valued CSPs. *J. ACM*, 60(2):10, 2013. `doi:10.1145/2450142.2450146`.

35 Stefan Kratsch, Dániel Marx, and Magnus Wahlström. Parameterized complexity and kernelizability of MaxOnes and ExactOnes problems. In *Mathematical Foundations of Computer Science 2010, 35th International Symposium, MFCS 2010, Brno, Czech Republic, August 23-27, 2010. Proceedings*, pages 489–500, 2010. `doi:10.1007/978-3-642-15155-2_43`.

36 Stefan Kratsch and Magnus Wahlström. Preprocessing of MinOnes problems: A dichotomy. In *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part I*, pages 653–665, 2010. `doi:10.1007/978-3-642-14165-2_55`.

37 Daniel Lokshtanov, N. S. Narayanaswamy, Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. Faster parameterized algorithms using linear programming. *ACM Transactions on Algorithms*, 11(2):15:1–15:31, 2014. `doi:10.1145/2566616`.

38 Dániel Marx. Parameterized complexity of constraint satisfaction problems. *Computational Complexity*, 14(2):153–183, 2005. `doi:10.1007/s00037-005-0195-9`.

39 Dániel Marx. Parameterized complexity and approximation algorithms. *Comput. J.*, 51(1):60–78, 2008.

40 Dániel Marx. Completely inapproximable monotone and antimonotone parameterized problems. *J. Comput. Syst. Sci.*, 79(1):144–151, 2013.

41 Dániel Marx and Igor Razgon. Constant ratio fixed-parameter approximation of the edge multicut problem. *Inf. Process. Lett.*, 109(20):1161–1166, 2009. `doi:10.1016/j.ipl.2009.07.016`.

42 Dana Moshkovitz and Ran Raz. Two-query PCP with subconstant error. *J. ACM*, 57(5), 2010. `doi:10.1145/1754399.1754402`.

43 R. Pöschel and Kalužnin. *Funktionen- und Relationenalgebren.* Deutscher Verlag der Wissenschaften, Berlin, 1979.

44 Emil L. Post. *On the Two-Valued Iterative Systems of Mathematical Logic.* Princeton University Press, 1941.

45 Igor Razgon and Barry O'Sullivan. Almost 2-SAT is fixed-parameter tractable. *J. Comput. Syst. Sci.*, 75(8):435–450, 2009.

46 Thomas J. Schaefer. The complexity of satisfiability problems. In *Conference Record of the Tenth Annual ACM Symposium on Theory of Computing (San Diego, Calif., 1978)*, pages 216–226. ACM, New York, 1978.

47 Johan Thapper and Stanislav Zivny. The complexity of finite-valued CSPs. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 695–704, 2013. `doi:10.1145/2488608.2488697`.

Parameterized Hardness of Art Gallery Problems[*]

Édouard Bonnet[1] and Tillmann Miltzow[2]

1 Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), Budapest, Hungary
edouard.bonnet@lamsade.dauphine.fr

2 Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), Budapest, Hungary
t.miltzow@gmail.com

Abstract

Given a simple polygon $\mathcal{P}$ on n vertices, two points x, y in $\mathcal{P}$ are said to be visible to each other if the line segment between x and y is contained in $\mathcal{P}$. The Point Guard Art Gallery problem asks for a minimum set S such that every point in $\mathcal{P}$ is visible from a point in S. The Vertex Guard Art Gallery problem asks for such a set S subset of the vertices of $\mathcal{P}$. A point in the set S is referred to as a guard. For both variants, we rule out a $f(k)n^{o(k/\log k)}$ algorithm, for any computable function f, where $k := |S|$ is the number of guards, unless the Exponential Time Hypothesis fails. These lower bounds almost match the $n^{O(k)}$ algorithms that exist for both problems.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases art gallery problem, computational geometry, parameterized complexity, ETH-based lower bound, geometric set cover/hitting set

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.19

1 Introduction

Given a simple polygon $\mathcal{P}$ on n vertices, two points x, y in $\mathcal{P}$ are said to be visible to each other if the line segment between x and y is contained in $\mathcal{P}$. The Point Guard Art Gallery problem asks for a minimum set S such that every point in $\mathcal{P}$ is visible from a point in S. The Vertex Guard Art Gallery problem asks for such a set S subset of the vertices of $\mathcal{P}$. The set S is referred to as guards. In what follows, n refers to the number of vertices of $\mathcal{P}$ and k to the size of an optimal set of guards.

The art gallery problem is arguably one of the most well-known problems in discrete and computational geometry. Since its introduction by Viktor Klee in 1976, three books [32, 34, 14] and two extensive surveys appeared [33, 8]. O'Rourke's book from 1987 has over a thousand citations, and each year, top conferences publish new results on the topic. Many variants of the art gallery problem, based on different definitions of visibility, restricted classes of polygons, different shapes of guards, have been defined and analyzed. One of the first results is the elegant proof of Fisk that $\lfloor n/3 \rfloor$ guards are always sufficient and sometimes necessary for a polygon with n vertices [12].

NP-hardness and APX-hardness have been shown for many variants of the art gallery problem and other related problems [11, 23, 4, 26]. Due to those negative results, most

[*] Supported by the ERC grant PARAMTIGHT: "Parameterized complexity and the search for tight complexity results", no. 280152.

24th Annual European Symposium on Algorithms (ESA 2016).
Editors: Piotr Sankowski and Christos Zaroliagis; Article No. 19; pp. 19:1–19:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

papers concentrate on finding approximation algorithms and restrictions that are polynomially tractable [15, 25, 24, 30, 26]. However, considering the recent lack of progress in this direction, the study of other approaches becomes interesting. One such approach is finding heuristics to solve large instances of the art gallery problem [8]. The fundamental drawback of this approach is the lack of performance guarantee.

In the last twenty-five years, a fruitful paradigm, parameterized complexity, has been gaining some popularity. The underlying idea is to study algorithmic problems with dependence on a natural parameter. If the dependence on the parameter is practical and the parameter is small for real-life instances, one gets algorithms that give optimal solutions with reasonable running times. For a gentle introduction to parameterized complexity, we recommend Niedermeier's book [31]. For a thorough reading highlighting complexity classes, we suggest the book by Downey and Fellows [9]. For a recent book on the topic with an emphasize on algorithms, we advise to read the book by Cygan et al. [6]. An approach based on logic is given by Flum and Grohe [13]. Despite the recent successes of parameterized complexity, only very few results on the art gallery problem are known.

The first such result is the trivial algorithm for the vertex guard variant to check if a solution of size k exists in a polygon with n vertices. The algorithm runs in $O(n^{k+2})$ time, by checking all possible subsets of size k of the vertices. The second *not so well-known* result is the fact that one can find in time $n^{O(k)}$ a set of k guards for the point guard variant, if it exists [10], using tools from real algebraic geometry [2]. This was first observed by Sharir [10, Acknowledgment]. Despite the fact that the first algorithm is extremely basic and the second algorithm, even with remarkably sophisticated tools, uses almost no problem-specific insights, no better exact parameterized algorithms are known.

The Exponential Time Hypothesis (ETH) asserts that there is no $2^{o(N)}$ time algorithm for Sat on N variables. The ETH is used to attain more precise conditional lower bounds than the mere NP-hardness. A simple reduction from Set Cover by Eidenbenz et al. shows that there is no $n^{o(k)}$ algorithm for these problems, when we consider polygons with holes [11, Sec.4], unless the ETH fails. However, polygons with holes are very different from simple polygons. For instance, they have unbounded VC-dimension while simple polygons have bounded VC-dimension [35, 20, 22, 19]. Our contribution is to show that, even on simple polygons, one cannot expect a large improvement over the $n^{O(k)}$ algorithms. More precisely, we prove:

▶ **Theorem 1** (Parameterized hardness point guard). *Assuming the ETH,* Point Guard Art Gallery *is not solvable in time $f(k)\, n^{o(k/\log k)}$, for any computable function f, even on simple polygons, where n is the number of vertices of the polygon and k is the number of guards allowed.*

▶ **Theorem 2** (Parameterized hardness vertex guard). *Assuming the ETH,* Vertex Guard Art Gallery *is not solvable in time $f(k)\, n^{o(k/\log k)}$, for any computable function f, even on simple polygons, where n is the number of vertices of the polygon and k is the number of guards allowed.*

Our reductions are from Subgraph Isomorphism. Therefore an algorithm solving the art gallery problem in time $f(k)\, n^{o(k/\log k)}$ would also improve current running times for Subgraph Isomorphism and for solving CSPs parameterized by treewidth, which are major open questions [28]. Our results imply, in particular, that both variants are $W[1]$-hard parameterized by the number of guards.

Finally, let us mention a sample of works on the parameterized complexity (with an emphasis on hardness) of other geometric problems. The complexity of some fundamental

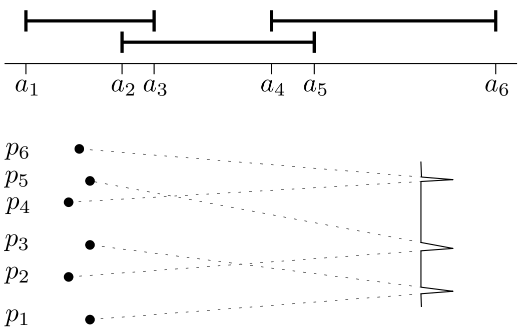


Figure 1 Reduction from Hitting Set on interval graphs to a restricted version of the art gallery problem.

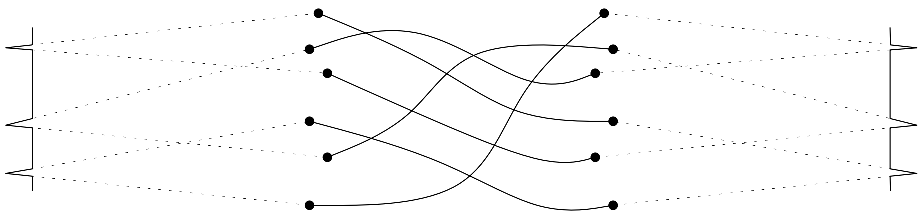


Figure 2 Two instances of Hitting Set "magically" linked.

problems parameterized by the dimension d has been addressed [17]; it was shown that, assuming the ETH, algorithms running in time $n^{O(d)}$ are essentially optimal (with n being the size of the instance). Extracting from a finite set of points of $\mathbb{R}^3$ the largest subset in convex position and whose convex-hull interior is empty is W[1]-hard [16]. More results on geometric covering or packing problems include the following papers [5, 27, 1, 29, 7]. We refer the interested reader to the extensive survey of Giannopoulos et al. [18].

Proof ideas. In order to achieve these results, we slightly extend some known hardness results of geometric set cover/hitting set problems and combine them with problem-specific insights of the art gallery problem. One of the first problem-specific insights is the ability to encode Hitting Set on interval graphs. The reader can refer to Figure 1 for the following description. Assume that we have some fixed points $p_1, \ldots, p_n$ with increasing y-coordinates in the plane. We can build a pocket "far enough to the right" that can be seen only from $\{p_i, \ldots, p_j\}$ for any $1 \leq i < j \leq n$.

Let $I_1, \ldots, I_n$ be n intervals with endpoints $a_1, \ldots, a_{2n}$. Then, we construct $2n$ points $p_1, \ldots, p_{2n}$ representing $a_1, \ldots, a_{2n}$. Further, we construct one pocket "far enough to the right" for each interval as described above. This way, we reduce Hitting Set on interval graphs to a restricted version of the art gallery problem. This observation is *not* so useful in itself since hitting set on interval graphs can be solved in polynomial time.

The situation changes rapidly if we consider Hitting Set on 2-track interval graphs, as described in Section 2. Unfortunately, we are not able to just "magically" link some specific pairs of points in the polygon of the art gallery instance. Therefore, we construct linker gadgets, which basically work as follows. We are given two set of points P and Q and a bijection σ between P and Q. The linker gadget is built in a way that it can be covered by two points (p, q) of $P \times Q$, if and only if $q = \sigma(p)$. The Structured 2-Track Hitting Set problem will be specifically designed so that the linker gadget is the main remaining ingredient to show hardness.

Organization of the paper. In Section 2, we introduce some notations, discuss the encoding of the polygon, and give some useful ETH-based lower bounds. We show a lower bound for Structured 2-Track Hitting Set based on the lower bound known for Multicolored Subgraph Isomorphism. Due to space limitation, this proof is only included in the arxiv version of the paper [3]. Then, we reduce from the particularly convenient Structured 2-Track Hitting Set. In Section 3, we show the lower bound for the Point Guard Art Gallery problem (Theorem 1). We design a linker gadget, show its correctness, and show how several linker gadgets can be combined consistently. In Section 4, we tackle the Vertex Guard Art Gallery problem (Theorem 2). We have to design a very different linker gadget, that has to be combined with other gadgets and ideas.

2 Preliminaries

For any two integers $x \leqslant y$, we set $[x, y] := \{x, x+1, \dots, y-1, y\}$, and for any positive integer x, $[x] := [1, x]$. Given two points a, b in the plane, we define $\mathrm{seg}(a, b)$ as the line segment with endpoints a, b. Given n points $v_1, \dots, v_n \in \mathbb{R}^2$, we define a polygonal closed curve c by $\mathrm{seg}(v_1, v_2)$, …, $\mathrm{seg}(v_{n-1}, v_n)$, $\mathrm{seg}(v_n, v_1)$. If c is not self intersecting, it partitions the plane into a closed bounded area and an unbounded area. The closed bounded area is a *simple polygon* on the vertices $v_1, \dots, v_n$. Note that we do not consider the boundary as the polygon but rather all the points bounded by the curve c as described above. Given two points a, b in a simple polygon $\mathcal{P}$, we say that *a sees b* or a is *visible* from b if $\mathrm{seg}(a, b)$ is contained in $\mathcal{P}$. By this definition, it is possible to "see through" vertices of the polygon. We say that S is a set of *point guards* of $\mathcal{P}$, if every point $p \in \mathcal{P}$ is visible from a point of S. We say that S is a set of *vertex guards* of $\mathcal{P}$, if additionally S is a subset of the vertices of $\mathcal{P}$. The Point Guard Art Gallery problem and the Vertex Guard Art Gallery problem are formally defined as follows.

Point Guard Art Gallery
Input: The vertices of a simple polygon $\mathcal{P}$ in the plane and a natural number k.
Question: Does there exist a set of k point guards for $\mathcal{P}$?

Vertex Guard Art Gallery
Input: A simple polygon $\mathcal{P}$ on n vertices in the plane and a natural number k.
Question: Does there exist a set of k vertex guards for $\mathcal{P}$?

For any two distinct points v and w in the plane we denote by $\mathrm{ray}(v, w)$ the ray starting at v and passing through w, and by $\ell(v, w)$ the supporting line passing through v and w. For any point x in a polygon $\mathcal{P}$, $V_{\mathcal{P}}(x)$, or simply $V(x)$, denotes the *visibility region* of x within $\mathcal{P}$, that is the set of all the points $y \in \mathcal{P}$ seen by x. We say that two vertices v and w of a polygon $\mathcal{P}$ are *neighbors* or *consecutive* if vw is an edge of $\mathcal{P}$. A *sub-polygon* $\mathcal{P}'$ of a simple polygon $\mathcal{P}$ is defined by any l distinct consecutive vertices $v_1, v_2, \dots, v_l$ of $\mathcal{P}$ (that is, for every $i \in [l-1]$, v_i and v_{i+1} are neighbors in $\mathcal{P}$) such that $v_1 v_l$ does not cross any edge of $\mathcal{P}$. In particular, $\mathcal{P}'$ is a simple polygon.

We assume that the vertices of the polygon are either given by integers or by rational numbers. We also assume that the output is given either by integers or by rational numbers. The instances we generate as a result of Theorem 1 and Theorem 2 have rational coordinates. We can represent them by specifying the nominator and denominator. The number of bits is bounded by $O(\log n)$ in both cases. We can transform the coordinates to integers by multiplying every coordinate with the least common multiple of all denominators. However, this leads to integers using $O(n \log n)$ bits.

ETH-based lower bounds. The *Exponential Time Hypothesis* (ETH) is a conjecture by Impagliazzo et al. [21] asserting that there is no $2^{o(n)}$-time algorithm for 3-SAT on instances with n variables.

The MULTICOLORED SUBGRAPH ISOMORPHISM problem can be defined in the following equivalent way. One is given a graph with n vertices partitioned into l color classes $V_1, \dots, V_l$ such that only k of the $\binom{l}{2}$ sets $E_{ij} = E(V_i, V_j)$ are non empty. The goal is to pick one vertex in each color class so that the selected vertices induce k edges. Observe that l corresponds to the number of vertices of the pattern graph. The technique of color coding and a result of Marx imply that:

▶ **Theorem 3** ([28]). *Unless the ETH fails,* MULTICOLORED SUBGRAPH ISOMORPHISM *cannot be solved in time $f(k)\, n^{o(k/\log k)}$ where k is the number of edges of the solution and f any computable function.*

Naturally, this result still holds when restricted to connected input graphs. In that case, $k \geqslant l - 1$.

In the 2-TRACK HITTING SET problem, the input consists of an integer k, two totally ordered ground sets A and B of the same cardinality, and two sets $\mathcal{S}_A$ of A-intervals, and $\mathcal{S}_B$ of B-intervals. In addition, the elements of A and B are in one-to-one correspondence $\phi : A \to B$ and each pair $(a, \phi(a))$ is called a 2-*element*. The goal is to find, if possible, a set S of k 2-elements such that the first projection of S is a hitting set of $\mathcal{S}_A$, and the second projection of S is a hitting set of $\mathcal{S}_B$.

STRUCTURED 2-TRACK HITTING SET is the same problem with color classes over the 2-elements, and a restriction on the one-to-one mapping ϕ. Given two integers k and t, A is partitioned into $(C_1, C_2, \dots, C_k)$ where $C_j = \{a^j_1, a^j_2, \dots, a^j_t\}$ for each $j \in [k]$. A is ordered: $a^1_1, a^1_2, \dots, a^1_t, a^2_1, a^2_2, \dots, a^2_t, \dots, a^k_1, a^k_2, \dots, a^k_t$. We define $C'_j := \phi(C_j)$ and $b^j_i := \phi(a^j_i)$ for all $i \in [t]$ and $j \in [k]$. We now impose that ϕ is such that, for each $j \in [k]$, the set C'_j is a B-interval. That is, B is ordered: $C'_{\sigma(1)}, C'_{\sigma(2)}, \dots, C'_{\sigma(k)}$ for some permutation on $[k]$, $\sigma \in \mathfrak{S}_k$. For each $j \in [k]$, the order of the elements within C'_j can be described by a permutation $\sigma_j \in \mathfrak{S}_t$ such that the ordering of C'_j is: $b^j_{\sigma_j(1)}, b^j_{\sigma_j(2)}, \dots, b^j_{\sigma_j(t)}$. In what follows, it will be convenient to see an instance of STRUCTURED 2-TRACK HITTING SET as a tuple $\mathcal{I} = (k \in \mathbb{N}, t \in \mathbb{N}, \sigma \in \mathfrak{S}_k, \sigma_1 \in \mathfrak{S}_t, \dots, \sigma_k \in \mathfrak{S}_t, \mathcal{S}_A, \mathcal{S}_B)$, where we recall that $\mathcal{S}_A$ is a set of A-intervals and $\mathcal{S}_B$ is a set of B-intervals. We denote by $[a^j_i, a^{j'}_{i'}]$ (resp. $[b^j_i, b^{j'}_{i'}]$) all the elements $a \in A$ (resp. $b \in B$) such that $a^j_i \leq_A a \leq_A a^{j'}_{i'}$ (resp. $b^j_i \leq_B b \leq_B b^{j'}_{i'}$).

Taking inspiration from previous results, we show hardness of STRUCTURED 2-TRACK HITTING SET by a reduction from MULTICOLORED SUBGRAPH ISOMORPHISM. Due to lack of space, we do no include the proof of the following theorem. The interested reader can find this proof in the arxiv version of the paper [3].

▶ **Theorem 4.** STRUCTURED 2-TRACK HITTING SET *is $W[1]$-hard, and not solvable in time $f(k)\, |\mathcal{I}|^{o(k/\log k)}$ for any computable function f, unless the ETH fails.*

3 Parameterized hardness of the point guard variant

As exposed in the introduction, we give a reduction from the STRUCTURED 2-TRACK HITTING SET problem. The main challenge is to design a *linker* gadget that groups together specific pairs of points in the polygon. The following introductory lemma inspires the *linker* gadgets for both POINT GUARD ART GALLERY and VERTEX GUARD ART GALLERY.

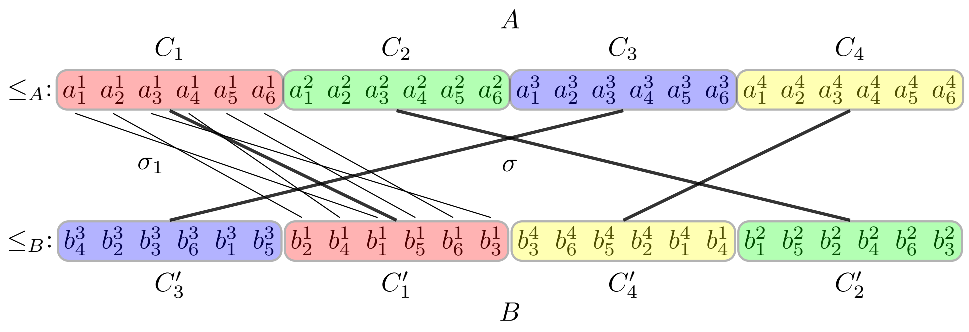

A
C_1
C_2
C_3
C_4
$\leq_A$:
a^1_1 a^1_2 a^1_3 a^1_4 a^1_5 a^1_6
a^2_1 a^2_2 a^2_3 a^2_4 a^2_5 a^2_6
a^3_1 a^3_2 a^3_3 a^3_4 a^3_5 a^3_6
a^4_1 a^4_2 a^4_3 a^4_4 a^4_5 a^4_6
σ_1
σ
$\leq_B$:
b^3_4 b^3_2 b^3_3 b^3_6 b^3_1 b^3_5
b^1_2 b^1_4 b^1_1 b^1_5 b^1_6 b^1_3
b^4_3 b^4_6 b^4_5 b^4_2 b^4_1 b^1_4
b^2_1 b^2_5 b^2_2 b^2_4 b^2_6 b^2_3
C'_3
C'_1
C'_4
C'_2
B

Figure 3 An illustration of the $k+1$ permutations $\sigma \in \mathfrak{S}_k$, $\sigma_1 \in \mathfrak{S}_t$, ..., $\sigma_k \in \mathfrak{S}_t$ of an instance of Structured 2-Track Hitting Set, with $k = 4$ and $t = 6$.

▶ **Lemma 5.** *The only minimum hitting sets of the set-system* $\mathcal{S} = \{S_i = \{1, 2, \dots, i, \overline{i+1}, \overline{i+2}, \dots, \overline{n}\} \mid i \in [n]\} \cup \{\overline{S}_i = \{\overline{1}, \overline{2}, \dots, \overline{i}, i+1, i+2, \dots, n\} \mid i \in [n]\}$ *are* $\{i, \overline{i}\}$*, for each* $i \in [n]$.

Proof. First, for each $i \in [n]$, one may easily observe that $\{i, \overline{i}\}$ is a hitting set of $\mathcal{S}$. Now, because of the sets S_n and $\overline{S}_n$ one should pick one element i and one element $\overline{j}$ for some $i, j \in [n]$. If $i < j$, then set $\overline{S}_i$ is not hit, and if $i > j$, then S_j is not hit. Therefore, i should be equal to j. ◀

▶ **Theorem 1** (Parameterized hardness point guard). *Assuming the ETH,* Point Guard Art Gallery *is not solvable in time* $f(k)\, n^{o(k/\log k)}$*, for any computable function* f*, even on simple polygons, where* n *is the number of vertices of the polygon and* k *is the number of guards allowed.*

Proof. Given an instance $\mathcal{I} = (k \in \mathbb{N}, t \in \mathbb{N}, \sigma \in \mathfrak{S}_k, \sigma_1 \in \mathfrak{S}_t, \dots, \sigma_k \in \mathfrak{S}_t, \mathcal{S}_A, \mathcal{S}_B)$ of Structured 2-Track Hitting Set, we build a simple polygon $\mathcal{P}$ with $O(kt + |\mathcal{S}_A| + |\mathcal{S}_B|)$ vertices, such that $\mathcal{I}$ is a YES-instance iff $\mathcal{P}$ can be guarded by $3k$ points.

Outline. We recall that A's order is: $a^1_1, \dots, a^1_t, \dots, a^k_1, \dots, a^k_t$ and B's order is determined by σ and the σ_j's (see Figure 3). Let us focus on one color class $j \in [k]$ together with a permutation $\sigma_j : A \to B$. The global strategy of the reduction is to *allocate*, $2t$ special points for this polygon. The points $a^j_1, \dots, a^j_t$ on track A are represented by $\alpha^j_1, \dots, \alpha^j_t$ points in $\mathcal{P}$. and the points $\sigma_j(a^j_1), \dots, \sigma_j(a^j_t)$ on track B are represented by $\beta^j_1, \dots, \beta^j_t$ in the polygon. Placing a guard in α^j_i and β^j_i shall correspond to picking the 2-element $(a^j_i, \sigma_j(b^j_i))$. The points α^j_i's and β^j_i's ordered by increasing y-coordinates will match the order of the a^j_i's along the order $\leq_A$ and then of the b^j_i's along $\leq_B$. Then, far in the horizontal direction, we will place pockets to encode each A-interval of $\mathcal{S}_A$, and each B-interval of $\mathcal{S}_B$.

The first critical issue will be to *link* point α^j_i to point β^j_i. Indeed, in the Structured 2-Track Hitting Set problem, one selects 2-elements (one per color class), so we should prevent one from placing two guards in α^j_i and $\beta^j_{i'}$ with $i \neq i'$. The so-called *point linker* gadget will realize the intervals as described in Lemma 5.

The second critical issue is to enforce these positions. For this purpose, we will need to introduce a *copy* $\overline{\alpha}^j_i$ of each α^j_i. In each part of the gallery encoding a color class $j \in [k]$, the only way of guarding all the pockets with only three guards will be to place them in α^j_i, $\overline{\alpha}^j_i$, and β^j_i for some $i \in [t]$ (see Figure 5). Hence, $3k$ guards will be necessary and sufficient to guard the whole $\mathcal{P}$ iff there is a solution to the instance of Structured 2-Track Hitting Set.

We now get into the details of the reduction. We will introduce several characteristic lengths and compare them; when $l_1 \ll l_2$ means that l_1 should be thought as really small compared to l_2, and $l_1 \approx l_2$ means that l_1 and l_2 are roughly of the same order. The motivation is to guide the intuition of the reader without bothering her/him too much about the details. At the end of the construction, we will specify more concretely how those lengths are chosen.

Construction. We start with an explicit specification of the coordinates. The description will be dependent on some parameters x, y, L, D, F that we will specify later. The value x represents the offset between elements with respect to the x-coordinate and likewise the value y represents the offset between elements with respect to the y-coordinate. D represents the vertical distance between different color classes and L represents the horizontal distance between all the $\alpha's$ and the $\beta's$, see also Figure 6. The value F will become relevant later and describes the distance of the points to the pockets to the far right. The crucial point of the construction is that the order of the α's corresponds exactly to the order of the a's along track A and the same relation holds between the β's and b's.

We recall that we want the points α_i^j's and β_i^j's ordered by increasing y-coordinates, to match the order of the a_i^j's and b_i^j's along $\leq_A$ and $\leq_B$, with first all the elements of A and then all the elements of B. Starting from some y-coordinate y_1 (which is the one given to point α_1^1), the y-coordinates of the α_i^j's are regularly spaced out by an offset y; that is, the y-coordinate of α_i^j is $y_1 + (i + (j-1)t)y$. Between the y-coordinate of the last element in A (i.e., a_t^k whose y-coordinate is $y_1 + (kt-1)y$) and the first element in B, there is a large offset L, such that the y-coordinate of β_i^j is $y_1 + (kt-1)y + L + (\text{ord}(b_i^j) - 1)y$ (for any $j \in [k]$ and $i \in [t]$) where $\text{ord}(b_i^j)$ is the rank of b_i^j along the order $\leq_B$.

For each color class $j \in [k]$, let $x_j := x_1 + (j-1)D$ for some x-coordinate x_1 and value D, and $y_j := y_1 + (j-1)ty$. The allocated points $\alpha_1^j, \alpha_2^j, \alpha_3^j, \ldots, \alpha_t^j$ are on a line at coordinates: $(x_j, y_j), (x_j + x, y_j + y), (x_j + 2x, y_j + 2y), \ldots, (x_j + (t-1)x, y_j + (t-1)y)$, for some value x. We place, to the left of those points, a rectangular pocket $\mathcal{P}_{j,r}$ of width, say, y and length, say[1], tx such that the uppermost longer side of the rectangular pocket lies on the line $\ell(\alpha_1^j, \alpha_t^j)$ (see Figure 4). The y-coordinates of $\beta_1^j, \beta_2^j, \beta_3^j, \ldots, \beta_t^j$ have already been defined. We set, for each $i \in [t]$, the x-coordinate of β_i^j to $x_j + (i-1)x$, so that β_i^j and α_i^j share the same x-coordinate. One can check that it is consistent with the previous paragraph. We also observe that, by the choice of the y-coordinate for the β_i^j's, we have both encoded the permutations σ_j's and permutation σ (see Figure 6 or Figure 4). This finishes the description of the coordinates.

Now, we will give a description how, we can encode intervals by on track A and B by small pockets and, we describe, where to place them. From hereon, for a vertex v and two points p and p', we informally call *triangular pocket rooted at vertex v and supported by ray(v,p) and ray(v,p')* a sub-polygon w, v, w' (a triangle) such that $\text{ray}(v,w)$ passes through p, $\text{ray}(v,w')$ passes through p', while w and w' are close to v (sufficiently close not to interfere with the rest of the construction). We say that v is the *root* of the triangular pocket, that we often denote by $\mathcal{P}(v)$. We also say that the pocket $\mathcal{P}(v)$ *points* towards p and p'. It is easy to see that each point that sees v also sees the entire triangular pocket $P(v)$.

[1] The exact width and length of this pocket are not relevant; the reader may just think of $\mathcal{P}_{j,r}$ as a thin pocket which forces to place a guard on a thin strip whose uppermost boundary is $\ell(\alpha_1^j, \alpha_t^j)$

For each A-interval $I_q = [a_i^j, a_{i'}^{j'}] \in \mathcal{S}_A$ we construct one triangular pocket $\mathcal{P}(z_{A,q})$ rooted at vertex $z_{A,q}$ and supported by $\text{ray}(z_{A,q}, \alpha_i^j)$ and $\text{ray}(z_{A,q}, \alpha_{i'}^{j'})$. The placement of this triangular pocket is very far to the right. The x-coordinate of $z_{A,q}$ equals $x_k + (t-1)x + F$, for some large value F to be specified later. The y-coordinate shall be between y_1 and $y_k + (kt-1)y$. We place those $|\mathcal{S}_A|$ pockets along the y-axis, and space them out by some small distance s. To guarantee that we have enough room to place all those pockets, s will be chosen sufficiently small ($s \ll y$).

We will show later, for appropriate values $y \ll x \ll D \ll F$, the only $\alpha_{i''}^{j''}$ seeing vertex $z_{A,q}$ should be the points such that $a_i^j \leq_A a_{i''}^{j''} \leq_A a_{i'}^{j'}$ (see Figure 6).

Similarly, we represent each interval $I_q \in \mathcal{S}_B$ by a triangular pocket rooted at $z_{B,q}$. These pockets are placed at the x-coordinate $x_k + (t-1)x + F$ and spaced out by distance s along the y-axis between y-coordinates $y_1 + (kt-1)y + L$ and $y_1 + 2(kt-1)y + L$. The B-interval $I_q = [b_i^j, b_{i'}^{j'}]$ is represented by the triangular pocket $\mathcal{P}(z_{B,q})$ rooted at vertex $z_{B,q}$ supported by $\text{ray}(z_{B,q}, \sigma_j(a_i^j))$ and $\text{ray}(z_{B,q}, \sigma_j(a_{i'}^{j'}))$. Note that $\sigma_j(a_i^j)$ is the point on track B that corresponds to β_i^j. The different values (s, x, y, D, L, and F) introduced so far compare in the following way: $s \ll y \ll x \ll D \ll F$, and $x \ll L \ll F$, see Figure 6.

Now, we describe how we *link* each point α_i^j to its associate β_i^j. For each $j \in [k]$, let us mentally draw $\text{ray}(\alpha_t^j, \beta_1^j)$ and consider points slightly to the left of this ray at a distance, say, L' from point α_t^j. Let us call $\mathcal{R}_{\text{left}}^j$ that informal region of points. Any point in $\mathcal{R}_{\text{left}}^j$ sees, from right to left, in the order α_1^j, α_2^j up to α_t^j, and then, β_1^j, β_2^j up to β_t^j. This observation relies on the fact that $y \ll x \ll L$. So, from the distance, the points $\beta_1^j, \dots, \beta_t^j$ look almost *flat*. It makes the following construction possible. In $\mathcal{R}_{\text{left}}^j$, for each $i \in [t-1]$, we place a triangular pocket $\mathcal{P}(c_i^j)$ rooted at vertex c_i^j and supported by $\text{ray}(c_i^j, \alpha_{i+1}^j)$ and $\text{ray}(c_i^j, \beta_i^j)$. We place also a triangular pocket $\mathcal{P}(c_t^j)$ rooted at c_t^j supported by $\text{ray}(c_i^j, \beta_1^j)$ and $\text{ray}(c_i^j, \beta_t^j)$. We place vertices c_i^j and c_{i+1}^j at the same y-coordinate and spaced out by distance x along the x-axis (see Figure 4). Similarly, let us informally refer to the region slightly to the right of $\text{ray}(\alpha_1^j, \beta_t^j)$ at a distance L' from point α_1^j, as $\mathcal{R}_{\text{right}}^j$. Any point $\mathcal{R}_{\text{right}}^j$ sees, from right to left, in this order β_1^j, β_2^j up to β_t^j, and then, α_1^j, α_2^j up to α_t^j. Therefore, one can place in $\mathcal{R}_{\text{left}}^j$, for each $i \in [t-1]$, a triangular pocket $\mathcal{P}(d_i^j)$ rooted at d_i^j supported by $\text{ray}(d_i^j, \beta_{i+1}^j)$ and $\text{ray}(c_i^j, \alpha_i^j)$. We place also a triangular pocket $\mathcal{P}(d_t^j)$ rooted at d_t^j supported by $\text{ray}(d_i^j, \alpha_1^j)$ and $\text{ray}(c_i^j, \alpha_t^j)$. Again, those t pockets are placed at the same y-coordinate and spaced out horizontally by x (see Figure 4). We denote by $\mathcal{P}_{j,\alpha,\beta}$ the set of pockets $\{\mathcal{P}(c_1^j), \dots, \mathcal{P}(c_t^j), \mathcal{P}(d_1^j), \dots, \mathcal{P}(d_t^j)\}$ and informally call it the *weak point linker* (or simply, *weak linker*) of $\alpha_1^j, \dots, \alpha_t^j$ and $\beta_1^j, \dots, \beta_t^j$. We may call the pockets of $\mathcal{R}_{\text{left}}^j$ (resp. $\mathcal{R}_{\text{right}}^j$) *left* pockets (resp. *right* pockets).

As we will show later, if one wants to guard with only two points all the pockets of $\mathcal{P}_{j,\alpha,\beta} = \{\mathcal{P}(c_1^j), \dots, \mathcal{P}(c_t^j), \mathcal{P}(d_1^j), \dots, \mathcal{P}(d_t^j)\}$ and one first decides to put a guard on point α_i^j (for some $i \in [t]$), then one is not forced to put the other guard on point β_i^j but only on an area whose uppermost point is β_i^j (see the shaded areas below the b_i^j's in Figure 4). Now, if the points $\beta_1^j, \dots, \beta_t^j$ would all lie on a common line ℓ, we could shrink the shaded area of each β_i^j (Figure 4) down to the single point β_i^j by adding a thin rectangular pocket on ℓ (similarly to what we have for $\alpha_1^j, \dots, \alpha_t^j$). Naturally, we need that $\beta_1^j, \dots, \beta_t^j$ are *not* on a common line to be able to encode the permutation σ_j. The remedy we pursue is the following. For each $j \in [k]$, we allocate t points $\overline{\alpha}_1^j, \overline{\alpha}_2^j, \dots, \overline{\alpha}_t^j$ on a horizontal line, spaced out by distance x, say, $\approx \frac{D}{2}$ to the right and $\approx L$ above of β_t^j. We place a thin horizontal rectangular pocket $\mathcal{P}_{j,\overline{r}}$ of the same dimension as $\mathcal{P}_{j,r}$ such that the lowermost longer side of $\mathcal{P}_{j,\overline{r}}$ is on the line $\ell(\overline{\alpha}_1^j, \overline{\alpha}_t^j)$. We add the $2t$ pockets corresponding to a weak

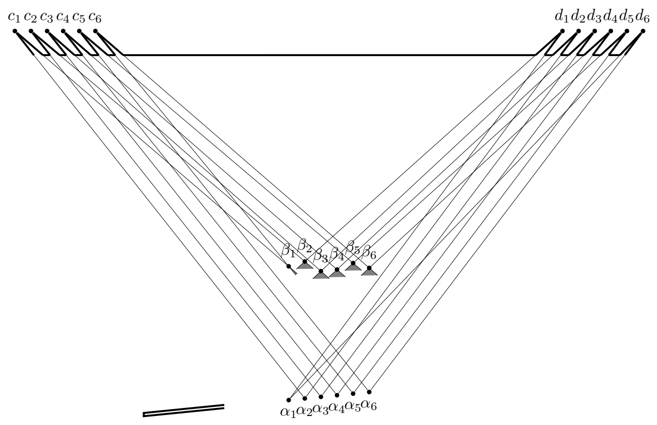

$c_1\,c_2\,c_3\,c_4\,c_5\,c_6$
$d_1d_2d_3d_4d_5d_6$
$\beta_1\beta_2\beta_3\beta_4\beta_5\beta_6$
$\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5\alpha_6$

Figure 4 Weak point linker gadget.

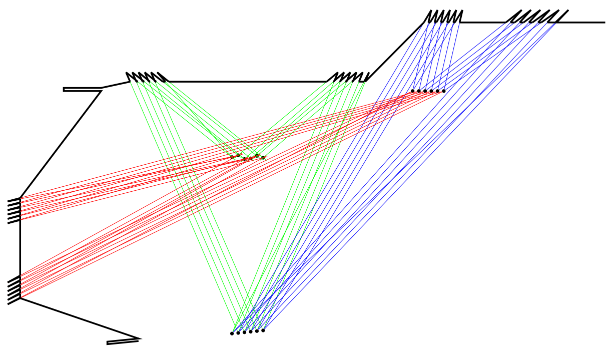
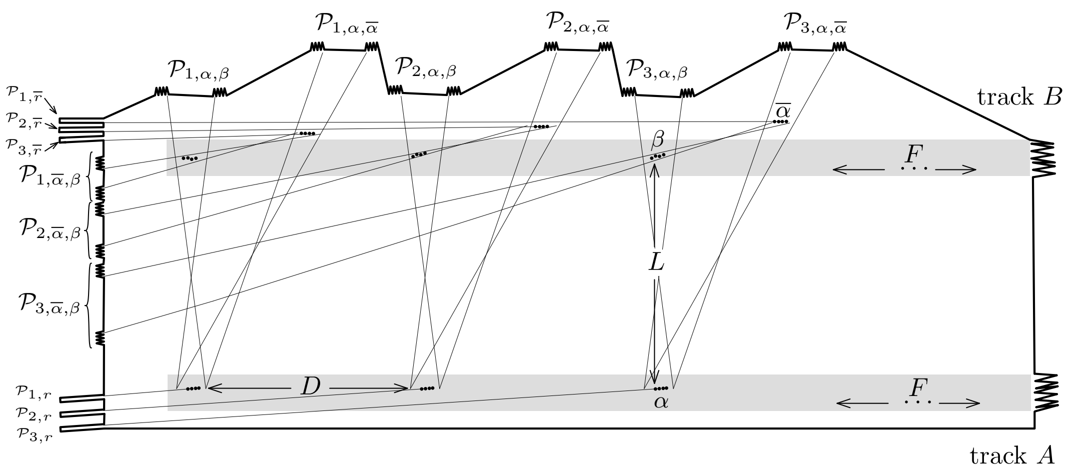


Figure 5 Point linker gadget: a triangle of (three) weak point linkers.

linker $\mathcal{P}_{j,\alpha,\overline{\alpha}}$ between $\alpha^j_1, \ldots, \alpha^j_t$ and $\overline{\alpha}^j_1, \ldots, \overline{\alpha}^j_t$ as well as the $2t$ pockets of a weak linker $\mathcal{P}_{j,\overline{\alpha},\beta}$ between $\overline{\alpha}^j_1, \ldots, \overline{\alpha}^j_t$ and $\beta^j_1, \ldots, \beta^j_t$ as pictured in Figure 5. We denote by $\mathcal{P}_j$ the union $\mathcal{P}_{j,r} \cup \mathcal{P}_{j,\overline{r}} \cup \mathcal{P}_{j,\alpha,\beta} \cup \mathcal{P}_{j,\alpha,\overline{\alpha}} \cup \mathcal{P}_{j,\overline{\alpha},\beta}$ of all the pockets involved in the encoding of color class j. Now, say, one wants to guard all the pockets of $\mathcal{P}_j$ with only three points, and chooses to put a guard on α^j_i (for some $i \in [t]$). Because of the pockets of $\mathcal{P}_{j,\alpha,\overline{\alpha}} \cup P_{j,\overline{r}}$, one is forced to place a second guard precisely on $\overline{\alpha}^j_i$. Now, because of the weak linker $\mathcal{P}_{j,\alpha,\beta}$ the third guard should be on a region whose uppermost point is β^j_i, while, because of $\mathcal{P}_{j,\overline{\alpha},\beta}$ the third guard should be on a region whose lowermost point is β^j_i. The conclusion is that the third guard should be put precisely on β^j_i. This *triangle* of weak linkers is called the *linker* of color class j. The k linkers are placed accordingly to Figure 6. This ends the construction.

Specification of the distances. We can specify the coordinates of positions of all the vertices by fractions of integers. These integers are polynomially bounded in n. If we want to get integer coordinates, we can transform the rational coordinates to integer coordinates by multiplying all of them with the least common multiple of all the denominators, which is not polynomially bounded anymore. The length of the integers in binary is still polynomially bounded.

We can safely set s to one, as it is the smallest length, we specified. We will put $|\mathcal{S}_a|$ pockets on track A and $|\mathcal{S}_b|$ pockets on track B. It is sufficient to have an opening space of one between them. Thus, the space on the right side of $\mathcal{P}$, for all pockets of track A is bounded by $2|\mathcal{S}_a|$. Thus setting y to $|\mathcal{S}_a| + |\mathcal{S}_b|$ secures us that we have plenty of space to



$\mathcal{P}_{1,\alpha,\overline{\alpha}}$
$\mathcal{P}_{2,\alpha,\overline{\alpha}}$
$\mathcal{P}_{3,\alpha,\overline{\alpha}}$
$\mathcal{P}_{1,\alpha,\beta}$
$\mathcal{P}_{2,\alpha,\beta}$
$\mathcal{P}_{3,\alpha,\beta}$
$\mathcal{P}_{1,\overline{r}}$
$\mathcal{P}_{2,\overline{r}}$
$\mathcal{P}_{3,\overline{r}}$
$\mathcal{P}_{1,\overline{\alpha},\beta}$
$\mathcal{P}_{2,\overline{\alpha},\beta}$
$\mathcal{P}_{3,\overline{\alpha},\beta}$
track B
$\overline{\alpha}$
β
F
L
D
α
F
$\mathcal{P}_{1,r}$
$\mathcal{P}_{2,r}$
$\mathcal{P}_{3,r}$
track A

Figure 6 The overall picture of the reduction with $k = 3$.

place all the pockets. We specify $F = (|\mathcal{S}_a| + |\mathcal{S}_b|)Dk = yDk$. We have to show that this is large enough to guarantee that the pockets on track A distinguish the picked points only by the y-coordinate. Let p and q be two points among the α_i^j. Their vertical distance is upper bounded by Dk and their horizontal distance is lower bounded by y. Thus the slope of $\ell = \ell(p, q)$ is at least $\frac{y}{Dk}$. At the right side of $\mathcal{P}$ the line ℓ will be at least $F\frac{y}{Dk}$ above the pockets of track A. Note $F\frac{y}{Dk} = yDk\frac{y}{Dk} > y^2 > |\mathcal{S}_a|^2 > 2|\mathcal{S}_a|$. The same argument shows that F is sufficiently large for track B.

The remaining lengths x, L, L', and D can be specified in a similar fashion. For the construction of the pockets, let $s \in \mathcal{S}_a$ be an A-interval with endpoints a and b, represented by some points p and q and assume the opening vertices v and w of the triangular pocket are already specified. Then the two lines $\ell(p, v)$ and $\ell(q, w)$ will meet at some point x to the right of v and w. It is easy to see that x has rational coordinates and the integers to represent them can be expressed by the coordinates of p, q, v, and w. This way, all the pockets can be explicitly constructed using rational coordinates as claimed above.

Soundness. We now show that the reduction is correct. The following lemma is the main argument for the easier implication: *if $\mathcal{I}$ is a YES-instance, then the gallery that we build can be guarded with $3k$ points.*

▶ **Lemma 6.** *$\forall j \in [k]$, $\forall i \in [t]$, the three associate points α_i^j, $\overline{\alpha}_i^j$, β_i^j guard entirely $\mathcal{P}_j$.*

Proof. The rectangular pockets $\mathcal{P}_{j,r}$ and $\mathcal{P}_{j,\overline{r}}$ are entirely seen by respectively α_i^j and $\overline{\alpha}_i^j$. The pockets $\mathcal{P}(c_1^j), \mathcal{P}(c_2^j), \dots \mathcal{P}(c_{i-1}^j)$ and $\mathcal{P}(d_i^j), \mathcal{P}(d_{i+1}^j), \dots \mathcal{P}(d_t^j)$ are all entirely seen by α_i^j, while the pockets $\mathcal{P}(c_i^j), \mathcal{P}(c_{i+1}^j), \dots \mathcal{P}(c_t^j)$ and $\mathcal{P}(d_1^j), \mathcal{P}(d_2^j), \dots \mathcal{P}(d_{i-1}^j)$ are all entirely seen by β_i^j. This means that α_i^j and β_i^j jointly see all the pockets of $\mathcal{P}_{j,\alpha,\beta}$. Similarly, α_i^j and $\overline{\alpha}_i^j$ jointly see all the pockets of $\mathcal{P}_{j,\alpha,\overline{\alpha}}$, and $\overline{\alpha}_i^j$ and β_i^j jointly see all the pockets of $\mathcal{P}_{j,\overline{\alpha},\beta}$. Therefore, α_i^j, $\overline{\alpha}_i^j$, β_i^j jointly see all the pockets of $\mathcal{P}_j$. ◀

Assume that $\mathcal{I}$ is a YES-instance and let $\{(a_{s_1}^1, b_{s_1}^1), \dots, (a_{s_k}^k, b_{s_k}^k)\}$ be a solution. We claim that $G = \{\alpha_{s_1}^1, \overline{\alpha}_{s_1}^1, \beta_{s_1}^1, \dots, \alpha_{s_k}^k, \overline{\alpha}_{s_k}^k, \beta_{s_k}^k\}$ guard the whole polygon $\mathcal{P}$. By Lemma 6, $\forall j \in [k]$, $\mathcal{P}_j$ is guarded. For each A-interval (resp. B-interval) in $\mathcal{S}_A$ (resp. $\mathcal{S}_B$) there is at least one 2-element $(a_{s_j}^j, b_{s_j}^j)$ such that $a_{s_j}^j \in \mathcal{S}_A$ (resp. $b_{s_j}^j \in \mathcal{S}_B$). Thus, the corresponding pocket is guarded by $\alpha_{s_j}^j$ (resp. $\beta_{s_j}^j$). The rest of the polygon $\mathcal{P}$ (which is not part of pockets) is guarded by, for instance, $\{\overline{\alpha}^1_{s_1}, \ldots, \overline{\alpha}^k_{s_k}\}$. So, G is indeed a solution and it contains $3k$ points.

Assume now that there is no solution to the instance $\mathcal{I}$ of Structured 2-Track Hitting Set. We show that there is no set of $3k$ points guarding $\mathcal{P}$. We observe that no point of $\mathcal{P}$ sees inside two triangular pockets one being in $\mathcal{P}_{j,\alpha,\gamma}$ and the other in $\mathcal{P}_{j',\alpha,\gamma'}$ with $j \neq j'$ and $\gamma, \gamma' \in \{\beta, \overline{\alpha}\}$. Further, $V(r(\mathcal{P}_{j,\alpha,\beta} \cup \mathcal{P}_{j,\alpha,\overline{\alpha}})) \cap V(r(\mathcal{P}_{j',\alpha,\beta} \cup \mathcal{P}_{j',\alpha,\overline{\alpha}})) = \emptyset$ when $j \neq j'$, where r maps a set of triangular pockets to the set of their root. Also, for each $j \in [k]$, seeing entirely $\mathcal{P}_{j,\alpha,\beta}$ and $\mathcal{P}_{j,\alpha,\overline{\alpha}}$ requires at least 3 points. This means that for each $j \in [k]$, one should place three guards in $V(r(\mathcal{P}_{j,\alpha,\beta} \cup \mathcal{P}_{j,\alpha,\overline{\alpha}}))$. Furthermore, one can observe among those three points one should guard a triangular pocket $\mathcal{P}_{j',r}$ and another should guard $\mathcal{P}_{j'',\overline{r}}$. Let us try to guard entirely $\mathcal{P}_1$ and two rectangular pockets $\mathcal{P}_{j',r}$ and $\mathcal{P}_{j'',\overline{r}}$, with only three guards. Let call ℓ_1 (resp. ℓ'_1) the line corresponding to the extension of the uppermost (resp. lowermost) longer side of $\mathcal{P}_{1,r}$ (resp. $\mathcal{P}_{1,\overline{r}}$). The only points of $\mathcal{P}$ that can see a rectangular pocket $\mathcal{P}_{j',r}$ and at least t pockets of $\mathcal{P}_{1,\alpha,\overline{\alpha}}$ are on ℓ_1: more specifically, they are the points $\alpha^1_1, \ldots, \alpha^1_t$. The only points that can see a rectangular pocket $\mathcal{P}_{j'',\overline{r}}$ and at least t pockets of $\mathcal{P}_{1,\alpha,\overline{\alpha}}$ are on ℓ'_1: they are the points $\overline{\alpha}^1_1, \ldots, \overline{\alpha}^1_t$. As $\mathcal{P}_{1,\alpha,\overline{\alpha}}$ has $2t$ pockets, one has to take a point α^1_i and $\overline{\alpha}^1_{i'}$. By the same argument argument as in Lemma 5, i should be equal to i' (otherwise, $i < i'$ and the left pocket pointing towards $\overline{\alpha}^1_{i'-1}$ and $\alpha^1_{i'}$ is not seen, or $i > i'$ and the right pocket pointing towards α^1_{i+1} and $\overline{\alpha}^1_i$ is not seen). We now denote by s_1 this shared value. Now, to see the left pocket $\mathcal{P}(c^1_{s_1})$ and the right pocket $\mathcal{P}(d^1_{s_1-1})$ (that should still be seen), the third guard should be to the left of $\ell(c^1_{s_1}, \beta^1_{s_1})$ and to the right of $\ell(d^1_{s_1-1}, \beta^1_{s_1})$ (see shaded area of Figure 4). That is, the third guard should be on a region in which $\beta^1_{s_1}$ is the uppermost point. The same argument with the pockets of $\mathcal{P}_{1,\overline{\alpha},\beta}$ implies that the third guard should also be on a region in which $\beta^1_{s_1}$ is the lowermost point. Thus, the position of the third guard has to be point $\beta^1_{s_1}$. Therefore, one should put guards on points $\alpha^1_{s_1}$, $\overline{\alpha}^1_{s_1}$, and $\beta^1_{s_1}$, for some $\alpha_1 \in [t]$.

As none of those three points see any pocket $\mathcal{P}_{j,\overline{\alpha},\beta}$ with $j > 1$ (we already mentioned that no pocket of $\mathcal{P}_{j,\alpha,\beta}$ and $\mathcal{P}_{j,\alpha,\overline{\alpha}}$ with $j > 1$ can be seen by those points), we can repeat the argument for the second color class; and so forth up to color class k. Thus, a potential solution with $3k$ guards should be of the form $\{\alpha^1_{s_1}, \overline{\alpha}^1_{s_1}, \beta^1_{s_1}, \ldots, \alpha^k_{s_k}, \overline{\alpha}^k_{s_k}, \beta^k_{s_k}\}$. As there is no solution to $\mathcal{I}$, there should be a set in $\mathcal{S}_A \cup \mathcal{S}_B$ that is not hit by $\{(a^1_{s_1}, b^1_{s_1}), \ldots, (a^k_{s_k}, b^k_{s_k})\}$. By construction, the pocket associated to this set is not entirely seen. ◀

4 Parameterized hardness of the vertex guard variant

We now turn to the vertex guard variant and show the same hardness result. Again, we reduce from Structured 2-Track Hitting Set and our main task is to design a *linker gadget*. Though, *linking* pairs of vertices turns out to be very different from *linking* pairs of points. Therefore, we have to come up with fresh ideas to carry out the reduction. In a nutshell, the principal ingredient is to *link* pairs of convex vertices by introducing reflex vertices at strategic places. As placing guards on those reflex vertices is not supposed to happen in the Structured 2-Track Hitting Set instance, we design a so-called *filter gadget* to prevent any solution from doing so.

▶ **Theorem 2** (Parameterized hardness vertex guard). *Assuming the ETH,* Vertex Guard Art Gallery *is not solvable in time $f(k)\, n^{o(k/\log k)}$, for any computable function f, even on simple polygons, where n is the number of vertices of the polygon and k is the number of guards allowed.*

Proof. From an instance $\mathcal{I} = (k \in \mathbb{N}, t \in \mathbb{N}, \sigma \in \mathfrak{S}_k, \sigma_1 \in \mathfrak{S}_t, \ldots, \sigma_k \in \mathfrak{S}_t, \mathcal{S}_A, \mathcal{S}_B)$, we build

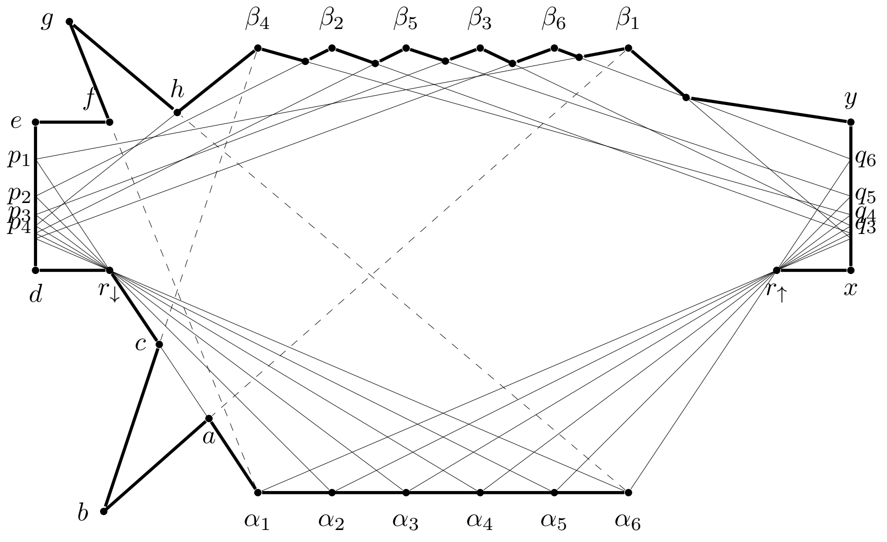


Figure 7 Vertex linker gadget. We omitted the superscript j in all the labels. Here, $\sigma_j(1) = 4,\ \sigma_j(2) = 2,\ \sigma_j(3) = 5,\ \sigma_j(4) = 3,\ \sigma_j(5) = 6,\ \sigma_j(6) = 1$.

a simple polygon $\mathcal{P}$ with $O(kt + |\mathcal{S}_A| + |\mathcal{S}_B|)$ vertices, such that $\mathcal{I}$ is a YES-instance iff $\mathcal{P}$ can be guarded by $3k$ vertices.

Linker gadget. For each $j \in [k]$, permutation σ_j will be encoded by a sub-polygon $\mathcal{P}_j$ that we call *vertex linker*, or simply *linker* (see Figure 7). We regularly set t consecutive vertices $\alpha_1^j, \alpha_2^j, \ldots, \alpha_t^j$ in this order, along the x-axis. Opposite to this *segment*, we place t vertices $\beta^j_{\sigma_j(1)}, \beta^j_{\sigma_j(2)}, \ldots, \beta^j_{\sigma_j(t)}$ in this order, along the x-axis, too. The $\beta^j_{\sigma_j(1)}, \ldots, \beta^j_{\sigma_j(t)}$, contrary to $\alpha_1^j, \ldots, \alpha_t^j$, are *not* consecutive; we will later add some reflex vertices between them. At mid-distance between α_1^j and $\beta^j_{\sigma_j(1)}$, to the left, we put a reflex vertex $r^j_\downarrow$. *Behind* this reflex vertex, we place a vertical *wall* d^je^j ($r^j_\downarrow$, d^j, and e^j are three consecutive vertices of $\mathcal{P}$), so that ray$(\alpha_1^j, r^j_\downarrow)$ and ray$(\alpha_t^j, r^j_\downarrow)$ both intersect seg(d^j, e^j). That implies that for each $i \in [t]$, ray$(\alpha_i^j, r^j_\downarrow)$ intersects seg(d^j, e^j). We denote by p_i^j this intersection. The greater i, the closer p_i^j is to d^j. Similarly, at mid-distance between α_t^j and $\beta^j_{\sigma_j(t)}$, to the right, we put a reflex vertex $r^j_\uparrow$ and place a vertical wall x^jy^j ($r^j_\uparrow$, x^j, and y^j are consecutive), so that ray$(\alpha_1^j, r^j_\uparrow)$ and ray$(\alpha_t^j, r^j_\uparrow)$ both intersect seg(x^j, y^j). For each $i \in [t]$, we denote by q_i^j the intersection between ray$(\alpha_i^j, r^j_\uparrow)$ and seg(x^j, y^j). The smaller i, the closer q_i^j is to x^j.

For each $i \in [t]$, we put around β_i^j two reflex vertices, one in ray(β_i^j, p_i^j) and one in ray(β_i^j, q_i^j). In Figure 7, we merged some reflex vertices but the essential part is that $V(\beta_i^j) \cap \text{seg}(d^j, e^j) = \text{seg}(d^j, p_i^j)$ and $V(\beta_i^j) \cap \text{seg}(x^j, y^j) = \text{seg}(x^j, q_i^j)$. Finally, we add a triangular pocket rooted at g^j and supported by ray(g^j, α_1^j) and ray(g^j, α_t^j), as well as a triangular pocket rooted at b^j and supported by ray$(g^j, \beta^j_{\sigma_j(1)})$ and ray$(g^j, \beta^j_{\sigma_j(t)})$. This ends the description of the vertex linker (see Figure 7).

The following lemma formalizes how exactly the vertices α_i^j and β_i^j are linked: say, one chooses to put a guard on a vertex α_i^j, then the only way to see entirely $\mathcal{P}_j$ by putting a second guard on a vertex of $\{\beta_1^j, \ldots, \beta_t^j\}$ is to place it on the vertex β_i^j.

▶ **Lemma 7.** *For any $j \in [k]$, the sub-polygon $\mathcal{P}_j$ is seen entirely by $\{\alpha_v^j, \beta_w^j\}$ iff $v = w$.*

Proof. The regions of $\mathcal{P}_j$ not seen by α^j_v (i.e., $\mathcal{P}_j \setminus V(\alpha^j_v)$) consist of the triangles $d^j r^j_{\downarrow} p^j_v$, $x^j r^j_{\uparrow} q^j_v$ and partially the triangle $a^j b^j c^j$. The triangle $a^j b^j c^j$ is anyway entirely seen by the vertex β^j_i, for any $i \in [t]$. It remains to prove that $d^j r^j_{\downarrow} p^j_v \cup x^j r^j_{\uparrow} q^j_v \subseteq V(\beta^j_w)$ iff $v = w$.

It holds that $d^j r^j_{\downarrow} p^j_v \cup x^j r^j_{\uparrow} q^j_v \subseteq V(\beta^j_v)$ since, by construction, the two reflex vertices neighboring β^j_v are such that β^j_v sees $\text{seg}(d^j, p^j_\alpha)$ (hence, the whole triangle $d^j r^j_{\downarrow} p^j_v$) and $\text{seg}(x^j, q^j_\alpha)$ (hence, the whole triangle $x^j r^j_{\uparrow} q^j_v$). Now, let us assume that $v \neq w$. If $v < w$, the interior of the segment $\text{seg}(p_v, p_w)$ is not seen by $\{\alpha^j_v, \beta^j_w\}$, and if $v > w$, the interior of the segment $\text{seg}(q_v, q_w)$ is not seen by $\{\alpha^j_v, \beta^j_w\}$. ◀

The issue we now have is that one could decide to place a guard on a vertex α^j_i and a second guard on a reflex vertex between $\beta^j_{\sigma_j(w)}$ and $\beta^j_{\sigma_j(w+1)}$ (for some $w \in [t-1]$). This is indeed another way to guard the whole $\mathcal{P}_j$. We will now describe a sub-polygon $\mathcal{F}_j$ (for each $j \in [k]$) called *filter gadget* (see Figure 8) satisfying the property that all its (triangular) pockets can be guarded by adding only one guard on a vertex of $\mathcal{F}_j$ iff there is already a guard on a vertex β^j_i of $\mathcal{P}_j$. Therefore, the filter gadget will prevent one from placing a guard on a reflex vertex of $\mathcal{P}_j$. The functioning of the gadget is again based on Lemma 5.

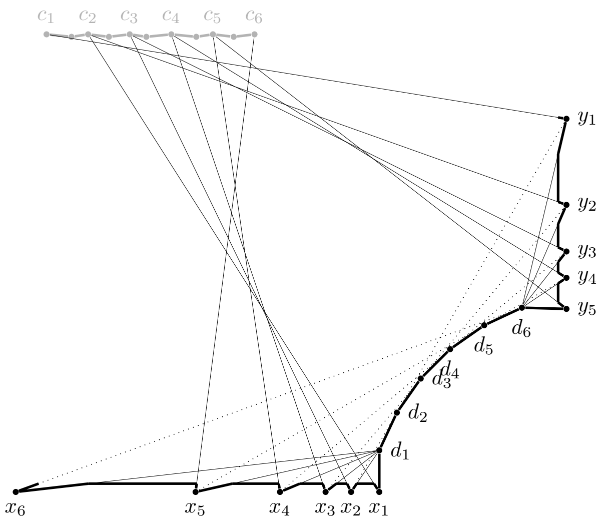
Filter gadget. Let $d^j_1, \ldots, d^j_t$ be t consecutive vertices of a regular, say, $20t$-gon, so that the angle made by $\text{ray}(d^j_1, d^j_2)$ and the x-axis is a bit below 45°, while the angle made by $\text{ray}(d^j_{t-1}, d^j_t)$ and the x-axis is a bit above 45°. The vertices $d^j_1, \ldots, d^j_t$ can therefore be seen as the discretization of an arc $\mathcal{C}$. We now mentally draw two lines ℓ_h and ℓ_v; ℓ_h is a horizontal line a bit below d^j_1, while ℓ_v is a vertical line a bit to the right of d^j_t. We put, for each $i \in [t]$, a vertex x^j_i at the intersection of ℓ_h and the tangent to $\mathcal{C}$ passing through d^j_i. Then, for each $i \in [t-1]$, we set a triangular pocket $\mathcal{P}(x^j_i)$ rooted at x^j_i and supported by $\text{ray}(x^j_i, d^j_1)$ and $\text{ray}(x^j_i, \beta^j_{\sigma_j(i+1)})$. For convenience, each point $\beta^j_{\sigma_j(i)}$ is denoted by c^j_i on Figure 8. We also set a triangular pocket $\mathcal{P}(x^j_t)$ rooted at x^j_t and supported by $\text{ray}(x^j_t, d^j_1)$ and $\text{ray}(x^j_t, d^j_t)$. Similarly, we place, for each $i \in [t-1]$, a vertex y^j_i at the intersection of ℓ_v and the tangent to $\mathcal{C}$ passing through d^j_{i+1}. Finally, we set a triangular pocket $\mathcal{P}(y^j_i)$ rooted at y^j_i and supported by $\text{ray}(y^j_i, \beta^j_{\sigma_j(i)})$ and $\text{ray}(y^j_i, d^j_t)$, for each $i \in [t-1]$ (see Figure 8). We denote by $\mathcal{P}(\mathcal{F}_j)$ the $2t-1$ triangular pockets of $\mathcal{F}_j$.

▶ **Lemma 8.** *For each $j \in [k]$, the only ways to see entirely $\mathcal{P}(\mathcal{F}_j)$ and the triangle $a^j b^j c^j$ with only two guards on vertices of $\mathcal{P}_j \cup \mathcal{F}_j$ is to place them on vertices c^j_i and d^j_i (for any $i \in [t]$).*

Proof. Proving this lemma will, in particular, entail that it is not possible to see entirely $\mathcal{P}(\mathcal{F}_j)$ with only two vertices if one of them is a reflex vertex between c^j_i and c^j_{i+1}. Let us call such a vertex an intermediate reflex vertex (in color class j). Because of the pocket $a^j b^j c^j$, one should put a guard on a c^j_i (for some $i \in [t]$) or on an intermediate reflex vertex in class j. As vertices a^j, b^j, and c^j do not see anything of $\mathcal{P}(\mathcal{F}_j)$, placing the first guard at one of those three vertices cannot work as a consequence of what follows.

Say, the first guard is placed at c^j_i $(= \beta^j_{\sigma(i)})$. The pockets $\mathcal{P}(x^j_1), \mathcal{P}(x^j_2), \ldots, \mathcal{P}(x^j_{i-1})$ and $\mathcal{P}(y^j_i), \mathcal{P}(y^j_{i+1}), \ldots, \mathcal{P}(x^j_{t-1})$ are entirely seen, while the vertices $x^j_i, x^j_{i+1}, \ldots, x^j_t$ and $y^j_1, y^j_2, \ldots, y^j_{i-1}$ are not. The only vertex that sees simultaneously all those vertices is d^j_i. The vertex d^j_i even sees the *whole* pockets $\mathcal{P}(x^j_i), \mathcal{P}(x^j_{i+1}), \ldots, \mathcal{P}(x^j_t)$ and $\mathcal{P}(y^j_1), \mathcal{P}(y^j_2), \ldots,$ $\mathcal{P}(y^j_{i-1})$. Therefore, all the pockets $\mathcal{P}(\mathcal{F}_j)$ are fully seen.

Now, say, the first guard is put on an intermediate reflex vertex r between c^j_i and c^j_{i+1} (for some $i \in [t-1]$). Both vertices x^j_i and y^j_i, as well as x^j_t, are not seen by r and should



c_1 c_2 c_3 c_4 c_5 c_6
y_1
y_2
y_3
y_4
y_5
d_6
d_5
d_4
d_3
d_2
d_1
x_6 x_5 x_4 x_3 x_2 x_1

Figure 8 The filter gadget $\mathcal{F}_j$. Again, we omit the superscript j on the labels. Vertices $c_1, c_2, \dots, c_t$ are not part of $\mathcal{F}_j$ and are in fact the vertices $\beta^j_{\sigma_j(1)}, \beta^j_{\sigma_j(2)}, \dots, \beta^j_{\sigma_j(t)}$ and the vertices in between the c_i's are the reflex vertices that we have to *filter out*.

therefore be seen by the second guard. However, no vertex simultaneously sees those three vertices. ◀

Putting the pieces together. The permutation σ is encoded the following way. We position the vertex linkers $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_k$ such that $\mathcal{P}_{i+1}$ is below and slightly to the left of $\mathcal{P}_i$. Far below and to the right of the $\mathcal{P}_i$'s, we place the $\mathcal{F}_i$'s such that the uppermost vertex of $\mathcal{F}_{\sigma(i)}$ is close and connected to the leftmost vertex of $\mathcal{F}_{\sigma(i+1)}$, for all $i \in [t-1]$. We add a constant number of vertices in the vicinity of each $\mathcal{P}_j$, so that the only filter gadget that vertices $\beta^j_1, \dots, \beta^j_t$ can see is $\mathcal{F}_{\sigma(j)}$ (see Figure 9). Similarly to the point guard version, we place vertically and far from the α^j_i's, one triangular pocket $\mathcal{P}(z_{A,q})$ rooted at vertex $z_{A,q}$ and supported by ray$(z_{A,q}, \alpha^j_i)$ and ray$(z_{A,q}, \alpha^{j'}_{i'})$, for each A-interval $I_q = [a^j_i, a^{j'}_{i'}] \in \mathcal{S}_A$ (Track A). Finally, we place vertically and far from the d^j_i's, one triangular pocket $\mathcal{P}(z_{B,q})$ rooted at vertex $z_{B,q}$ and supported by ray$(z_{B,q}, d^j_i)$ and ray$(z_{B,q}, d^{j'}_{i'})$, for each B-interval $I_q = [b^j_{\sigma_j(i)}, b^{j'}_{\sigma_{j'}(i')}] \in \mathcal{S}_B$ (Track B). This ends the construction (see Figure 9).

Soundness. We now prove the correctness of the reduction. Assume that $\mathcal{I}$ is a YES-instance and let $\{(a^1_{s_1}, b^1_{s_1}), \dots, (a^k_{s_k}, b^k_{s_k})\}$ be a solution. We claim that the set of vertices $G = \{\alpha^1_{s_1}, \beta^1_{s_1}, d^1_{\sigma_1^{-1}(s_1)}, \dots, \alpha^k_{s_k}, \beta^k_{s_k}, d^k_{\sigma_k^{-1}(s_k)}\}$ guards the whole polygon $\mathcal{P}$. Let $z^j := d^j_{\sigma_j^{-1}(s_j)}$ for notational convenience. By Lemma 7, for each $j \in [k]$, the sub-polygon $\mathcal{P}_j$ is entirely seen, since there are guards on $\alpha^j_{s_j}$ and $\beta^j_{s_j}$. By Lemma 8, for each $j \in [k]$, all the pockets of $\mathcal{F}_j$ are entirely seen, since there are guards on $\beta^j_{s_j} = c^j_{\sigma_j^{-1}(s_j)}$ and $d^j_{\sigma_j^{-1}(s_j)} = z^j$. For each A-interval (resp. B-interval) in $\mathcal{S}_A$ (resp. $\mathcal{S}_B$) there is at least one 2-element $(a^j_{s_j}, b^j_{s_j})$ such that $a^j_{s_j} \in \mathcal{S}_A$ (resp. $b^j_{s_j} \in \mathcal{S}_B$). Thus, the corresponding pocket is guarded by $\alpha^j_{s_j}$ (resp. $\beta^j_{s_j}$). The rest of the polygon is seen by, for instance, $z^{\sigma(1)}$ and $z^{\sigma(k)}$.

Assume now that there is no solution to the instance $\mathcal{I}$ of Structured 2-Track Hitting Set, and, for the sake of contradiction, that there is a set G of $3k$ vertices guarding $\mathcal{P}$. For each $j \in [k]$, vertices b^j, g^j, and x^j_t are seen by three disjoint set of vertices. The first

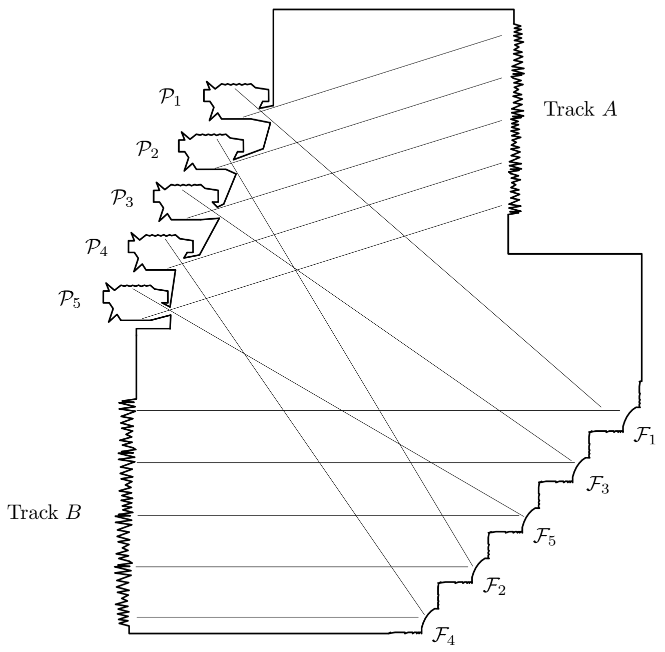


Figure 9 Overall picture of the reduction with $k = 5$.

two sets are contained in the vertices of sub-polygon $\mathcal{P}_j$ and the third one is contained in the vertices of $\mathcal{F}_j$. Therefore, to see entirely $\mathcal{P}_j \cup \mathcal{P}(\mathcal{F}_j)$, three vertices are necessary. Summing that over the k color classes, this corresponds already to $3k$ vertices which is the size of the supposed set G. Thus, there should *exactly* 3 guards placed among the vertices of $\mathcal{P}_j \cup \mathcal{F}_j$. Therefore, by Lemma 8, there should be an $s_j \in [t]$ such that both $d^j_{s_j}$ and $c^j_{s_j} = \beta^j_{\sigma_j(s_j)}$ are in G. Then, by Lemma 7, a guard should be placed at vertex $\alpha^j_{\sigma_j(s_j)}$. Indeed, the only vertices seeing g^j are f^j, g^j, h^j and $a^j_1, \dots, a^j_t$; but, if the third guard is placed at vertex f^j, g^j, or h^j, then vertices β^j_w (with $w \neq \sigma_j(i)$) are not seen. So far, we showed that G should be of the form $\{\alpha^1_{\sigma_1(s_1)}, \beta^1_{\sigma_1(s_1)}, d^1_{s_1}, \dots, \alpha^j_{\sigma_j(s_j)}, \beta^j_{\sigma_j(s_j)}, d^j_{s_j}, \dots, \alpha^k_{\sigma_k(s_k)}, \beta^k_{\sigma_k(s_k)}, d^k_{s_k}, \}$. Though, as there is no solution to $\mathcal{I}$, there should be a set in $\mathcal{S}_A \cup \mathcal{S}_B$ that is not hit by $\{(a^1_{\sigma_1(s_1)}, b^1_{\sigma_1(s_1)}), \dots, (a^k_{\sigma_k(s_k)}, b^k_{\sigma_k(s_k)})\}$. By construction, the pocket associated to this set is not entirely seen; a contradiction.

Let us bound the number of vertices of $\mathcal{P}$. Each sub-polygon $\mathcal{P}_j$ or $\mathcal{F}_j$ contains $O(t)$ vertices. *Track A* contains $3|\mathcal{S}_A|$ vertices and *Track B* contains $3|\mathcal{S}_B|$ vertices. Linking everything together requires $O(k)$ additional vertices. So, in total, there are $O(kt+|\mathcal{S}_A|+|\mathcal{S}_B|)$ vertices. Thus, this reduction together with Theorem 4 implies that Vertex Guard Art Gallery is W[1]-hard and cannot be solved in time $f(k)\, n^{o(k/\log k)}$ for any computable function f, where n is the number of vertices of the polygon and k the number of guards, unless the ETH fails. ◀

Acknowledgment. We thank Saeed Mehrabi and Meirav Zehavi for fruitful discussions.

References

1 Jochen Alber and Jirí Fiala. Geometric separation and exact solutions for the parameterized independent set problem on disk graphs. *J. Algorithms*, 52(2):134–151, 2004. `doi:10.1016/j.jalgor.2003.10.001`.

2 Saugata Basu, Richard Pollack, and Marie-Francoise Roy. Algorithms in real algebraic geometry. *AMC*, 10:12, 2011.

3 Édouard Bonnet and Tillmann Miltzow. The parameterized hardness of the art gallery problem. *CoRR*, abs/1603.08116, 2016. URL: `http://arxiv.org/abs/1603.08116`.

4 Björn Brodén, Mikael Hammar, and Bengt J. Nilsson. Guarding lines and 2-link polygons is apx-hard. In *Proceedings of the 13th Canadian Conference on Computational Geometry, University of Waterloo, Ontario, Canada, August 13-15, 2001*, pages 45–48, 2001. URL: `https://dspace.mah.se/handle/2043/6645`.

5 Sergio Cabello, Panos Giannopoulos, Christian Knauer, Dániel Marx, and Günter Rote. Geometric clustering: Fixed-parameter tractability and lower bounds with respect to the dimension. *ACM Trans. Algorithms*, 7(4):43, 2011. `doi:10.1145/2000807.2000811`.

6 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`.

7 Mark de Berg, Hans Bodlaender, and Sándor Kisfaludi-Bak. Connected dominating set in unit-disk graphs is w[1]-hard. In *EuroCG 2016*, 2016.

8 Pedro Jussieu de Rezende, Cid C. de Souza, Stephan Friedrichs, Michael Hemmer, Alexander Kröller, and Davi C. Tozoni. Engineering art galleries. *CoRR*, abs/1410.8720, 2014. URL: `http://arxiv.org/abs/1410.8720`.

9 Rodney G. Downey and Michael R. Fellows. *Parameterized complexity*. Springer Science & Business Media, 2012.

10 Alon Efrat and Sariel Har-Peled. Guarding galleries and terrains. *Inf. Process. Lett.*, 100(6):238–245, 2006. `doi:10.1016/j.ipl.2006.05.014`.

11 Stephan Eidenbenz, Christoph Stamm, and Peter Widmayer. Inapproximability results for guarding polygons and terrains. *Algorithmica*, 31(1):79–113, 2001.

12 Steve Fisk. A short proof of Chvátal's watchman theorem. *J. Comb. Theory, Ser. B*, 24(3):374, 1978. `doi:10.1016/0095-8956(78)90059-X`.

13 Jörg Flum and Martin Grohe. Parameterized complexity theory, volume xiv of texts in theoretical computer science. an EATCS series, 2006.

14 Subir K. Ghosh. *Visibility algorithms in the plane*. Cambridge University Press, 2007.

15 Subir K. Ghosh. Approximation algorithms for art gallery problems in polygons. *Discrete Applied Mathematics*, 158(6):718–722, 2010.

16 Panos Giannopoulos and Christian Knauer. Finding a largest empty convex subset in space is w[1]-hard. *CoRR*, abs/1304.0247, 2013. URL: `http://arxiv.org/abs/1304.0247`.

17 Panos Giannopoulos, Christian Knauer, and Günter Rote. The parameterized complexity of some geometric problems in unbounded dimension. In *Parameterized and Exact Computation, 4th International Workshop, IWPEC 2009, Copenhagen, Denmark, September 10-11, 2009, Revised Selected Papers*, pages 198–209, 2009. `doi:10.1007/978-3-642-11269-0_16`.

18 Panos Giannopoulos, Christian Knauer, and Sue Whitesides. Parameterized complexity of geometric problems. *Comput. J.*, 51(3):372–384, 2008. `doi:10.1093/comjnl/bxm053`.

19 Matt Gibson, Erik Krohn, and Qing Wang. The VC-dimension of visibility on the boundary of a simple polygon. In *Algorithms and Computation*, pages 541–551. Springer, 2015.

20 Alexander Gilbers and Rolf Klein. A new upper bound for the VC-dimension of visibility regions. *Computational Geometry*, 47(1):61–74, 2014.

21 Russell Impagliazzo and Ramamohan Paturi. Complexity of k-sat. In *Computational Complexity, 1999. Proceedings. Fourteenth Annual IEEE Conference on*, pages 237–240. IEEE, 1999.

22 Gil Kalai and Jiří Matoušek. Guarding galleries where every point sees a large area. *Israel Journal of Mathematics*, 101(1):125–139, 1997.

23 Matthew J. Katz and Gabriel S. Roisman. On guarding the vertices of rectilinear domains. *Computational Geometry*, 39(3):219–228, 2008.

24 James King. Fast vertex guarding for polygons with and without holes. *Comput. Geom.*, 46(3):219–231, 2013. `doi:10.1016/j.comgeo.2012.07.004`.

25 David G. Kirkpatrick. An $O(lglgOPT)$-approximation algorithm for multi-guarding galleries. *Discrete & Computational Geometry*, 53(2):327–343, 2015. `doi:10.1007/s00454-014-9656-8`.

26 Erik Krohn and Bengt J. Nilsson. Approximate guarding of monotone and rectilinear polygons. *Algorithmica*, 66(3):564–594, 2013. `doi:10.1007/s00453-012-9653-3`.

27 Dániel Marx. Parameterized complexity of independence and domination on geometric graphs. In *Parameterized and Exact Computation, Second International Workshop, IWPEC 2006, Zürich, Switzerland, September 13-15, 2006, Proceedings*, pages 154–165, 2006. `doi:10.1007/11847250_14`.

28 Dániel Marx. Can you beat treewidth? *Theory of Computing*, 6(1):85–112, 2010. `doi:10.4086/toc.2010.v006a005`.

29 Dániel Marx and Michal Pilipczuk. Optimal parameterized algorithms for planar facility location problems using Voronoi diagrams. In *ESA 2015*, pages 865–877, 2015. `doi:10.1007/978-3-662-48350-3_72`.

30 Rajeev Motwani, Arvind Raghunathan, and Huzur Saran. Covering orthogonal polygons with star polygons: The perfect graph approach. *J. Comput. Syst. Sci.*, 40(1):19–48, 1990. `doi:10.1016/0022-0000(90)90017-F`.

31 Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford University Press, 2006.

32 Joseph O'Rourke. *Art gallery theorems and algorithms*, volume 57. Oxford University Press Oxford, 1987.

33 Thomas C. Shermer. Recent results in art galleries [geometry]. *Proceedings of the IEEE*, 80(9):1384–1399, 1992.

34 Jorge Urrutia et al. Art gallery and illumination problems. *Handbook of computational geometry*, 1(1):973–1027, 2000.

35 Pavel Valtr. Guarding galleries where no point sees a small area. *Israel Journal of Mathematics*, 104(1):1–16, 1998.

KADABRA is an ADaptive Algorithm for Betweenness via Random Approximation*

Michele Borassi[1] **and Emanuele Natale**[2]

1 **IMT Insitute for Advanced Studies, 55100 Lucca, Italy**
`michele.borassi@imtlucca.it`

2 **Sapienza University of Rome, 00185 Roma, Italy**
`natale@di.uniroma1.it`

Abstract

We present KADABRA, a new algorithm to approximate betweenness centrality in directed and undirected graphs, which significantly outperforms all previous approaches on real-world complex networks. The efficiency of the new algorithm relies on two new theoretical contributions, of independent interest.

The first contribution focuses on sampling shortest paths, a subroutine used by most algorithms that approximate betweenness centrality. We show that, on realistic random graph models, we can perform this task in time $|E|^{\frac{1}{2}+o(1)}$ with high probability, obtaining a significant speedup with respect to the $\Theta(|E|)$ worst-case performance. We experimentally show that this new technique achieves similar speedups on real-world complex networks, as well.

The second contribution is a new rigorous application of the adaptive sampling technique. This approach decreases the total number of shortest paths that need to be sampled to compute all betweenness centralities with a given absolute error, and it also handles more general problems, such as computing the k most central nodes. Furthermore, our analysis is general, and it might be extended to other settings, as well.

1998 ACM Subject Classification G.2.2 [Discrete Mathematics] Graph Theory, Graph algorithms

Keywords and phrases Betweenness centrality, shortest path algorithm, graph mining, sampling, network analysis

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.20

1 Introduction

In this work we focus on estimating the *betweenness centrality*, which is one of the most famous measures of *centrality* for nodes and edges of real-world complex networks [19, 30]. The rigorous definition of betweenness centrality has its roots in sociology, dating back to the Seventies, when Freeman formalized the informal concept discussed in the previous decades in different scientific communities [6, 40, 39, 17, 13], although the definition already appeared in [4]. Since then, this notion has been very successful in network science [44, 31, 22, 30].

A probabilistic way to define the betweenness centrality[1] $\mathrm{bc}(v)$ of a node v in a graph $G = (V, E)$ is the following. We choose two nodes s and t, and we go from s to t through a shortest path π; if the choices of s, t and π are made uniformly at random, the betweenness centrality of a node v is the probability that we pass through v.

* This work was done while the authors were visiting the Simons Institute for the Theory of Computing.
[1] As explained in Section 2, to simplify notation we consider the *normalized* betweenness centrality.

In a seminal paper [14], Brandes showed that it is possible to exactly compute the betweenness centrality of all the nodes in a graph in time $\mathcal{O}(mn)$, where n is the number of nodes and m is the number of edges. A corresponding lower bound was proved in [10]: if we are able to compute the betweenness centrality of a single node in time $\mathcal{O}(mn^{1-\epsilon})$ for some $\epsilon > 0$, then the Strong Exponential Time Hypothesis [23] is false.

This result further motivates the rich line of research on computing approximations of betweenness centrality, with the goal of trading precision with efficiency. The main idea is to define a probability distribution over the set of all paths, by choosing two uniformly random nodes s, t, and then a uniformly distributed st-path $\boldsymbol{\pi}$, so that $\Pr(v \in \boldsymbol{\pi}) = \mathrm{bc}(v)$. As a consequence, we can approximate $\mathrm{bc}(v)$ by sampling paths $\boldsymbol{\pi}_1, \dots, \boldsymbol{\pi}_\tau$ according to this distribution, and estimating $\tilde{\boldsymbol{b}}(v) := \frac{1}{\tau}\sum_{i=1}^{\tau} \boldsymbol{X}_i(v)$, where $\boldsymbol{X}_i(v) = 1$ if $v \in \boldsymbol{\pi}_i$ (and $v \neq s, t$), 0 otherwise.

The tricky part of this approach is to provide probabilistic guarantees on the quality of this approximation: the goal is to obtain a $1 - \delta$ confidence interval $\boldsymbol{I}(v) = [\tilde{\boldsymbol{b}}(v) - \lambda_L, \tilde{\boldsymbol{b}}(v) + \lambda_U]$ for $\mathrm{bc}(v)$, which means that $\Pr(\forall v \in V, \mathrm{bc}(v) \in \boldsymbol{I}(v)) \geq 1 - \delta$. Thus, the research for approximating betweenness centrality has been focusing on obtaining, as fast as possible, the smallest possible $\boldsymbol{I}$.

Our Contribution

In this work, we propose a new and faster algorithm to approximate betweenness centrality in directed and undirected graphs, named KADABRA. In the standard task of approximating betweenness centralities with absolute error at most λ, we show that, on average, the new algorithm is more than 100 times faster than the previous ones, on graphs with approximately 10 000 nodes. Moreover, differently from previous approaches, our algorithm can perform more general tasks, since it does not need all confidence intervals to be equal. As an example, we consider the computation of the k most central nodes: all previous approaches compute all centralities with an error λ, and use this approximation to obtain the ranking. Conversely, our approach allows us to use small confidence interval only when they are needed, and allows bigger confidence intervals for nodes whose centrality values are "well separated". This way, we can compute for the first time an approximation of the k most central nodes in networks with millions of nodes and hundreds of millions of edges, like the Wikipedia citation network and the IMDB actor collaboration network.

Our results rely on two main theoretical contributions, which are interesting in their own right, since their generality naturally extends to other applications.

Balanced bidirectional breadth-first search. By leveraging on recent advanced results, we prove that, on many realistic random models of real-world complex networks, it is possible to sample a random path between two nodes s and t in time $m^{\frac{1}{2}+o(1)}$ if the degree distribution has finite second moment, or $m^{\frac{4-\beta}{2}+o(1)}$ if the degree distribution is power law with exponent $2 < \beta < 3$. The models considered are the Configuration Model [9], and all Rank-1 Inhomogeneous Random Graph models [42, Chapter 3], such as the Chung-Lu model [29], the Norros-Reittu model [32], and the Generalized Random Graph [42, Chapter 3]. Our proof techniques have the merit of adopting a unified approach that simultaneously works in all models considered. These models well represent metric properties of real-world networks [11]: indeed, our results are confirmed by practical experiments.

The algorithm used is simply a balanced bidirectional BFS (bb-BFS): we perform a BFS from each of the two endpoints s and t, in such a way that the two BFSs are likely to explore about the same number of edges, and we stop as soon as the two BFSs "touch each other". Rather surprisingly, this technique was never implemented to approximate betweenness centrality, and it is rarely used in the experimental algorithm community. Our theoretical analysis provides a clear explanation of the reason why this technique improves over the standard BFS: this means that many state-of-the-art algorithm for real-world complex networks can be improved by the bb-BFS.

Adaptive sampling made rigorous. To speed up the estimation of the betweenness centrality, previous work make use of the technique of adaptive sampling, which consists in testing during the execution of the algorithm whether some condition on the sample obtained so far has been met, and terminating the execution of the algorithm as soon as this happens. However, this technique introduces a subtle stochastic dependence between the time in which the algorithm terminates and the correctness of the given output, which previous papers claiming a formal analysis of the technique did not realize (see Section 3 for details). With an argument based on martingale theory, we provide a general analysis of such useful technique. Through this result, we do not only improve previous estimators, but we also make it possible to define more general stopping conditions, that can be decided "on the fly": this way, with little modifications, we can adapt our algorithm to perform more general tasks than previous ones.

To better illustrate the power of our techniques, we focus on the unweighted, static graphs, and to the centrality of nodes. However, our algorithm can be easily adapted to compute the centrality of edges, to handle weighted graphs and, since its core part consists merely in sampling paths, we conjecture that it may be coupled with the existing techniques in [8] to handle dynamic graphs.

Related Work

Computing Betweenness Centrality. With the recent event of big data, the major shortcoming of betweenness centrality has been the lack of efficient methods to compute it [14]. In the worst case, the best exact algorithm to compute the centrality of all the nodes is due to Brandes [14], and its time complexity is $\mathcal{O}(mn)$: the basic idea of the algorithm is to define the dependency $\delta_s(v) = \sum_{t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$, which can be computed in time $\mathcal{O}(m)$, for each $v \in V$ (we denote by $\sigma_{st}(v)$ the number of shortest paths from s to t passing through v, and by σ_{st} the number of st-shortest paths). In [10], it is also shown that Brandes algorithm is almost optimal on sparse graphs: an algorithm that computes the betweenness centrality of a single vertex in time $\mathcal{O}(mn^{1-\epsilon})$ falsifies widely believed complexity assumptions, such as the Strong Exponential Time Hypothesis [23], the Orthogonal Vector conjecture [2], or the Hitting Set conjecture [45]. Corresponding results in the dense, weighted case are available in [1]: computing the betweenness centrality exactly is as hard as computing the All Pairs Shortest Path, and computing an approximation with a given relative error is as hard as computing the diameter. For both these problems, there is no algorithm with running-time $\mathcal{O}(n^{3-\epsilon})$, for any $\epsilon > 0$. This shows that, for dense graphs, having an additive approximation rather than a multiplicative one is essential for a provably fast algorithm to exist. These negative results further motivates the already rich line of research on approaches that overcome this barrier. A first possibility is to use heuristics, that do not provide analytical guarantees on their performance [38, 21, 43]. Another line of research has defined variants of betweenness centrality, that might be easier to compute [15, 33, 18]. Finally, a third line of research has investigated approximation algorithms, which trade accuracy for speed [24, 16, 22, 26]. Our work follows the latter approach. The first approximation algorithm proposed in the literature [24] adapts Eppstein and Wang's approach for computing closeness centrality [20], using Hoeffding's inequality and the union bound technique. This way, it is possible to obtain an estimate of the betweenness centrality of every node that is correct up to an additive error λ with probability δ, by sampling $\mathcal{O}(\frac{D^2}{\lambda^2}\log\frac{n}{\delta})$ nodes, where D is the diameter of the graph. In [22], it is shown that this can lead to an overestimation. Riondato and Kornaropoulos improve this sampling-based approach by sampling single shortest paths instead of the whole dependency of a node [36], introducing the use of the VC-dimension. As a result, the number of samples is decreased to $\frac{c}{\lambda^2}(\lfloor\log_2(\mathrm{VD}-2)\rfloor+1+\log(\frac{1}{\delta}))$, where VD is the vertex diameter, that is, the minimum number of nodes in a shortest path in G (it can be different from $D+1$ if the graph is weighted). This use of the VC-dimension is further developed and generalized in [37]. Finally, many of these results were adapted to handle dynamic networks [8, 37].

Approximating the top-k betweenness centrality set. Let us order the nodes $v_1,...,v_n$ such that $\mathrm{bc}(v_1) \geq ... \geq \mathrm{bc}(v_n)$ and define $TOP(k) = \{(v_i, \mathrm{bc}(v_i)) : i \leq k\}$. In [36] and [37], the authors provide an algorithm that, for any given δ, ϵ, with probability $1-\delta$ outputs a set $\widetilde{TOP}(k) = \{(v_i, \tilde{\boldsymbol{b}}(v_i))\}$ such that: i) If $v \in TOP(k)$ then $v \in \widetilde{TOP}(k)$ and $|\mathrm{bc}(v) - \tilde{\boldsymbol{b}}(v)| \leq \epsilon\,\mathrm{bc}(v)$; ii) If $v \in \widetilde{TOP}(k)$ but $v \notin TOP(k)$ then $\tilde{\boldsymbol{b}}(v) \leq (\mathbf{b}_k - \epsilon)(1+\epsilon)$ where $\mathbf{b}_k$ is the k-th largest betweenness given by a preliminary phase of the algorithm.

Adaptive sampling. In [5, 37], the number of samples required is substantially reduced using the adaptive sampling technique introduced by Lipton and Naughton in [28, 27]. Let us clarify that, by adaptive sampling, we mean that the termination of the sampling process depends on the sample observed so far (in other cases, the same expression refers to the fact that the distribution of the new samples is a function of the previous ones [3], while the sample size is fixed in advance). Except for [34], previous approaches tacitly assume that there is little dependency between the stopping time and the correctness of the output: indeed, they prove that, for each *fixed* τ, the probability that the estimate is wrong at time τ is below δ. However, the stopping time $\boldsymbol{\tau}$ is a random variable, and in principle there might be dependency between the event $\boldsymbol{\tau} = \tau$ and the event that the estimate is correct at time τ. As for [34], they consider a specific stopping condition and their proof technique does not seem to extend to other settings. For a more thorough discussion of this issue, we defer the reader to Section 3.

Bidirectional BFS. The possibility of speeding up a breadth-first search for the shortest-path problem by performing, at the same time, a BFS from the final end-point, has been considered since the Seventies [35]. Unfortunately, because of the lack of theoretical results dealing with its efficiency, the bidirectional BFS has apparently not been considered a fundamental heuristic improvement [25]. However, in [36] (and in some public talks by M. Riondato), the bidirectional BFS was proposed as a possible way to improve the performance of betweenness centrality approximation algorithms.

Structure of the Paper

In Section 2, we describe our algorithm, and in Section 3 we discuss the main difficulty of the adaptive sampling, and the reasons why our techniques are not affected. In Section 4, we define the balanced bidirectional BFS, and we sketch the proof of its efficiency on random graphs. In Section 5, we show that our algorithm can be adapted to compute the k most central nodes. In Section 6 we experimentally show the effectiveness of our new algorithm. Finally, all our proofs are in the appendix.

2 Algorithm Overview

To simplify notation, we always consider the *normalized* betweenness centrality of a node v, which is defined by:

$$\mathrm{bc}(v) = \frac{1}{n(n-1)} \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

where σ_{st} is the number of shortest paths between s and t, and $\sigma_{st}(v)$ is the number of shortest paths between s and t that pass through v. Furthermore, to simplify the exposition, we use bold symbols to denote random variables, and light symbols to denote deterministic quantities. On the same line of previous works, our algorithm samples random paths $\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_\tau$, where $\boldsymbol{\pi}_i$ is chosen by selecting uniformly at random two nodes s, t, and then selecting uniformly at random one of the shortest paths from s to t. Then, it estimates $\mathrm{bc}(v)$ with $\tilde{\boldsymbol{b}}(v) := \frac{1}{\tau}\sum_{i=1}^{\tau} \boldsymbol{X}_i(v)$, where $\boldsymbol{X}_i(v) = 1$ if $v \in \boldsymbol{\pi}_i$, 0 otherwise. By definition of $\boldsymbol{\pi}_i$, $\mathbb{E}\left[\tilde{\boldsymbol{b}}(v)\right] = \mathrm{bc}(v)$.

The tricky part is to bound the distance between $\tilde{\boldsymbol{b}}(v)$ and its expected value. With a straightforward application of Hoeffding's inequality (see Appendix B of the full version [12]), it is possible to prove that $\Pr\left(\left|\tilde{\boldsymbol{b}}(v) - \mathrm{bc}(v)\right| \geq \lambda\right) \leq 2e^{-2\tau\lambda^2}$. A direct application of this inequality considers a union bound on all possible nodes v, obtaining $\Pr(\exists v \in V, |\tilde{\boldsymbol{b}}(v) - \mathrm{bc}(v)| \geq \lambda) \leq 2ne^{-2\tau\lambda^2}$. This means that the algorithm can safely stop as soon as $2ne^{-2\tau\lambda^2} \leq \delta$, that is, after $\tau = \frac{1}{2\lambda^2}\log(\frac{2n}{\delta})$ steps.

In order to improve this idea, we can start from the Chernoff bound (see Appendix B of the full version [12]), instead of Hoeffding inequality, obtaining that $\Pr\left(\left|\tilde{\boldsymbol{b}}(v) - \mathrm{bc}(v)\right| \geq \lambda\right) \leq 2\exp(-\frac{\tau\lambda^2}{2(\mathrm{bc}(v)+\lambda/3)})$.

If we assume the error λ to be small, this inequality is stronger than the previous one for all values of $\mathrm{bc}(v) < \frac{1}{4}$ (a condition which holds for almost all nodes, in almost all graphs considered). However, in order to apply this inequality, we have to deal with the fact that we do not know $\mathrm{bc}(v)$ in advance, and hence we do not know when to stop. Intuitively, to solve this problem, we make a "change of variable", and we rewrite the previous inequality as

$$\Pr\left(\mathrm{bc}(v) \leq \tilde{\boldsymbol{b}}(v) - f\right) \leq \delta_L^{(v)} \quad \text{and} \quad \Pr\left(\mathrm{bc}(v) \geq \tilde{\boldsymbol{b}}(v) + g\right) \leq \delta_U^{(v)}, \tag{1}$$

for some functions $f = f(\tilde{\boldsymbol{b}}(v), \delta_L^{(v)}, \tau), g = g(\tilde{\boldsymbol{b}}(v), \delta_U^{(v)}, \tau)$. Our algorithm fixes at the beginning the values $\delta_L^{(v)}, \delta_U^{(v)}$ for each node v, and, at each step, it tests if $f(\tilde{\boldsymbol{b}}(v), \delta_L^{(v)}, \tau)$ and $g(\tilde{\boldsymbol{b}}(v), \delta_U^{(v)}, \tau)$ are small enough. If this condition is satisfied, the algorithm stops. Note that this approach lets us define very general stopping conditions, that might depend on the centralities computed until now, on the single nodes, and so on.

▶ Remark. Instead of fixing the values $\delta_L^{(v)}, \delta_U^{(v)}$ at the beginning, one might want to decide them during the algorithm, depending on the outcome. However, this is not formally correct, because of dependency issues (for example, (1) does not even make sense, if $\delta_L^{(v)}, \delta_U^{(v)}$ are random). Finding a way to overcome this issue is left as a challenging open problem (more details are provided in Section 3).

In order to implement this idea, we still need to solve an issue: (1) holds for each *fixed* time τ, but the stopping time of our algorithm is a random variable $\boldsymbol{\tau}$, and there might be dependency between the value of $\boldsymbol{\tau}$ and the probability in (1). To this purpose, we use Mcdiarmid's inequality (see Appendix B of the full version [12]), that holds even if $\boldsymbol{\tau}$ is a random variable. However, to use this inequality, we need to assume that $\boldsymbol{\tau} < \omega$ for some deterministic ω: in our algorithm, we choose $\omega = \frac{c}{\lambda^2}\left(\lfloor \log_2(\mathrm{VD}-2) \rfloor + 1 + \log\left(\frac{2}{\delta}\right)\right)$, because, by the results in [36], after ω samples, the maximum error is at most λ, with probability $1-\frac{\delta}{2}$. Furthermore, also f and g should be modified, since they now depend on the value of ω. The pseudocode of the algorithm obtained is available in Algorithm 1 (as was done in previous approaches, we can easily parallelize the while loop in Line 5).

Algorithm 1: our algorithm for approximating betweenness centrality.

```
Input  : a graph $G=(V,E)$
Output : for each $v \in V$, an approximation $\tilde{\boldsymbol{b}}(v)$ of $\mathrm{bc}(v)$ such that
         $\Pr\left(\forall v, |\tilde{\boldsymbol{b}}(v) - \mathrm{bc}(v)| \leq \lambda\right) \geq 1-\delta$
1  $\omega \leftarrow \frac{c}{\lambda^2}\left(\lfloor \log_2(\mathrm{VD}-2)\rfloor + 1 + \log\left(\frac{2}{\delta}\right)\right)$;
2  $(\delta_L^{(v)}, \delta_U^{(v)}) \leftarrow$ computeDelta();
3  $\tau \leftarrow 0$;
4  foreach $v \in V$ do $\tilde{\boldsymbol{b}}(v) \leftarrow 0$;
5  while $\tau < \omega$ and not haveToStop ($\tilde{\boldsymbol{b}}, \delta_L, \delta_U, \omega, \tau$) do
6      $\boldsymbol{\pi}$ = samplePath();
7      foreach $v \in \boldsymbol{\pi}$ do $\tilde{\boldsymbol{b}}(v) \leftarrow \tilde{\boldsymbol{b}}(v) + 1$;
8      $\tau \leftarrow \tau + 1$;
9  end
10 foreach $v \in V$ do $\tilde{\boldsymbol{b}}(v) \leftarrow \tilde{\boldsymbol{b}}(v)/\tau$;
11 return $\tilde{\boldsymbol{b}}$
```

Algorithm 2: the function `computeDelta`.

```
Input  : a graph $G=(V,E)$, and two values $\lambda_L^{(v)}, \lambda_U^{(v)}$ for each $v \in V$
Output : for each $v \in V$, two values $\delta_L^{(v)}, \delta_U^{(v)}$
1  $\alpha \leftarrow \frac{\omega}{100}$;
2  $\epsilon \leftarrow 0.0001$;
3  foreach $i \in [1, \alpha]$ do
4      $\boldsymbol{\pi}$ = samplePath();
5      foreach $v \in \boldsymbol{\pi}$ do $\tilde{\boldsymbol{b}}(v) \leftarrow \tilde{\boldsymbol{b}}(v) + 1$;
6  end
7  foreach $v \in V$ do
8      $\tilde{\boldsymbol{b}}(v) \leftarrow \tilde{\boldsymbol{b}}(v)/\alpha$;
9      $c_L(v) \leftarrow \frac{2\tilde{b}(v)\omega}{(\lambda_L^{(v)})^2}$;
10     $c_U(v) \leftarrow \frac{2\tilde{b}(v)\omega}{(\lambda_U^{(v)})^2}$;
11 end
12 Binary search to find $C$ such that $\sum_{v \in V} \exp\left(-\frac{C}{c_L(v)}\right) + \exp\left(-\frac{C}{c_U(v)}\right) = \frac{\delta}{2} - \epsilon\delta$;
13 foreach $v \in V$ do
14     $\delta_L^{(v)} \leftarrow \exp\left(-\frac{C}{c_L(v)}\right) + \frac{\epsilon\delta}{2n}$;
15     $\delta_U^{(v)} \leftarrow \exp\left(-\frac{C}{c_U(v)}\right) + \frac{\epsilon\delta}{2n}$;
16 end
17 return $\boldsymbol{b}$;
```

The correctness of the algorithm follows from the following theorem, which is the base of our adaptive sampling, and which we prove in Section 2.1 (where we also define the functions f and g).

▶ **Theorem 1.** *Let $\tilde{\boldsymbol{b}}(v)$ be the output of Algorithm 1, and let $\boldsymbol{\tau}$ be the number of samples at the end of the algorithm. Then, with probability $1-\delta$, the following conditions hold:*

- *if $\boldsymbol{\tau} = \omega$, $|\tilde{\boldsymbol{b}}(v) - \mathrm{bc}(v)| < \lambda$ for all v;*
- *if $\boldsymbol{\tau} < \omega$, $-f(\boldsymbol{\tau}, \tilde{\boldsymbol{b}}(v), \delta_L^{(v)}, \omega) \leq \mathrm{bc}(v) - \tilde{\boldsymbol{b}}(v) \leq g(\boldsymbol{\tau}, \tilde{\boldsymbol{b}}(v), \delta_U^{(v)}, \omega)$ for all v.*

Algorithm 3: The function `haveToStop` to compute the top-k nodes.

```
Input   : for each node $v$, the values of $\tilde{\boldsymbol{b}}(v), \delta_L^{(v)}, \delta_U^{(v)}$, and the values of $\omega$ and $\tau$
Output : True if the algorithm should stop, False otherwise
Sort nodes in decreasing order of $\tilde{\boldsymbol{b}}(v)$, obtaining $v_1, \dots, v_n$;
for $i \in [1, \dots, k]$ do
    if $f(\tilde{\boldsymbol{b}}(v_i), \delta_L^{(v_i)}, \omega, \tau) > \lambda$ or $g(\tilde{\boldsymbol{b}}(v_i), \delta_U^{(v_i)}, \omega, \tau) > \lambda$ then
        if $\tilde{\boldsymbol{b}}(v_{i-1}) - f(\tilde{\boldsymbol{b}}(v_{i-1}), \delta_L^{(v_{i-1})}, \omega, \tau) < \tilde{\boldsymbol{b}}(v_i) + g(\tilde{\boldsymbol{b}}(v_i), \delta_U^{(v_i)}, \omega, \tau)$ or
          $\tilde{\boldsymbol{b}}(v_i) - f(\tilde{\boldsymbol{b}}(v_i), \delta_L^{(v_i)}, \omega, \tau) < \tilde{\boldsymbol{b}}(v_{i+1}) + g(\tilde{\boldsymbol{b}}(v_{i+1}), \delta_U^{(v_{i+1})}, \omega, \tau)$ then
            return False;
        end
    end
end
for $i \in [k+1, \dots, n]$ do
    if $f(\tilde{\boldsymbol{b}}(v_i), \delta_L^{(v_i)}, \omega, \tau) > \lambda$ or $g(\tilde{\boldsymbol{b}}(v_i), \delta_U^{(v_i)}, \omega, \tau) > \lambda$ then
        if $\tilde{\boldsymbol{b}}(v_k) - f(\tilde{\boldsymbol{b}}(v_k), \delta_L^{(v_k)}, \omega, \tau) < \tilde{\boldsymbol{b}}(v_i) + g(\tilde{\boldsymbol{b}}(v_i), \delta_U^{(v_i)}, \omega, \tau)$ then
            return False;
        end
    end
end
return True;
```

▶ Remark. This theorem says that, at the beginning of the algorithm, we know that, with probability $1-\delta$, one of the two conditions will hold when the algorithm stops, independently of the final value of $\boldsymbol{\tau}$. This is essential to avoid the stochastic dependence that we discuss in Section 3.

In order to apply this theorem, we choose λ such that our goal is reached if all centralities are known with error at most λ. Then, we choose the function `haveToStop` in a way that our goal is reached if the stopping condition is satisfied. This way, our algorithm is correct, both if $\boldsymbol{\tau} = \omega$ and if $\boldsymbol{\tau} < \omega$. For example, if we want to compute all centralities with bounded absolute error, we simply choose λ as the bound we want to achieve, and we plug the stopping condition $f, g \leq \lambda$ in the function `haveToStop`. Instead, if we want to compute an approximation of the k most central nodes, we need a different definition of f and g, which is provided in Section 5.

To complete the description of this algorithm, we need to specify the following functions.

`computeDelta`: The algorithm works for any choice of the $\delta_L^{(v)}, \delta_U^{(v)}$s, but a good choice yields better running times. We adopt the heuristic given in Algorithm 2, which we discuss in Appendix D of the full version.

`samplePath`: In order to sample a path between two random nodes s and t, we use a balanced bidirectional BFS (see Appendix E of the full version [12] for a detailed description).

2.1 Proof of Theorem 1

In our algorithm, we sample $\boldsymbol{\tau}$ shortest paths $\boldsymbol{\pi}_i$, where $\boldsymbol{\tau}$ is a random variable such that $\boldsymbol{\tau} = \tau$ can be decided by looking at the first τ paths sampled (see Algorithm 1). Furthermore, thanks to Eq. (3) in [36], we assume that $\boldsymbol{\tau} \leq \omega$ for some fixed $\omega \in \mathbb{R}^+$ such that, after ω steps, $\Pr(\forall v, |\tilde{\boldsymbol{b}}(v) - \mathrm{bc}(v)| \leq \lambda) \geq 1 - \frac{\delta}{2}$. When the algorithm stops, our estimate of the betweenness is $\tilde{\boldsymbol{b}}(v) := \frac{1}{\boldsymbol{\tau}} \sum_{i=1}^{\boldsymbol{\tau}} \boldsymbol{X}_i(v)$, where $\boldsymbol{X}_i(v)$ is 1 if v belongs to $\boldsymbol{\pi}_i$, 0 otherwise.

To estimate the error, we use the following theorem.

▶ **Theorem 2.** *For each node v and for every fixed real numbers δ_L, δ_U, it holds*

$$\Pr\left(\mathrm{bc}(v) \leq \tilde{\boldsymbol{b}}(v) - f\left(\tilde{\boldsymbol{b}}(v), \delta_L, \omega, \boldsymbol{\tau}\right)\right) \leq \delta_L \quad \textit{and}$$
$$\Pr\left(\mathrm{bc}(v) \geq \tilde{\boldsymbol{b}}(v) + g\left(\tilde{\boldsymbol{b}}(v), \delta_U, \omega, \boldsymbol{\tau}\right)\right) \leq \delta_U,$$

where

$$f\left(\tilde{\boldsymbol{b}}(v), \delta_L, \omega, \boldsymbol{\tau}\right) = \frac{1}{\boldsymbol{\tau}} \log \frac{1}{\delta_L} \left(\frac{1}{3} - \frac{\omega}{\boldsymbol{\tau}} + \sqrt{\left(\frac{1}{3} - \frac{\omega}{\boldsymbol{\tau}}\right)^2 + \frac{2\tilde{\boldsymbol{b}}(v)\omega}{\log \frac{1}{\delta_L}}}\right) \quad \textit{and} \tag{2}$$

$$g\left(\tilde{\boldsymbol{b}}(v), \delta_U, \omega, \boldsymbol{\tau}\right) = \frac{1}{\boldsymbol{\tau}} \log \frac{1}{\delta_U} \left(\frac{1}{3} + \frac{\omega}{\boldsymbol{\tau}} + \sqrt{\left(\frac{1}{3} + \frac{\omega}{\boldsymbol{\tau}}\right)^2 + \frac{2\tilde{\boldsymbol{b}}(v)\omega}{\log \frac{1}{\delta_U}}}\right). \tag{3}$$

Before proving Theorem 2, let us see how this theorem implies Theorem 1. To simplify notation, we often omit the arguments of the function f and g.

Proof of Theorem 1. Let $\boldsymbol{E}_1$ be the event $(\boldsymbol{\tau} = \omega \wedge \exists v \in V, |\tilde{\boldsymbol{b}}(v) - \mathrm{bc}(v)| > \lambda)$, and let $\boldsymbol{E}_2$ be the event $(\boldsymbol{\tau} < \omega \wedge (\exists v \in V, -f \geq \mathrm{bc}(v) - \tilde{\boldsymbol{b}}(v) \vee \mathrm{bc}(v) - \tilde{\boldsymbol{b}}(v) \geq g))$. Let us also denote $\tilde{\boldsymbol{b}}_\tau(v) = \frac{1}{\tau}\sum_{i=1}^{\tau} \boldsymbol{X}_i(v)$ (note that $\tilde{\boldsymbol{b}}_{\boldsymbol{\tau}}(v) = \tilde{\boldsymbol{b}}(v)$).

By our choice of ω and Eq. (3) in [36],

$$\Pr(\boldsymbol{E}_1) \leq \Pr(\exists v \in V, |\tilde{\boldsymbol{b}}_\omega(v) - \mathrm{bc}(v)| > \lambda) \leq \frac{\delta}{2}$$

where $\tilde{\boldsymbol{b}}_\omega(v)$ is the approximate betweenness of v after ω samples. Furthermore, by Theorem 2,

$$\begin{aligned}\Pr(\boldsymbol{E}_2) &\leq \sum_{v \in V} \Pr(\boldsymbol{\tau} < \omega \wedge -f \geq \mathrm{bc}(v) - \tilde{\boldsymbol{b}}(v)) + \Pr(\boldsymbol{\tau} < \omega \wedge \mathrm{bc}(v) - \tilde{\boldsymbol{b}}(v) \leq g) \\ &\leq \sum_{v \in V} \delta_L^{(v)} + \delta_U^{(v)} \leq \frac{\delta}{2}.\end{aligned}$$

By a union bound, $\Pr(\boldsymbol{E}_1 \vee \boldsymbol{E}_2) \leq \Pr(\boldsymbol{E}_1) + \Pr(\boldsymbol{E}_1) \leq \delta$, concluding the proof of Theorem 1. ◀

Thus, it remains to prove Theorem 2.

Proof of Theorem 2. Since this theorem deals with a single node v, let us simply write $\mathrm{bc} = \mathrm{bc}(v), \tilde{\boldsymbol{b}} = \tilde{\boldsymbol{b}}(v), \boldsymbol{X}_i = \boldsymbol{X}_i(v)$. Let us consider $\boldsymbol{Y}^\tau = \sum_{i=1}^{\tau}(\boldsymbol{X}_i - \mathrm{bc})$ (we recall that $\boldsymbol{X}_i = 1$ if v is in the i-th path sampled, $\boldsymbol{X}_i = 0$ otherwise). Clearly, $\boldsymbol{Y}^\tau$ is a martingale, and $\boldsymbol{\tau}$ is a stopping time for $\boldsymbol{Y}^\tau$: this means that also $\boldsymbol{Z}^\tau = \boldsymbol{Y}^{\min(\boldsymbol{\tau}, \tau)}$ is a martingale.

Let us apply Mcdiarmid's inequality (see e.g. Theorem 8 in Appendix B of the full version [12]) to the martingales $\boldsymbol{Z}$ and $-\boldsymbol{Z}$: for each fixed $\lambda_L, \lambda_U > 0$ we have

$$\Pr\left(\boldsymbol{Z}^\omega \geq \lambda_L\right) = \Pr\left(\boldsymbol{\tau}\tilde{\boldsymbol{b}} - \boldsymbol{\tau}\,\mathrm{bc} \geq \lambda_L\right) \leq \exp\left(-\frac{\lambda_L^2}{2\left(\omega\,\mathrm{bc} + \lambda_L/3\right)}\right) = \delta_L \quad \text{and} \tag{4}$$

$$\Pr\left(-\boldsymbol{Z}^\omega \geq \lambda_U\right) = \Pr\left(\boldsymbol{\tau}\tilde{\boldsymbol{b}} - \boldsymbol{\tau}\,\mathrm{bc} \leq -\lambda_U\right) \leq \exp\left(-\frac{\lambda_U^2}{2\left(\omega\,\mathrm{bc} + \lambda_U/3\right)}\right) = \delta_U. \tag{5}$$

We now show how to prove (2) from (4). The way to derive (3) from (5) is analogous.

If we express λ_L as a function of δ_L we get

$$\lambda_L^2 = 2\log\frac{1}{\delta_L}\left(\omega\,\mathrm{bc} + \frac{\lambda_L}{3}\right) \iff \lambda_L^2 - \frac{2}{3}\lambda_L \log\frac{1}{\delta_L} - 2\omega\,\mathrm{bc}\log\frac{1}{\delta_L} = 0,$$

which implies that

$$\lambda_L = \frac{1}{3}\log\frac{1}{\delta_L} \pm \sqrt{\frac{1}{9}\left(\log\frac{1}{\delta_L}\right)^2 + 2\omega\,\mathrm{bc}\log\frac{1}{\delta_L}}.$$

Since (4) holds for any positive value λ_L, it also holds for the value corresponding to the positive solution of this equation, that is,

$$\lambda_L = \frac{1}{3}\log\frac{1}{\delta_L} + \sqrt{\frac{1}{9}\left(\log\frac{1}{\delta_L}\right)^2 + 2\omega\,\mathrm{bc}\log\frac{1}{\delta_L}}.$$

Plugging this value into (4), we obtain

$$\Pr\left(\boldsymbol{\tau}\tilde{\boldsymbol{b}} - \boldsymbol{\tau}\,\mathrm{bc} \geq \frac{1}{3}\log\frac{1}{\delta_L} + \sqrt{\frac{1}{9}\left(\log\frac{1}{\delta_L}\right)^2 + 2\omega\,\mathrm{bc}\log\frac{1}{\delta_L}}\right) \leq \delta_L. \tag{6}$$

By assuming $\tilde{\boldsymbol{b}} - \mathrm{bc} \geq \frac{1}{3\boldsymbol{\tau}}\log(\frac{1}{\delta_L})$, the event in (6) can be rewritten as

$$(\boldsymbol{\tau}\,\mathrm{bc})^2 - 2\,\mathrm{bc}\left(\boldsymbol{\tau}^2\tilde{\boldsymbol{b}} + \omega\log\frac{1}{\delta_L} - \frac{1}{3}\boldsymbol{\tau}\log\frac{1}{\delta_L}\right) - \frac{2}{3}\log\frac{1}{\delta_L}\boldsymbol{\tau}\tilde{\boldsymbol{b}} + \left(\boldsymbol{\tau}\tilde{\boldsymbol{b}}\right)^2 \geq 0.$$

By solving the previous quadratic equation w.r.t. bc we get

$$\mathrm{bc} \leq \tilde{\boldsymbol{b}} + \log\frac{1}{\delta_L}\left(\frac{\omega}{\boldsymbol{\tau}^2} - \frac{1}{3\boldsymbol{\tau}} - \sqrt{\left(\frac{\tilde{\boldsymbol{b}}}{\log\frac{1}{\delta_L}} + \frac{\omega}{\boldsymbol{\tau}^2} - \frac{1}{3\boldsymbol{\tau}}\right)^2 - \left(\frac{\tilde{\boldsymbol{b}}}{\log\frac{1}{\delta_L}}\right)^2 + \frac{2}{3\boldsymbol{\tau}}\frac{\tilde{\boldsymbol{b}}}{\log\frac{1}{\delta_L}}}\right),$$

where we only considered the solution which upper bounds bc, since we assumed $\tilde{\boldsymbol{b}} - \mathrm{bc} \geq \frac{1}{3\tau}\log(\frac{1}{\delta_L})$. After simplifying the terms under the square root in the previous expression, we get

$$\mathrm{bc} \leq \tilde{\boldsymbol{b}} + \log\frac{1}{\delta_L}\left(\frac{\omega}{\boldsymbol{\tau}^2} - \frac{1}{3\boldsymbol{\tau}} - \sqrt{\left(\frac{\omega}{\boldsymbol{\tau}^2} - \frac{1}{3\boldsymbol{\tau}}\right)^2 + \frac{2\tilde{\boldsymbol{b}}\omega}{\boldsymbol{\tau}^2\log\frac{1}{\delta_L}}}\right),$$

which means that

$$\Pr\left(\mathrm{bc} \leq \tilde{\boldsymbol{b}} - f\left(\tilde{\boldsymbol{b}}, \delta_L, \omega, \boldsymbol{\tau}\right)\right) \leq \delta_L,$$

concluding the proof. ◀

3 Adaptive Sampling

In this section, we highlight the main technical difficulty in the formalization of adaptive sampling, which previous works claiming analogous results did not address. Furthermore, we sketch the way we overcome this difficulty: our argument is quite general, and it could be easily adapted to formalize these claims.

As already said, the problem is the stochastic dependence between the time $\boldsymbol{\tau}$ in which the algorithm terminates and the event $\boldsymbol{A}_\tau$ = "at time τ, the estimate is within the required distance from the true value", since both $\boldsymbol{\tau}$ and $\boldsymbol{A}_\tau$ are functions of the same random sample. Since it is typically possible to prove that $\Pr(\neg\boldsymbol{A}_\tau) \leq \delta$ for every fixed τ, one may be tempted to argue that also $\Pr(\neg \boldsymbol{A}_{\boldsymbol{\tau}}) \leq \delta$, by applying these inequalities at time $\boldsymbol{\tau}$. However, this is not correct: indeed, if we have no assumptions on $\boldsymbol{\tau}$, $\boldsymbol{\tau}$ could even be defined as the smallest τ such that $\boldsymbol{A}_{\tau}$ does not hold!

More formally, if we want to link $\Pr(\neg \boldsymbol{A}_{\boldsymbol{\tau}})$ to $\Pr(\neg \boldsymbol{A}_{\tau})$, we have to use the law of total probability, that says that:

$$\Pr(\neg \boldsymbol{A}_{\boldsymbol{\tau}}) = \sum_{\tau=1}^{\infty} \Pr(\neg \boldsymbol{A}_{\boldsymbol{\tau}} \,|\, \boldsymbol{\tau} = \tau) \Pr(\boldsymbol{\tau} = \tau) \tag{7}$$

$$= \Pr(\neg \boldsymbol{A}_{\boldsymbol{\tau}} \,|\, \boldsymbol{\tau} < \tau) \Pr(\boldsymbol{\tau} < \tau) + \Pr(\neg \boldsymbol{A}_{\boldsymbol{\tau}} \,|\, \boldsymbol{\tau} \geq \tau) \Pr(\boldsymbol{\tau} \geq \tau). \tag{8}$$

Then, if we want to bound $\Pr(\neg \boldsymbol{A}_{\boldsymbol{\tau}})$, we need to assume that

$$\Pr(\neg A_{\boldsymbol{\tau}} \,|\, \boldsymbol{\tau} = \tau) \leq \Pr(\neg A_{\tau}) \quad \text{or that} \quad \Pr(\neg A_{\tau} \,|\, \boldsymbol{\tau} \geq \tau) \leq \Pr(\neg A_{\tau}), \tag{9}$$

which would allow to bound (7) or (8) from above. The equations in (9) are implicitly assumed to be true in previous works adopting adaptive sampling techniques. Unfortunately, because of the stochastic dependence, it is quite difficult to prove such inequalities, even if some approaches managed to overcome these difficulties [34].

For this reason, our proofs avoid dealing with such relations: in the proof of Theorem 1, we fix a deterministic time ω, we impose that $\boldsymbol{\tau} \leq \omega$, and we apply the inequalities with $\tau = \omega$. Then, using martingale theory, we convert results that hold at time ω to results that hold at the stopping time $\boldsymbol{\tau}$ (see Section 2.1).

4 Balanced Bidirectional BFS

A major improvement of our algorithm, with respect to previous counterparts, is that we sample shortest paths through a balanced bidirectional BFS, instead of a standard BFS. In this section, we describe this technique, and we bound its running time on realistic models of random graphs, with high probability. The idea behind this technique is very simple: if we need to sample a uniformly random shortest path from s to t, instead of performing a full BFS from s until we reach t, we perform at the same time a BFS from s and a BFS from t, until the two BFSs touch each other (if the graph is directed, we perform a “forward” BFS from s and a “backward” BFS from t).

More formally, assume that we have visited up to level l_s from s and to level l_t from t, let $\Gamma^{l_s}(s)$ be the set of nodes at distance l_s from s, and similarly let $\Gamma^{l_t}(t)$ be the set of nodes at distance l_t from t. If $\sum_{v \in \Gamma^{l_s}(s)} \deg(v) \leq \sum_{v \in \Gamma^{l_t}(t)} \deg(v)$, we process all nodes in $\Gamma^{l_s}(s)$, otherwise we process all nodes in $\Gamma^{l_t}(t)$ (since the time needed to process level l_s is proportional to $\sum_{v \in \Gamma^{l_s}(s)} \deg(v)$, this choice minimizes the time needed to visit the next level). Assume that we are processing the node $v \in \Gamma^{l_s}(s)$ (the other case is analogous). For each neighbor w of v we do the following:

- if w was never visited, we add w to $\Gamma^{l_s+1}(s)$;
- if w was already visited in the BFS from s, we do not do anything;
- if w was visited in the BFS from t, we add the edge (v, w) to the set Π of candidate edges in the shortest path.

After we have processed a level, we stop if $\Gamma^{l_s}(s)$ or $\Gamma^{l_t}(t)$ is empty (in this case, s and t are not connected), or if Π is not empty. In the latter case, we select an edge from Π, so that the probability of choosing the edge (v, w) is proportional to $\sigma_{sv}\sigma_{wt}$ (we recall that σ_{xy} is the number of shortest paths from x to y, and it can be computed during the BFS as in [16]).

Then, the path is selected by considering the concatenation of a random path from s to v, the edge (v, w), and a random path from w to t. These random paths can be easily chosen by backtracking, as shown in [36] (since the number of paths might be exponential in the input size, in order to avoid pathological cases, we assume that we can perform arithmetic operations in $\mathcal{O}(1)$ time).

4.1 Analysis on Random Graph

In order to show the effectiveness of the balanced bidirectional BFS, we bound its running time in several models of random graphs: the Configuration Model (CM, [9]), and Rank-1 Inhomogeneous Random Graph models (IRG, [42, Chapter 3]), such as the Chung-Lu model [29], the Norros-Reittu model [32], and the Generalized Random Graph [42, Chapter 3]. In these models, we fix the number n of nodes, and we give a weight ρ_u to each node. In the CM, we create edges by giving ρ_u half-edges to each node u, and pairing these half-edges uniformly at random; in IRG we connect each pair of nodes (u, v) independently with probability close to $\rho_u\rho_v/\sum_{w\in V}\rho_w$. With some technical assumptions discussed in Appendix E of the full version [12], we prove the following theorem.

▶ **Theorem 3.** *Let G be a graph generated through the aforementioned models. Then, for each fixed $\epsilon > 0$, and for each pair of nodes s, t, w.h.p., the time needed to compute an st-shortest path through a bidirectional BFS is $\mathcal{O}(n^{\frac{1}{2}+\epsilon})$ if the degree distribution λ has finite second moment, $\mathcal{O}(n^{\frac{4-\beta}{2}} + \epsilon)$ if λ is a power law distribution with $2 < \beta < 3$.*

Sketch of proof. The idea of the proof is that the time needed by a bidirectional BFS is proportional to the number of visited edges, which is close to the sum of the degrees of the visited nodes, which are very close to their weights. Hence, we have to analyze the weights of the visited edges: for this reason, if V' is a subset of V, we define the volume of V' as $\rho_{V'} = \sum_{v\in V'}\rho_v$.

Our visit proceeds by "levels" in the BFS trees from s and t: if we never process a level with total weight at least $n^{\frac{1}{2}+\epsilon}$, since the diameter is $\mathcal{O}(\log n)$, the volume of the set of processed vertices is $\mathcal{O}(n^{\frac{1}{2}+\epsilon}\log n)$, and the number of visited edges cannot be much bigger (for example, this happens if s and t are not connected). Otherwise, assume that, at some point, we process a level l_s in the BFS from s with total weight $n^{\frac{1}{2}+\epsilon}$: then, the corresponding level l_t in the BFS from t has also weight $n^{\frac{1}{2}+\epsilon}$ (otherwise, we would have expanded from t, because weights and degrees are strongly correlated). We use the "birthday paradox": levels $l_s + 1$ in the BFS from s, and level $l_t + 1$ in the BFS from t are random sets of nodes with size close to $n^{\frac{1}{2}+\epsilon}$, and hence there is a node that is common to both, w.h.p.. This means that the time needed by the bidirectional BFS is proportional to the volume of all levels in the BFS tree from s, until l_s, plus the volume of all levels in the BFS tree from t, until l_t (note that we do not expand levels $l_s + 1$ and $l_t + 1$). All levels except the last have volume at most $n^{\frac{1}{2}+\epsilon}$, and there are $\mathcal{O}(\log n)$ such levels because the diameter is $\mathcal{O}(\log n)$: it only remains to estimate the volume of the last level.

By definition of the models, the probability that a node v with weight ρ_v belongs to the last level is about $\frac{\rho_v\rho_{\mathbf{\Gamma}^{l_s-1}(s)}}{M} \leq \rho_v n^{-\frac{1}{2}+\epsilon}$: hence, the expected volume of $\mathbf{\Gamma}^{l_s}(s)$ is at most $\sum_{v\in V}\rho_v \Pr(v \in \mathbf{\Gamma}^{l_s-1}(s)) \leq \sum_{v\in V}\rho_v^2 n^{-\frac{1}{2}+\epsilon}$. Through standard concentration inequalities, we prove that this random variable is concentrated: hence, we only need to compute this expected value. If the degree distribution has finite second moment, then $\sum_{v\in V}\rho_v^2 = \mathcal{O}(n)$, concluding the proof. If the degree distribution is power law with $2 < \beta < 3$, then we have to consider separately nodes v such that $\rho_v < n^{\frac{1}{2}}$ and such that $\rho_v > n^{\frac{1}{2}}$. In the first case, $\sum_{\rho_v<n^{\frac{1}{2}}}\rho_v^2 \approx \sum_{d=0}^{n^{\frac{1}{2}}} nd^2\lambda(d) \approx \sum_{d=0}^{n^{\frac{1}{2}}} nd^{2-\beta} \approx n^{1+\frac{3-\beta}{2}}$. In the second case, we prove that

the volume of the set of nodes with weight bigger than $n^{\frac{1}{2}}$ is at most $n^{\frac{4-\beta}{2}}$. Hence, the total volume of $\mathbf{\Gamma}^{l_s}(s)$ is at most $n^{-\frac{1}{2}+\epsilon}n^{1+\frac{3-\beta}{2}}+n^{\frac{4-\beta}{2}} \approx n^{\frac{4-\beta}{2}}$. ◀

5 Computing the k Most Central Nodes

Differently from previous works, our algorithm is more flexible, making it possible to compute the betweenness centrality of different nodes with different precision. This feature can be exploited if we only want to rank the nodes: for instance, if v is much more central than all the other nodes, we do not need a very precise estimation on the centrality of v to say that it is the top node. Following this idea, in this section we adapt our approach to the approximation of the ranking of the k most central nodes: as far as we know, this is the first approach which computes the ranking without computing a λ-approximation of all betweenness centralities, allowing significant speedups. Clearly, we cannot expect our ranking to be always correct, otherwise the algorithm does not terminate if two of the k most central nodes have the same centrality. For this reason, the user fixes a parameter λ, and, for each node v, the algorithm does one of the following:

- it provides the exact position of v in the ranking;
- it guarantees that v is not in the top-k;
- it provides a value $\tilde{\boldsymbol{b}}(v)$ such that $|\operatorname{bc}(v)-\tilde{\boldsymbol{b}}(v)| \leq \lambda$.

In other words, similarly to what is done in [36], the algorithm provides a set of $k' \geq k$ nodes containing the top-k nodes, and for each pair of nodes v, w in this subset, either we can rank correctly v and w, or v and w are almost even, that is, $|\operatorname{bc}(v)-\operatorname{bc}(w)| \leq 2\lambda$. In order to obtain this result, we plug into Algorithm 1 the aforementioned conditions in the function `haveToStop` (see Algorithm 3 in the appendix).

Then, we have to adapt the function `computeDelta` to optimize the $\delta_L^{(v)}$s and the $\delta_U^{(v)}$s to the new stopping condition: in other words, we have to choose the values of $\lambda_L^{(v)}$ and $\lambda_U^{(v)}$ that should be plugged into the function `computeDelta` (we recall that the heuristic `computeDelta` chooses the $\delta_L^{(v)}$s so that we can guarantee as fast as possible that $\tilde{\boldsymbol{b}}(v)-\lambda_L^{(v)} \leq \operatorname{bc}(v) \leq \tilde{\boldsymbol{b}}(v)+\lambda_U^{(v)}$). To this purpose, we estimate the betweenness of all nodes with few samples and we sort all nodes according to these approximate values $\tilde{b}(v)$, obtaining $v_1, \ldots, v_n$. The basic idea is that, for the first k nodes, we set $\lambda_U^{(v_i)} = \frac{\tilde{b}(v_{i-1})-\tilde{b}(v_i)}{2}$, and $\lambda_L^{(v_i)} = \frac{\tilde{b}(v_i)-\tilde{b}(v_{i+1})}{2}$ (the goal is to find confidence intervals that separate the betweenness of v_i from the betweenness of v_{i+1} and v_{i-1}). For nodes that are not in the top-k, we choose $\lambda_L^{(v)} = 1$ and $\lambda_U^{(v)} = \tilde{b}(v_k)-\lambda_L^{(v_k)}-\tilde{b}(v_i)$ (the goal is to prove that v_i is not in the top-k). Finally, if $\tilde{b}(v_i)-\tilde{b}(v_{i+1})$ is small, we simply set $\lambda_L^{(v_i)} = \lambda_U^{(v_i)} = \lambda_L^{(v_{i+1})} = \lambda_U^{(v_{i+1})} = \lambda$, because we do not know if $\operatorname{bc}(v_{i+1}) > \operatorname{bc}(v_i)$, or viceversa.

6 Experimental Results

In this section, we test the four variations of our algorithm on several real-world networks, in order to evaluate their performances. The platform for our tests is a server with 1515 GB RAM and 48 Intel(R) Xeon(R) CPU E7-8857 v2 cores at 3.00GHz, running Debian GNU Linux 8. The algorithms are implemented in C++, and they are compiled using gcc 5.3.1. The source code of our algorithm is available at `https://sites.google.com/a/imtlucca.it/borassi/publications`.

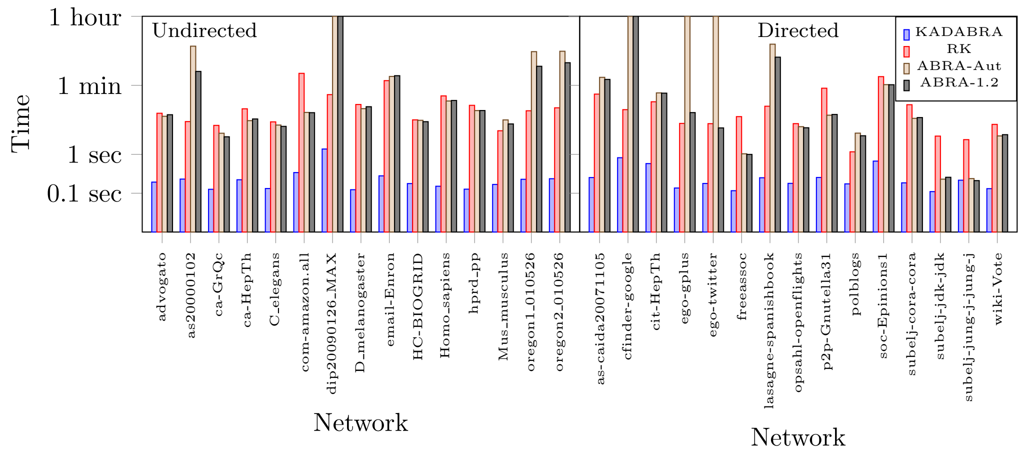


Figure 1 The time needed by the different algorithms, on all the graphs of our dataset.

Comparison with the State of the Art

The first experiment compares the performances of our algorithm KADABRA with the state of the art. The first competitor is the RK algorithm [36], available in the open-source *NetworKit* framework [41]. This algorithm uses the same estimator as our algorithm, but the stopping condition is different: it simply stops after sampling $k = \frac{c}{\epsilon^2}\left(\lfloor \log_2(\text{VD}-2)\rfloor + 1 + \log\left(\frac{1}{\delta}\right)\right)$, and it uses a heuristic to upper bound the vertex diameter. Following suggestions by the author of the *NetworKit* implementation, we set to 20 the number of samples used in the latter heuristic [7].

The second competitor is the ABRA algorithm [37], available at `http://matteo.rionda.to/software/ABRA-radebetw.tbz2`. This algorithm samples pairs of nodes (s,t), and it adds the fraction of st-paths passing from v to the approximation of the betweenness of v, for each node v. The stopping condition is based on a key result in statistical learning theory, and there is a scheduler that decides when it should be tested. Following the suggestions by the authors, we use both the automatic scheduler ABRA-Aut, which uses a heuristic approach to decide when the stopping condition should be tested, and the geometric scheduler ABRA-1.2, which tests the stopping condition after $(1.2)^i k$ iterations, for each integer i.

The test is performed on a dataset made by 15 undirected and 15 directed real-world networks, taken from the datasets SNAP (`snap.stanford.edu/`), LASAGNE (`piluc.dsi.unifi.it/lasagne`), and KONECT (`http://konect.uni-koblenz.de/networks/`). As in [37], we have considered all values of $\lambda \in \{0.03, 0.025, 0.02, 0.015, 0.01, 0.005\}$, and $\delta = 0.1$. All the algorithms have to provide an approximation $\tilde{\boldsymbol{b}}(v)$ of $\text{bc}(v)$ for each v such that $\Pr\left(\forall v, \left|\tilde{\boldsymbol{b}}(v) - \text{bc}(v)\right| \le \lambda\right) \ge 1-\delta$. In Figure 1, we report the time needed by the different algorithms on every graph for $\lambda = 0.005$ (the behavior with different values of λ is very similar). More detailed results are reported in Appendix F of the full version [12].

From the figure, we see that KADABRA is much faster than all the other algorithms, on all graphs: on average, our algorithm is about 100 times faster than RK in undirected graphs, and about 70 times faster in directed graphs; it is also more than 1 000 times faster than ABRA. The latter value is due to the fact that the ABRA algorithm has large running times on few networks: in some cases, it did not even conclude its computation within one hour. The authors confirmed that this behavior might be due to some bugs in the code, which seems to affect it only on specific graphs: indeed, in most networks, the performances

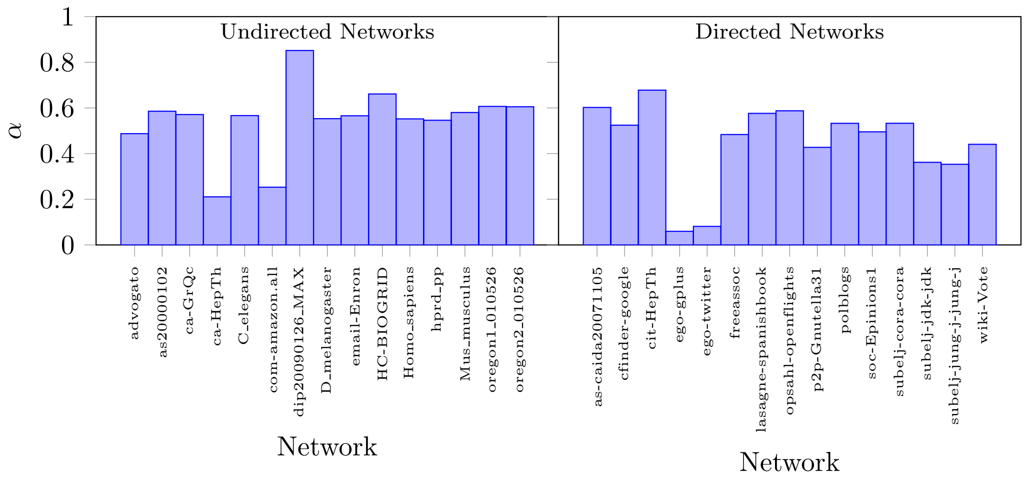


Figure 2 The exponent α such that the average number of edges visited during a bidirectional BFS is n^{α}.

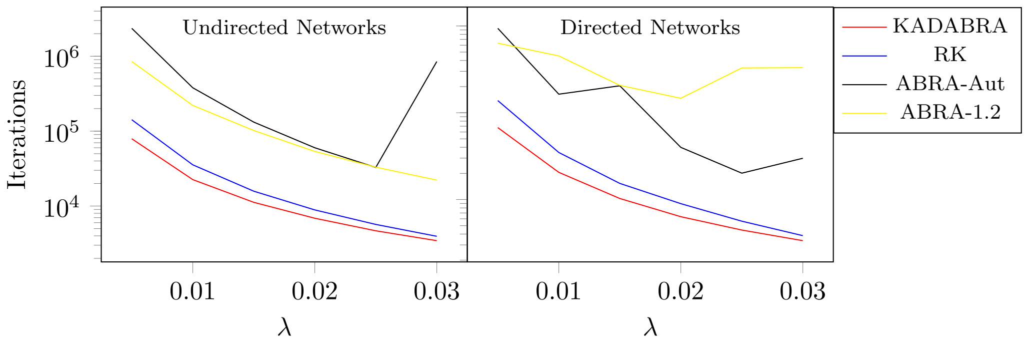


Figure 3 The average number of samples needed by the different algorithms.

of ABRA are better than those of the RK algorithm (but, still, not better than KADABRA).

In order to explain these data, we take a closer look at the improvements obtained through the bidirectional BFS, by considering the average number of edges m_{avg} that the algorithm visits in order to sample a shortest path (for all our competitors, $m_{\text{avg}} = m$, since they perform a full BFS). In Figure 2, for each graph in our dataset, we plot $\alpha = \frac{\log(m_{\text{avg}})}{\log(m)}$ (intuitively, this means that the average number of edges visited is m^{α}).

The figure shows that, apart from few cases, the number of edges visited is close to $n^{\frac{1}{2}}$, confirming the results in Section 4. This means that, since many of our networks have approximately 10 000 edges, the bidirectional BFS is about 100 times faster than the standard BFS. Finally, for each value of λ, we report in Figure 3 the number of samples needed by all the algorithms, averaged over all the graphs in the dataset.

From the figure, KADABRA needs to sample the smallest amount of shortest paths, and the average improvement over RK grows when λ tends to 0, from a factor 1.14 (resp., 1.14) if $\lambda = 0.03$, to a factor 1.79 (resp., 2.05) if $\lambda = 0.005$ in the case of undirected (resp., directed) networks. Again, the behavior of ABRA is highly influenced by the behavior on few networks, and as a consequence the average number of samples is higher. In any case, also in the graphs where ABRA has good performances, KADABRA still needs a smaller number of samples.

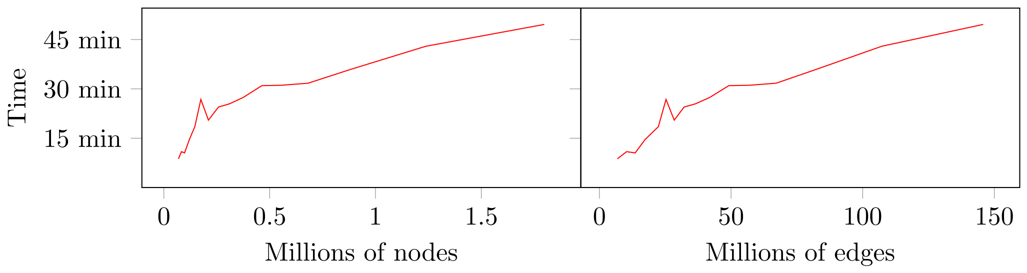


Figure 4 The total time of computation of KADABRA on increasing snapshots of the IMDB graph.

Computing Top-k Centralities

In the second experiment, we let KADABRA compute the top-k betweenness centralities of large graphs, which were unfeasible to handle with the previous algorithms.

The first set of graph is a series of temporal snapshots of the IMDB actor collaboration network, in which two actors are connected if they played together in a movie. The snapshots are taken every 5 years from 1940 to 2010, including a last snapshot in 2014, with 1 797 446 nodes and 145 760 312 edges. The graphs are extracted from the IMDB website (`http://www.imdb.com`), and they do not consider TV-series, awards-shows, documentaries, game-shows, news, realities and talk-shows, in accordance to what was done in `http://oracleofbacon.org`.

The other graph considered is the Wikipedia citation network, whose nodes are Wikipedia pages, and which contains an edge from page p_1 to page p_2 if the text of page p_1 contains a link to page p_2. The graph is extracted from DBPedia 3.7 (`http://wiki.dbpedia.org/`), and it consists of 4 229 697 nodes and 102 165 832 edges.

We have run our algorithm with $\lambda = 0.0002$ and $\delta = 0.1$: as discussed in Section 5, this means that either two nodes are ranked correctly, or their centrality is known with precision at most λ. As a consequence, if two nodes are not ranked correctly, the difference between their real betweenness is at most 2λ. The full results are available in Appendix G of the full version [12].

All the graphs were processed in less than one hour, apart from the Wikipedia graph, which was processed in approximately 1 hour and 38 minutes. In Figure 4, we plot the running times for the actor graphs: from the figure, it seems that the time needed by our algorithm scales slightly sublinearly with respect to the size of the graph. This result respects the results in Section 4, because the degrees in the actor collaboration network are power law distributed with exponent $\beta \approx 2.13$ (`http://konect.uni-koblenz.de/networks/actor-collaboration`). Finally, we observe that the ranking is quite precise: indeed, most of the times, there are very few nodes in the top-5 with the same ranking, and the ranking rarely contains significantly more than 10 nodes.

Acknowledgements. The authors would like to thank Matteo Riondato for several constructive comments on an earlier version of this work. We also thank Elisabetta Bergamini, Richard Lipton, and Sebastiano Vigna for helpful discussions and Holger Dell for his help with the experiments.

References

1 Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. Subcubic equivalences between graph centrality problems, apsp and diameter. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1681–1697. SIAM, 2015.

2 Amir Abboud, Virginia V. Williams, and Joshua Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter. In *Proceedings of the 26th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 377–391, may 2016. URL: http://arxiv.org/abs/1506.0179.

3 Ankit Aggarwal, Amit Deshpande, and Ravi Kannan. Adaptive Sampling for k-Means Clustering. In Irit Dinur, Klaus Jansen, Joseph Naor, and José Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, number 5687 in Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009.

4 Jac M Anthonisse. The rush in a directed graph. *Stichting Mathematisch Centrum. Mathematische Besliskunde*, BN(9/71):1–10, 1971.

5 David A. Bader, Shiva Kintali, Kamesh Madduri, and Milena Mihail. Approximating betweenness centrality. *The 5th Workshop on Algorithms and Models for the Web-Graph*, 2007.

6 Alex Bavelas. A mathematical model for group structures. *Human organization*, 7(3):16–30, 1948.

7 Elisabetta Bergamini. private communication, 2016.

8 Elisabetta Bergamini and Henning Meyerhenke. Fully-dynamic approximation of betweenness centrality. In *ESA*, 2015.

9 Béla Bollobás. A probabilistic proof of an asymptotic formula for the number of labelled regular graphs. *European Journal of Combinatorics*, 1(4):311–316, 1980. doi:10.1016/S0195-6698(80)80030-8.

10 Michele Borassi, Pierluig Crescenzi, and Michel Habib. Into the square - On the complexity of some quadratic-time solvable problems. In *Proceedings of the 16th Italian Conference on Theoretical Computer Science (ICTCS)*, pages 1–17, 2015.

11 Michele Borassi, Pierluigi Crescenzi, and Luca Trevisan. An Axiomatic and an Average-Case Analysis of Algorithms and Heuristics for Metric Properties of Graphs. *arXiv:1604.01445 [cs]*, April 2016.

12 Michele Borassi and Emanuele Natale. Kadabra is an adaptive algorithm for betweenness via random approximation. *arXiv preprint arXiv:1604.08553*, 2016.

13 Stephen P. Borgatti and Martin G. Everett. A graph-theoretic perspective on centrality. *Social Networks*, 28:466–484, 2006.

14 Ulrik Brandes. A faster algorithm for betweenness centrality. *The Journal of Mathematical Sociology*, 25(2):163–177, jun 2001. doi:10.1080/0022250X.2001.9990249.

15 Ulrik Brandes. On variants of shortest-path betweenness centrality and their generic computation. *Social Networks*, 30:136–145, 2008.

16 Ulrik Brandes and Christian Pich. Centrality Estimation in Large Networks. *International Journal of Bifurcation and Chaos*, 17(07):2303–2318, 2007. doi:10.1142/S0218127407018403.

17 Bernard S Cohn and McKim Marriott. Networks and centres of integration in indian civilization. *Journal of social Research*, 1(1):1–9, 1958.

18 Shlomi Dolev, Yuval Elovici, and Rami Puzis. Routing betweenness centrality. *J. ACM*, 57, 2010.

19 David A. Easley and Jon M. Kleinberg. Networks, crowds, and markets - reasoning about a highly connected world. In *DAGLIB*, 2010.

20 David Eppstein and Joseph Wang. Fast approximation of centrality. *J. Graph Algorithms Appl.*, 8:39–45, 2001.

21 Dóra Erdős, Vatche Ishakian, Azer Bestavros, and Evimaria Terzi. A divide-and-conquer algorithm for betweenness centrality. In *Proceedings of the 2015 SIAM International Conference on Data Mining*, pages 433–441, 2015.

22 Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In Catherine C. McGeoch, editor, *Experimental Algorithms: 7th International Workshop, WEA 2008*, pages 319–333. Springer Berlin Heidelberg, 2008.

23 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which Problems Have Strongly Exponential Complexity? *Journal of Computer and System Sciences*, 63(4):512–530, dec 2001. `doi:10.1006/jcss.2001.1774`.

24 Riko Jacob, Dirk Koschützki, Katharina Anna Lehmann, Leon Peeters, and Dagmar Tenfelde-Podehl. Algorithms for centrality indices. In *DAGSTUHL*, 2004.

25 Hermann Kaindl and Gerhard Kainz. Bidirectional heuristic search reconsidered. *J. Artif. Intell. Res. (JAIR)*, 7:283–317, 1997.

26 Yeon-sup Lim, Daniel S Menasché, Bruno Ribeiro, Don Towsley, and Prithwish Basu. Online estimating the k central nodes of a network. *Proceedings of IEEE NSW*, pages 118–122, 2011.

27 Richard J. Lipton and Jeffrey F. Naughton. Query Size Estimation by Adaptive Sampling. *Journal of Computer and System Sciences*, 51(1):18–25, August 1995. `doi:10.1006/jcss.1995.1050`.

28 Richard J. Lipton and Naughton, Jeffrey F. Estimating the size of generalized transitive closures. In *Proceedings of the 15th Int. Conf. on Very Large Data Bases*, 1989.

29 Linyuan Lu and Fan R. K. Chung. *Complex graphs and networks*. Number no. 107 in CBMS regional conference series in mathematics. American Mathematical Society, 2006.

30 Mark Newman. *Networks: an introduction*. OUP Oxford, 2010.

31 Mark EJ Newman. Scientific collaboration networks. ii. shortest paths, weighted networks, and centrality. *Physical review E*, 64(1):016132, 2001.

32 Ilkka Norros and Hannu Reittu. On a conditionally Poissonian graph process. *Advances in Applied Probability*, 38(1):59–75, 2006.

33 Jürgen Pfeffer and Kathleen M Carley. k-centralities: local approximations of global measures based on shortest paths. In *Proceedings of the 21st international conference companion on World Wide Web*, pages 1043–1050. ACM, 2012.

34 Andrea Pietracaprina, Matteo Riondato, Eli Upfal, and Fabio Vandin. Mining Top-K Frequent Itemsets Through Progressive Sampling. *Data Mining and Knowledge Discovery*, 21(2):310–326, September 2010. `doi:10.1007/s10618-010-0185-7`.

35 Ira Pohl. *Bi-directional and heuristic search in path problems*. PhD thesis, Dept. of Computer Science, Stanford University., 1969.

36 Matteo Riondato and Evgenios M Kornaropoulos. Fast approximation of betweenness centrality through sampling. *Data Mining and Knowledge Discovery*, 30(2):438–475, 2015.

37 Matteo Riondato and Eli Upfal. ABRA: Approximating Betweenness Centrality in Static and Dynamic Graphs with Rademacher Averages. *arXiv preprint 1602.05866*, pages 1–27, 2016. `arXiv:1602.05866`.

38 Ahmet Erdem Sariyüce, Erik Saule, Kamer Kaya, and Ümit V Çatalyürek. Shattering and compressing networks for betweenness centrality. In *SIAM Data Mining Conference (SDM). SIAM*, 2013.

39 Marvin E Shaw. Group structure and the behavior of individuals in small groups. *The Journal of psychology*, 38(1):139–149, 1954.

40 Alfonso Shimbel. Structural parameters of communication networks. *The bulletin of mathematical biophysics*, 15(4):501–507, 1953.

41 Christian L. Staudt, Aleksejs Sazonovs, and Henning Meyerhenke. Networkit: an interactive tool suite for high-performance network analysis. *arXiv preprint 1403.3005*, pages 1–25, 2014.

42 Remco van der Hofstad. Random graphs and complex networks. Vol. II. Manuscript, 2014.

43 Flavio Vella, Giancarlo Carbone, and Massimo Bernaschi. Algorithms and heuristics for scalable betweenness centrality computation on multi-gpu systems. *CoRR*, abs/1602.00963, 2016.

44 Stanley Wasserman and Katherine Faust. *Social network analysis: Methods and applications*, volume 8. Cambridge university press, 1994.

45 Ryan Williams and Huacheng Yu. Finding orthogonal vectors in discrete structures. In *Proceedings of the 24th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1867–1877, 2014. `doi:10.1137/1.9781611973402.135`.

Separation of Cycle Inequalities for the Periodic Timetabling Problem*

Ralf Borndörfer[1], Heide Hoppmann[2], and Marika Karbstein[3]

1 **Zuse Institute Berlin, Berlin, Germany**
`borndoerfer@zib.de`
2 **Zuse Institute Berlin, Berlin, Germany**
`hoppmann@zib.de`
3 **Zuse Institute Berlin, Berlin, Germany**
`karbstein@zib.de`

Abstract

Cycle inequalities play an important role in the polyhedral study of the periodic timetabling problem. We give the first pseudo-polynomial time separation algorithm for cycle inequalities, and we give a rigorous proof for the pseudo-polynomial time separability of the change-cycle inequalities. The efficiency of these cutting planes is demonstrated on real-world instances of the periodic timetabling problem.

1998 ACM Subject Classification F.2.2. Nonnumerical Algorithms and Problems, G.1.6 Optimization

Keywords and phrases periodic timetabling, cycle inequalities, separation algorithm

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.21

1 Introduction

Periodic timetable construction is a fascinating problem because it is intuitively understood and mathematically formulated, but very hard to solve. In fact, even though real world instances give rise to relatively small optimization models, branch-and-bound based methods can easily stall with large duality gaps. The likely reasons for this resistivity are the occurrence of genuine integer variables, symmetries, and modulo constraints.

The classical approach to periodic timetabling is to use a formulation in terms of the periodic event scheduling problem (PESP) by Serafini and Ukovich [14]. This model has been the basis for the development of a variety of exact and heuristic solution methods for the optimization and the feasibility version. Integer programming approaches were proposed, e.g., by Nachtigall [9], Lindner [7], and Liebchen [3]. A topological search method based on cohomology feasibility was used by Schrijver [12] to optimize a Dutch railway timetable. A modulo network simplex heuristic was invented by Nachtigall and Opitz [8]. Liebchen and Peeters [5] studied the relation to integral cycle bases to find tighter lower bounds. A SAT approach for the feasibility problem was developed by Kümmling et al. [2]. A comprehensive and up-to-date survey of the literature on mathematical timetable optimization and its applications is summarized in Sels et al. [13].

The best method to compute lower bounds for the optimization problem is to study the polyhedral structure of the periodic timetabling problem (PTP) associated with the PESP.

* This research was carried out in the framework of MATHEON supported by Einstein Foundation Berlin.

Several classes of valid inequalities have been identified, namely, chain, cycle, change-cycle, flow, and multi-circuit inequalities, see [3, 7, 9, 10, 11]. Some of them are known to be facet defining or in some Chvátal closure for the PESP polytope or relaxations of it under certain conditions [6]. It is also known that the change-cycle inequalities can be separated in pseudo-polynomial time [9]. The cycle inequalities have been used computationally by means of heuristic separation. They improve the lower bound significantly and are considered to be "the computationally most interesting cuts" [3, p. 210].

We study in this paper the separation problem for the cycle and the change-cycle inequalities for the periodic timetabling problem. We give the first pseudo-polynomial time separation algorithm for cycle inequalities. Its complexity is $\mathcal{O}(Tn^2m)$, where T is the period time, n the number of nodes, and m the number of arcs. The change-cycle inequalities have been studied by Nachtigall [9], who gave a rough sketch of a pseudo-polynomial algorithm and claimed a complexity of $\mathcal{O}(T(mn+n^2))$. We cannot follow this argument, but give a precise description of the algorithm and prove a complexity of $\mathcal{O}(T^2n^2m)$. Computational results on real world instances from a Dutch railway system and two German cities corroborate the efficiency of these cuts.

The paper is structured as follows. Section 2 gives a mathematical statement of the problem. Section 3 introduces the periodic slack polyhedron and states the cycle and the change-cycle inequalities. Sections 4 and 5 contain the separation algorithms for the change-cycle inequalities and the cycle inequalities. They are based on similar ideas, but cycle separation requires the setup of an additional auxiliary graph. Section 6 concludes with computational results.

2 Periodic Timetabling Problem and PESP

Most models in the literature about periodic timetabling are based on the *periodic event scheduling problem* (PESP) developed by [14]. In this problem, we are given a directed graph $\mathcal{N} = (V, A)$, the *event-activity network*. The nodes V are called *events* and represent arrivals and departures of lines at their stations. The arcs $A \subseteq V \times V$ are called *activities* and model lines driving between stations, waiting at stations, and possible transfers for passengers between lines at stations. Further, each activity $a \in A$ is associated with a lower and an upper time bound $\ell_a, u_a \in \mathbb{Q}_{\geq 0}$, respectively, on its duration. Let $n = |V|$ be the number of events and $m = |A|$ be the number of activities.

A *periodic timetable* $\pi : V \to [0, T)$ determines the timings of all events, which are assumed to repeat periodically w.r.t. a *period time* $T \in \mathbb{N}$. Given $x \in \mathbb{Q}$, we define the modulo operator by $[x]_T := \min\{x + zT : x + zT \geq 0, z \in \mathbb{Z}\}$. We call a timetable *feasible* if the *periodic interval constraints*

$$[\pi_w - \pi_v - \ell_a]_T \in [0, u_a - \ell_a] \quad \forall\, a = (v, w) \in A \tag{1}$$

are satisfied. We assume w.l.o.g. that $\ell_a < T$ and $u_a - \ell_a < T$ for all $a \in A$. Many operational requirements can be modeled with the constraints (1), see [4]. For a feasible timetable π, the *periodic tension* of activity $a \in A$ is defined by $x_a := \ell_a + [\pi_w - \pi_v - \ell_a]_T$ and corresponds to its duration. The *periodic slack* of activity $a \in A$ is defined by $y_a := [\pi_w - \pi_v - \ell_a]_T$. Given activity weights $w \in \mathbb{Q}^A$, the goal of the periodic timetabling problem is to find a feasible timetable that minimizes the weighted sum of the periodic slacks, i.e., $\min \sum_{a \in A} w_a\, y_a$.

An *oriented cycle* C in $\mathcal{N}$ is a sequence $C = (v_0, a_1, v_1, \ldots, a_k, v_k)$, where $k \geq 1$, $v_1, \ldots, v_k \in V$, $a_1, \ldots, a_k \in A$, $v_0 = v_k$, and $a_i \in \{(v_{i-1}, v_i), (v_i, v_{i-1})\}$. Activities with $a_i = (v_{i-1}, v_i) \in C$ are called *forward directed* and activities with $a_i = (v_i, v_{i-1}) \in C$

backward directed. An oriented cycle containing only forward directed activities is called a *circuit*. An oriented cycle is *elementary* if no event appears more than once in the sequence. For an oriented cycle C in $\mathcal{N}$, we define its incident vector $\gamma_C \in \{-1, 0, 1\}^A$ as

$$\gamma_{C_a} = \begin{cases} 1 & \text{if } a \in C \text{ and } a \text{ is forward directed,} \\ -1 & \text{if } a \in C \text{ and } a \text{ is backward directed,} \\ 0 & \text{if } a \notin C. \end{cases}$$

For convenience, we will refer interchangeably to C and γ_C. Let $\mathcal{B} = \{C_1, \dots, C_\nu\}$, $\nu = m - n + 1$, be a cycle basis of $\mathcal{N}$ and denote by $\Gamma \in \mathbb{Z}^{\mathcal{B} \times A}$ the corresponding cycle matrix, i.e., the rows of Γ correspond to the characteristic vectors $\gamma_{C_i} \in \{-1, 0, 1\}^A$, $i \in \{1, \dots, \nu\}$. Introducing periodic slack variables $y \in \mathbb{R}^A$ and *periodic offset* variables $z \in \mathbb{Z}^A$, we can state the periodic timetabling problem as the following mixed-integer program [9, 3]:

$$\begin{aligned} \text{(PTP)} \quad \min \quad & \sum_{a \in A} w_a y_a \\ \text{s.t.} \quad & \Gamma\, y - T\, \Gamma\, z = -\Gamma \ell && (2) \\ & 0 \leq y \leq u - \ell && (3) \\ & z \in \mathbb{Z}^A && (4) \end{aligned}$$

3 Periodic Slack Polyhedron

The literature considers different versions of the PTP polyhedron, e.g., the projection on the space of the periodic slack variables or the periodic offset variables, see [9, 3, 6]. Nachtigall [9] also considers the polyhedron that is obtained when the upper bounds in constraints (3) are omitted. In the following, we study the polyhedron $P_{IP}(\text{PTP})$ associated with the feasible solutions of (PTP), i.e., a polyhedron defined in the slack and offset space. We recall the cycle and change-cycle inequalities in a unified notation.

▶ **Definition 1.** The *periodic slack and offset space* is defined by

$$\mathcal{S} = \left\{ (y, z) \in \mathbb{R}^A \times \mathbb{Z}^A | \Gamma\, y - T\, \Gamma\, z = -\Gamma \ell, 0 \leq y \leq u - \ell \right\}.$$

The *periodic slack polyhedron* is defined by

$$P_{IP}(\text{PTP}) = \text{conv}(\mathcal{S})$$

and the corresponding LP relaxation by

$$P_{LP}(\text{PTP}) = \left\{ (y, z) \in \mathbb{R}^A \times \mathbb{R}^A | \Gamma\, y - T\, \Gamma\, z = -\Gamma \ell, 0 \leq y \leq u - \ell \right\}.$$

The following lemma shows that the cycle equations (2) do not only hold for integer periodic offset variables and the cycles of the cycle basis but for any feasible solution of the LP relaxation of (PTP) and any cycle in the event-activity network.

▶ **Lemma 2.** *Let $(y, z) \in P_{LP}(\text{PTP})$ and let $\gamma \in \mathbb{Z}^A$ be an oriented cycle in $\mathcal{N}$. Then we have*

$$\gamma^t y = -\gamma^t \ell + T\, \gamma^t z.$$

Proof. Since $\mathcal{B}$ is a cycle basis, there exists a vector $\lambda \in \mathbb{R}^\nu$ such that $\gamma = \Gamma^t\, \lambda$. Hence, we get

$$\gamma^t y = \lambda^t\, \Gamma\, y = \lambda^t\, (-\Gamma\, \ell + T\, \Gamma\, z) = -(\Gamma^t\, \lambda)^t \ell + T\, (\Gamma^t\, \lambda)^t\, z = -\gamma^t \ell + T\, \gamma^t\, z. \qquad ◀$$

Lemma 2 implies that for any feasible solution of (PTP) the following modulo cycle equations hold.

▶ **Corollary 3.** *Let $(y, z) \in \mathcal{S}$ and let $\gamma \in \mathbb{Z}^A$ be an oriented cycle in $\mathcal{N}$. Then we have*

$$\gamma^t y \equiv_T -\gamma^t \ell.$$

Proof. The corollary follows directly from $\gamma^t z \in \mathbb{Z}$ and Lemma 2. ◀

For convenience, we introduce further notation. For an oriented cycle $\gamma \in \mathbb{Z}^m$ in $\mathcal{N}$ we define the positive part $\gamma_+ \in \mathbb{Z}^m$ and the negative part $\gamma_- \in \mathbb{Z}^m$, respectively, by

$$\gamma_{+,a} = \begin{cases} 1 & \text{if } \gamma_a = 1 \\ 0 & \text{else} \end{cases} \quad \text{and} \quad \gamma_{-,a} = \begin{cases} 1 & \text{if } \gamma_a = -1 \\ 0 & \text{else} \end{cases}$$

for all $a \in A$ i.e., $\gamma = \gamma_+ - \gamma_-$.

The following class of valid inequalities was introduced by Nachtigall [9] and are defined for every oriented cycle in the event-activity network.

▶ **Theorem 4.** *Let $\gamma \in \mathbb{Z}^A$ be an oriented cycle in $\mathcal{N}$ and define $\alpha = [-\gamma^t \ell]_T$. Then the* change-cycle inequality

$$(T - \alpha)\, \gamma_+^t y + \alpha\, \gamma_-^t y \geq \alpha\, (T - \alpha) \tag{5}$$

is valid for P_{IP}(PTP)*.*

A second class of inequalities are also induced by the oriented cycles in the event-activity network and were first described by Odijk [10]. These inequalities are denoted as *cycle inequalities*. They are usually defined in terms of the periodic offset variables. We will show next that they can also be defined in terms of the slack variables.

▶ **Theorem 5.** *Let $\gamma \in \mathbb{Z}^A$ be an oriented cycle in $\mathcal{N}$. Then the z*-cycle inequality

$$\gamma^t\, z \geq \left\lceil \frac{1}{T} \left(\gamma_+^t \ell - \gamma_-^t u \right) \right\rceil \tag{6}$$

is valid for P_{IP}(PTP)*.*

Proof. Let $(y, z) \in \mathcal{S}$. We have with Lemma 2

$$T\, \gamma^t\, z = \gamma^t y + \gamma^t \ell = \gamma_+^t y - \gamma_-^t y + \gamma^t \ell \geq -\gamma_-^t\, (u - \ell) + \gamma^t \ell = \gamma_+^t \ell - \gamma_-^t u.$$

Since $\gamma^t\, z \in \mathbb{Z}$, the inequality (6) follows. ◀

▶ **Lemma 6.** *Let $\alpha \in \mathbb{R}$, then*

$$[-\alpha]_T + \alpha = T \left\lceil \frac{1}{T} \alpha \right\rceil. \tag{7}$$

Proof. Let $z \in \mathbb{Z}$ and $\alpha \in \mathbb{R}$ then $-\alpha - T\, z = [-\alpha]_T$ and $-1 < -\frac{1}{T}\, [-\alpha]_T \leq 0$. We get

$$\begin{aligned} [-\alpha]_T + \alpha = -T\, z &= T\Big(-z + \underbrace{\left\lceil -\frac{1}{T}\, [-\alpha]_T \right\rceil}_{=0} \Big) \\ &= T \left\lceil -z - \frac{1}{T}\, [-\alpha]_T \right\rceil = T \left\lceil \frac{1}{T}(-T\, z - [-\alpha]_T) \right\rceil = T \left\lceil \frac{1}{T} \alpha \right\rceil. \end{aligned}$$ ◀

With Lemma 6 we can show that the z-cycle inequalities can be expressed equivalently in terms of the slack variables.

▶ **Theorem 7.** *Let $\gamma \in \mathbb{Z}^A$ be an oriented cycle in $\mathcal{N}$. Let $(y, z) \in P_{LP}(\text{PTP})$, then the* z-cycle inequality (6) *holds if and only if the the* y-cycle inequality

$$\gamma^t y \geq \left[-\gamma_+^t \ell + \gamma_-^t u\right]_T + \gamma_-^t (\ell - u) \tag{8}$$

holds.

Proof. Let $(y, z) \in P_{LP}(\text{PTP})$ and assume that z satisfies the *z-cycle inequality* (6) for γ. Using first Lemma 6 and then Lemma 2 we have

$$\begin{aligned}
& & \gamma^t z &\geq \left\lceil \frac{1}{T}\left(\gamma_+^t \ell - \gamma_-^t u\right) \right\rceil \\
& & &= \frac{1}{T}\left(\left[-\gamma_+^t \ell + \gamma_-^t u\right]_T + \gamma_+^t \ell - \gamma_-^t u\right) \\
\Leftrightarrow & & \gamma^t y + \gamma^t \ell &\geq \left[-\gamma_+^t \ell + \gamma_-^t u\right]_T + \gamma_+^t \ell - \gamma_-^t u \\
\Leftrightarrow & & \gamma^t y &\geq \left[-\gamma_+^t \ell + \gamma_-^t u\right]_T + \gamma_+^t \ell - \gamma_-^t u - \gamma^t \ell \\
& & &= \left[-\gamma_+^t \ell + \gamma_-^t u\right]_T + \gamma_+^t \ell - \gamma_-^t u - \gamma_+^t \ell + \gamma_-^t \ell \\
& & &= \left[-\gamma_+^t \ell + \gamma_-^t u\right]_T - \gamma_-^t u + \gamma_-^t \ell \\
& & &= \left[-\gamma_+^t \ell + \gamma_-^t u\right]_T + \gamma_-^t (\ell - u).
\end{aligned}$$ ◀

▶ **Corollary 8.** *Let $\gamma \in \mathbb{Z}^A$ be an oriented cycle in $\mathcal{N}$. Then the* y-cycle inequality

$$\gamma^t y \geq \left[-\gamma_+^t \ell + \gamma_-^t u\right]_T + \gamma_-^t (\ell - u)$$

is valid for $P_{IP}(\text{PTP})$.

4 Separation of Change-Cycle Inequalities

In this section we describe a pseudo-polynomial dynamic programming procedure to separate violated change-cycle inequalities (5). The idea of this algorithm was originally proposed by Nachtigall [9]. He claimed a running time of $\mathcal{O}(T(mn + n^2)$, but did not give a proof. We prove a complexity of $\mathcal{O}(T^2 n^2 m)$.

Given a point $(y^*, z^*) \in P_{LP}(\text{PTP})$, the separation problem is to find an oriented cycle γ in $\mathcal{N}$ such that the change-cycle inequality (5) induced by γ is violated, i.e., for $\alpha_0 = [-\gamma^t \ell]_T$ it holds

$$(T - \alpha_0)\gamma_+^t y^* + \alpha_0\, \gamma_-^t y^* < \alpha_0\, (T - \alpha_0),$$

or to conclude that no such cycle exists. The idea is to solve for each fixed $\alpha_0 \in \{0, \ldots, T-1\}$ the problem

$$f^*(\alpha_0) = \min\{(T - \alpha_0)\gamma_+^t y^* + \alpha_0\, \gamma_-^t y^* | \gamma \text{ oriented cycle in } \mathcal{N}, \left[-\gamma^t \ell\right]_T = \alpha_0\}, \tag{9}$$

which is to find the minimum cost cycle w.r.t.

$$c_a = \begin{cases} (T - \alpha_0) y_a^* & \text{if } \gamma_a = 1 \\ \alpha_0 y_a^* & \text{if } \gamma_a = -1 \\ 0 & \text{else} \end{cases}$$

of all oriented cycles with $[-\gamma^t \ell]_T = \alpha_0$. Note that a violated change-cycle inequality exists if and only if for some $\alpha_0 \in \{0, \dots, T-1\}$ it holds $f^*(\alpha_0) < \alpha_0 \, (T - \alpha_0)$.

The minimization problem (9), again, can be solved with a dynamic program that iterates over the cycle lengths w.r.t. the number of activities. We denote by a *chain* a path that can contain forward directed activities as well as backward directed activities. Let $\mathcal{C}_{ij}^k$ be the set of all chains in $\mathcal{N}$ from event $i \in V$ to event $j \in V$ that contain exactly k activities, given by their characteristic vectors. For $\alpha \in \{0, \dots, T-1\}$, let

$$f_{ij}^k(\alpha_0, \alpha) := \min \left\{ (T - \alpha_0) \sum_{\substack{a \in A: \\ p_a > 0}} y_a^* + \alpha_0 \sum_{\substack{a \in A: \\ p_a < 0}} y_a^* \middle| p \in \mathcal{C}_{ij}^k, \alpha = \left[-p^t \ell\right]_T \right\}$$

be the minimum length w.r.t. c_a of all chains in $\mathcal{C}_{ij}^k$ with $\alpha = [-p^t \ell]_T$. Since a chain of length $k \geq 2$ consists of a chain of length $k-1$ and an additional activity, the following recursive equation holds

$$f_{ij}^{k+1}(\alpha_0, \alpha) := \min \left\{ \min_{\substack{a=(u,j) \\ \left[\alpha' - \ell_a\right]_T = \alpha}} f_{iu}^k(\alpha_0, \alpha') + (T - \alpha_0) y_a^*, \min_{\substack{a=(j,u) \\ \left[\alpha' + \ell_a\right]_T = \alpha}} f_{iu}^k(\alpha_0, \alpha') + \alpha_0 y_a^* \right\}, \tag{10}$$

for all $k \geq 0$ with

$$f_{ij}^0(\alpha_0, \alpha) = \begin{cases} 0 & \text{if } i = j, \alpha = 0 \\ \infty & \text{else.} \end{cases}$$

Since every elementary cycle has at most n activities and $c_a \geq 0$ for all $a \in A$, the minimum length w.r.t. c of all oriented cycles γ with $\alpha_0 = [-\gamma^t \ell]_T$ is given by

$$f^*(\alpha_0) = \min_{i \in V} \min_{k=1}^{n} f_{ii}^k(\alpha_0, \alpha_0).$$

For fixed $k \in \{0, \dots, n-1\}$, the recursive equation (10) can be solved with Algorithm 1.

▶ **Theorem 9.** *For given $\alpha_0 \in \{0, \dots, T-1\}$, $k \in \{0, \dots, n-1\}$, and f_{ij}^k for all $i, j \in V$, Algorithm 1 computes $f_{ij}^{k+1}(\alpha_0, \alpha)$ for all $i, j \in V$ and $\alpha \in \{0, \dots, T-1\}$ in $\mathcal{O}(Tmn)$.*

Proof. For a given $k \in \{0, \dots, n-1\}$ and $\alpha_0 \in \{0, \dots, T-1\}$, Algorithm 1 obviously solves equation (10) for all $i, j \in V$ and for all $\alpha \in \{0, \dots, T-1\}$. The computation involves $\mathcal{O}(Tmn)$ elementary operations. ◀

The described separation algorithm is given in pseudocode in Algorithm 2.

▶ **Theorem 10.** *Algorithm 2 solves the separation problem for the change-cycle inequalities* (5) *in $\mathcal{O}(T^2 n^2 m)$.*

Proof. The Algorithm 2 solves for each $\alpha_0 \in \{0, \dots, T-1\}$ the minimization problem (9) and correctly reports if there exists an $\alpha_0 \in \{0, \dots, T-1\}$ such that $f^*(\alpha_0) < \alpha_0 \, (T - \alpha_0)$. Hence, with the previous argumentation, the correctness of the algorithm follows. The algorithm needs to call Algorithm 1, see line 6 of Algorithm 2, in total kT-times. By Theorem 9, this results in a running time in $\mathcal{O}(T^2 n^2 m)$. ◀

Algorithm 1: Computing $f_{ij}^{k+1}(\alpha_0, \alpha)$ for all $\alpha \in \{0, \ldots, T-1\}$

```
Input  : $(y^*, z^*) \in P_{LP}(\text{PTP})$, $\alpha_0 \in \{0, \ldots, T-1\}$, $k \in \{0, \ldots, n-1\}$,
         $f_{ij}^k, \forall i, j \in V$
Output : $f_{ij}^{k+1}, \forall i, j \in V$
1  for $a = (u, v) \in A$ do
2      for $\alpha' := 0, \ldots, T-1$ do
3          for $i \in V$ do
4              $\alpha := [\alpha' - \ell_a]_T$
5              if $f_{iv}^{k+1}(\alpha_0, \alpha) > f_{iu}^k(\alpha_0, \alpha') + (T - \alpha_0)y_a^*$ then
6                  $f_{iv}^{k+1}(\alpha_0, \alpha) := f_{iu}^k(\alpha_0, \alpha') + (T - \alpha_0)y_a^*$
7              end
8              $\alpha := [\alpha' + \ell_a]_T$
9              if $f_{iu}^{k+1}(\alpha_0, \alpha) > f_{iv}^k(\alpha_0, \alpha') + \alpha_0 y_a^*$ then
10                 $f_{iu}^{k+1}(\alpha_0, \alpha) := f_{iv}^k(\alpha_0, \alpha') + \alpha_0 y_a^*$
11             end
12         end
13     end
14 end
15 return $f_{ij}^{k+1}, \forall i, j \in V$
```

Algorithm 2: Separation of Change-Cycle Inequalities

```
Input  : LP-point $(y^*, z^*) \in P_{LP}(\text{PTP})$
Output : Cycle $\gamma \in \mathcal{N}$ such that the change-cycle inequality (5) is violated, or
         $NULL$ if no such cycle exists.
1  $f^* := \infty$
2  $\rho^* := 0$
3  for $\alpha_0 := 0, \ldots, T-1$ do
4      $f_{ik}^k(\alpha_0, \alpha) := \begin{cases} 0 & \text{if } k = 0, i = j, \alpha = 0 \\ \infty & \text{else} \end{cases}$
5      for $k := 1, \ldots, n$ do
6          compute $f_{ij}^k(\alpha_0, \alpha)$ for all $\alpha \in \{0, \ldots, T-1\}$, for all $i, j \in V$
7      end
8      $f^*(\alpha_0) := \min_{i \in V} \min_{k=1}^n f_{ii}^k(\alpha_0, \alpha_0)$
9      if $f^*(\alpha_0) - \alpha_0(T - \alpha_0) < \rho^*$ then
10         $f^* := f^*(\alpha_0)$
11         $\rho^* := f^*(\alpha_0) - \alpha_0(T - \alpha_0)$
12     end
13 end
14 if $\rho^* < 0$ then
15     return $\gamma(f^*)$
16 else
17     return $NULL$
18 end
```

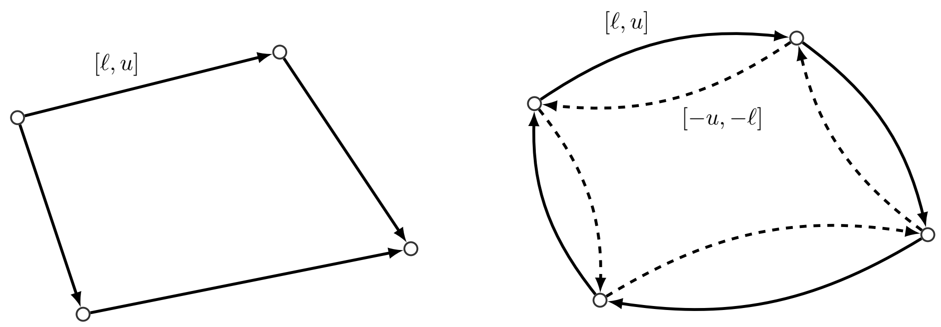


Figure 1 *Left*: the network $\mathcal{N}$, *Right*: the auxiliary network $\tilde{\mathcal{N}}$. The solid arcs correspond to the cycle γ and the circuit $\tilde{\gamma}$ from Theorem 11, respectively.

5 Separation of Cycle Inequalities

In this section we describe a pseudo-polynomial dynamic programming procedure to separate violated cycle inequalities (8) using an auxiliary network that was proposed by Nachtigall [9]. The auxiliary network allows to reduce cycle separation to finding certain modulo constrained circuits. Such a circuit can be found by a modification of the change-cycle separation Algorithm 2.

We obtain $\tilde{\mathcal{N}}$ by copying $\mathcal{N}$ and additionally introducing for each activity $a = (i, j)$ the back activity $\bar{a} = (j, i)$. The copies of the original activities keep their bounds, the bounds of the back activities are set to $\tilde{\ell}_{\bar{a}} = -u_a$ and $\tilde{u}_{\bar{a}} = -\ell_a$, see Figure 1. For a point $(y, z) \in P_{LP}(\text{PTP})$ we define $\tilde{y}$ on the activities of $\tilde{\mathcal{N}}$ by $\tilde{y}_a = y_a$ and $\tilde{y}_{\bar{a}} = u_a - \ell_a - y_a$.

▶ **Theorem 11.** *Let be $(y^*, z^*) \in P_{LP}(\text{PTP})$. Then, $\mathcal{N}$ contains a cycle γ that violates the y-cycle inequality (8), i.e.,*

$$\gamma^t y^* < \left[-\gamma_+^t \ell + \gamma_-^t u\right]_T + \gamma_-^t (\ell - u),$$

if and only if $\tilde{\mathcal{N}}$ contains a circuit $\tilde{\gamma}$ (cycle containing only forward directed activities) with

$$\tilde{\gamma}^t \tilde{y}^* < \left[-\tilde{\gamma}^t \tilde{\ell}\right]_T.$$

Proof. Let γ be a cycle in $\mathcal{N}$ and $\tilde{\gamma}$ be the circuit in $\tilde{\mathcal{N}}$ obtained by replacing all backward directed activities in γ with their auxiliary back activities, see Figure 1. Then we get

$$\begin{aligned}
& & \gamma^t y^* &< \left[-\gamma_+^t \ell + \gamma_-^t u\right]_T + \gamma_-^t (\ell - u) \\
&\Leftrightarrow & \gamma_+^t y^* + \gamma_-^t (u - \ell - y^*) &< \left[-\gamma_+^t \ell + \gamma_-^t u\right]_T \\
&\Leftrightarrow & \gamma_+^t \tilde{y}^* + \gamma_-^t \tilde{y}^* &< \left[-\gamma_+^t \ell - \gamma_-^t (-u)\right]_T \\
&\Leftrightarrow & \tilde{\gamma}^t \tilde{y}^* &< \left[-\tilde{\gamma}^t \tilde{\ell}\right]_T
\end{aligned}$$ ◀

By Theorem 11, there exists a violated cycle inequality (8) if and only if

$$\delta^* := \min\{\tilde{\gamma}^t \tilde{y}^* - \left[-\tilde{\gamma}^t \tilde{\ell}\right]_T | \tilde{\gamma} \text{ directed circuit in } \tilde{\mathcal{N}}\} < 0.$$

Hence, we can solve the separation problem by minimizing $\delta(\tilde{\gamma}) := \tilde{\gamma}^t \tilde{y}^* - \left[-\tilde{\gamma}^t \tilde{\ell}\right]_T$ over all directed circuits in the auxiliary network $\tilde{\mathcal{N}}$. We describe in the following the idea of the algorithm, which is given in pseudocode in Algorithm 3.

Let $\mathcal{P}_{ij}^k$ be the set of all (i,j)-paths in $\tilde{\mathcal{N}}$ that contain exactly k activities, given by their characteristic vectors. For $\alpha \in \{0,\dots,T-1\}$, let

$$d_{ij}^k(\alpha) := \min \left\{ \sum_{a \in A: p_a > 0} \tilde{y}_a^* \middle| p \in \mathcal{P}_{ij}^k, \alpha = \left[-p^t \tilde{\ell}\right]_T \right\} \tag{11}$$

be the minimum length with respect to $\tilde{y}^*$ of all paths in $\mathcal{P}_{ij}^k$ with $\alpha = \left[-p^t \tilde{\ell}\right]_T$. We can use the following recursive equation to compute (11)

$$d_{ij}^{k+1}(\alpha) := \min_{\substack{a=(u,j) \\ \left[\alpha' - \ell_a\right]_T = \alpha}} d_{iu}^k(\alpha') + \tilde{y}_a^*, \quad \forall\, k \geq 0$$

with

$$d_{ij}^0(\alpha) = \begin{cases} 0 & \text{if } i = j, \alpha = 0 \\ \infty & \text{else.} \end{cases}$$

Since every elementary circuit has at most n activities and $\tilde{y}_a^* \geq 0$ for all $a \in A$, the minimum length w.r.t. $\tilde{y}^*$ of all directed circuits $\tilde{\gamma}$ with $\alpha = \left[-\tilde{\gamma}^t \tilde{\ell}\right]_T$ is given by

$$d^*(\alpha) = \min_{i \in V} \min_{k=1}^{n} d_{ii}^k(\alpha)$$

and we have

$$\delta^* = \min\{d^*(\alpha) - \alpha | \alpha \in \{0,\dots,T-1\}\}.$$

▶ **Theorem 12.** *Algorithm 3 detects a violated cycle inequality in $\mathcal{O}(Tn^2m)$.*

Proof. The argumentation in this section proves that the algorithm computes δ^* and, thus, correctly detects violated cycle inequalities. The algorithm involves in total $\mathcal{O}(Tn^2m)$ elementary operations. ◀

6 Computational Results

This section gives some indication of the computational usefulness of cycle-separation compared to a heuristic separation.

As far as we know, the cycle inequalities (6) are added in cutting-plane algorithms only with heuristic separation algorithms, see [9, 3, 6]. In the so-called spanning tree heuristic, a minimum spanning tree of the event-activity network weighted with the slack values of the LP-solution is computed and the fundamental cycles of this tree are checked for violated inequalities.

We have implemented a full separation algorithm according to Algorithm 3 to separate all violated cycle inequalities (8) with a given maximum length. Such a length restriction is necessary to handle the memory consumption and the computation time of the separation algorithm. We tested a length restriction of 10, 15, and 20.

Our test set consists of seven instances, which are given in Table 1. The instance Wuppertal is based on the real multi-modal public transportation network of the city of Wuppertal for 2013. The remaining Wuppertal-instances are obtained by selecting a subset of lines of this instance. The Dutch instance is based on a network that was introduced

Algorithm 3: Separation of Cycle Inequalities

```
Input  : LP-point $(y^*, z^*) \in P_{LP}(\text{PTP})$
Output : Cycle $\gamma \in \mathcal{N}$ such that the cycle inequality (8) is violated, or $NULL$ if
         no such cycle exists.
1  $d^* := \infty$
2  $\delta^* = 0$
3  $d_{ij}^k(\alpha) := \begin{cases} 0 & \text{if } k = 0, i = j, \alpha = 0 \\ \infty & \text{else} \end{cases}$
4  for $k := 1, \dots, n$ do
5      for $a = (u, v) \in \tilde{A}$ do
6          for $\alpha' \in \{0, \dots, T-1\}$, $i \in V$ with $d_{iu}^k(\alpha') < \infty$ do
7              $\alpha := [\alpha' - \tilde{\ell}_a]_T$
8              if $d_{iv}^k(\alpha) > d_{iu}^{k-1}(\alpha') + \tilde{y}_a^*$ then
9                  $d_{iv}^k(\alpha) := d_{iu}^{k-1}(\alpha') + \tilde{y}_a^*$
10                 if $i = v$ and $d_{ii}^k(\alpha) - \alpha < \delta^*$ then
11                     $d^* := d_{ii}^k(\alpha)$
12                     $\delta^* := d_{ii}^k(\alpha) - \alpha$
13                 end
14             end
15         end
16     end
17 end
18 if $\delta^* < 0$ then
19     return $\gamma(d^*)$
20 else
21     return $NULL$
22 end
```

Table 1 Statistics on the test instances. The columns list the instance name, the number of stations and lines of the transportation network, the number of events and activities of the event-activity network, the number of slack variables, periodic offset variables, and constraints in the original problem, and the number of variables and constraints after preprocessing.

name	$\lvert\mathcal{S}\rvert$	$\lvert\mathcal{L}\rvert$	n	m	$\#y$	$\#z$	#cons	#vars*	#cons*
Wuppertal 14	28	14	168	499	52	39	39	52	39
Wuppertal 44	64	44	395	1 426	122	85	85	106	77
Wuppertal 98	123	98	1 242	8 997	1 299	1 208	1 208	1 294	1 204
Wuppertal core	148	154	1 677	14 446	2 048	1 903	1 903	2 044	1 902
Wuppertal	1 582	311	13 202	79 251	3 188	2 886	2 886	3 150	2 862
Dutch	23	58	419	3 138	115	70	70	111	70
Potsdam	320	164	8 092	99 103	1 413	1 262	1 262	1 400	1 255

Table 2 Statistics on the computations. The columns list the instance name, the used cut separation, the solving time, the separation time, the number of separated cycle cuts, the number of cycle cuts selected by SCIP to be applied to the LP, the dual bound of the root node, the dual bound after termination, the best known primal bound, and the primal-dual gap in %.

name	method	solving time	sepa. time	cuts	applied cuts	root dual	dual	primal	gap in %
	no add. cuts	0.06s	-	-	-	16 231.80	24 074.55	24 074.55	0.00
	heuristic	0.04s	0.00s	2	2	16 499.35	24 074.55	24 074.55	0.00
Wuppertal 14	length ≤ 10	0.10s	0.03s	28	9	23 050.60	24 074.55	24 074.55	0.00
	length ≤ 15	0.18s	0.12s	84	16	23 088.89	24 074.55	24 074.55	0.00
	length ≤ 20	0.28s	0.22s	129	18	20 775.85	24 074.55	24 074.55	0.00
	no add. cuts	0.10s	-	-	-	28 778.74	37 755.40	37 755.40	0.00
Wuppertal 44	heuristic	0.11s	0.00s	1	1	28 669.75	37 755.40	37 755.40	0.00
	length ≤ 10	0.19s	0.05s	18	5	31 846.58	37 755.40	37 755.40	0.00
	length ≤ 15	0.46s	0.29s	40	9	31 953.26	37 755.40	37 755.40	0.00
	length ≤ 20	1.05s	0.80s	72	10	31 953.26	37 755.40	37 755.40	0.00
	no add. cuts	1h	-	-	-	81 940.30	112 023.51	477 161.17	325.95
Wuppertal 98	heuristic	1h	0.02s	20	20	89 284.15	124 697.64	468 467.85	275.68
	length ≤ 10	1h	1.16s	747	354	128 857.01	161 485.01	477 161.17	195.48
	length ≤ 15	1h	11.52s	2 413	887	149 847.94	173 291.01	477 161.17	175.35
	length ≤ 20	1h	41.34s	3 644	1 128	155 819.69	180 986.98	477 161.17	163.64
	no add. cuts	1h	-	-	-	98 654.95	118 462.71	464 533.25	292.13
Wuppertal core	heuristic	1h	0.02s	24	22	99 896.39	117 042.04	464 533.25	296.89
	length ≤ 10	1h	2.40s	949	448	137 337.00	155 433.06	464 533.25	198.86
	length ≤ 15	1h	25.90s	3 211	1 186	167 898.07	187 486.25	464 533.25	147.77
	length ≤ 20	1h	82.94s	4 886	1 488	175 122.92	184 265.70	464 533.25	153.09
	no add. cuts	1h	-	-	-	190 989.51	235 669.35	997 285.99	323.17
Wuppertal	heuristic	1h	0.08s	65	63	198 269.80	248 616.97	997 285.99	301.13
	length ≤ 10	1h	2.10s	1 082	402	232 178.52	273 620.80	997 285.99	264.48
	length ≤ 15	1h	21.55s	3 336	810	244 127.40	281 855.42	997 285.99	253.83
	length ≤ 20	1h	123.19s	5 307	1 098	255 288.10	290 249.68	997 285.99	243.60
	no add. cuts	7.06s	-	-	-	2 455.13	6 155.00	6 155.00	0.00
Dutch	heuristic	7.14s	0.00s	0	0	2 455.13	6 155.00	6 155.00	0.00
	length ≤ 10	7.99s	0.01s	0	0	2 455.13	6 155.00	6 155.00	0.00
	length ≤ 15	8.26s	0.04s	0	0	2 455.13	6 155.00	6 155.00	0.00
	length ≤ 20	8.24s	0.08s	0	0	2 455.13	6 155.00	6 155.00	0.00
	no add. cuts	1h	-	-	-	25 797.07	43 944.09	130 840.00	197.74
Potsdam	heuristic	1h	0.03s	10	10	28 407.66	46 545.79	130 840.00	181.10
	length ≤ 10	1h	0.34s	26	10	26 231.44	46 671.69	130 840.00	180.34
	length ≤ 15	1h	1.82s	106	33	27 115.22	45 784.24	130 840.00	185.76
	length ≤ 20	1h	8.04s	254	86	34 422.07	51 912.86	130 840.00	152.04

by Bussieck in the context of line planning [1]. The Potsdam instance is based on the real multi-modal public transportation network for 1998. We consider a period time of 20 for all instances. The activity weights are obtained by computing an uncapacitated multi-commodity flow in the event-activity network for a given passenger demand.

Our code is based on the constraint integer programming framework SCIP version 3.2.0 using Cplex 12.6.3 as an LP-solver. All computations were done on an Intel(R) Xeon(R) CPU E3-1245, 3.4 GHz computer (in 64 bit mode) with 8 MB cache, running Linux and 32 GB of memory. We set the time limit to one hour.

We compare the performance of the general MIP separators implemented in SCIP (*no add. cuts*), adding either the spanning-tree heuristic (*heuristic*) or our separation algorithm with a given length restriction (*length* ≤ 10, *length* ≤ 15, and *length* ≤ 20). The additional separators are only called at the root node. The results are listed in Table 2.

Looking at the root dual bound, one can see significant improvements, e.g., of up to 90% for Wuppertal 98, in comparison to the strategy without cycle cuts, and almost 75% over heuristic cycle cut separation. Hence, the separation algorithm has a greater effect on the dual bound than the heuristic, even though the separator only considers cycles of a restricted length. Only Wuppertal 14 has a smaller root dual bound if all cycles of maximum length 20 are separated compared to a cycle length of 10 or 15. This is not caused by the cycle inequalities, but by the additional “flow cover” and “strong cg” inequalities (heuristically) found by the default separator of SCIP. The given length restriction influences the performance of the separation algorithm: Separating cycle inequalities with higher length increases the computation time, but also, in general, the dual bound, especially for larger instances. In particular, the root dual bound for Potsdam can be further improved by 30% by using a length restriction of 20 compared to a length restriction of 10. Potsdam features the largest number of events, see Table 1, and benefits from a consideration of longer cycles.

References

1 Michael Bussieck. Gams – lop.gms: Line optimization. `http://www.gams.com/modlib/libhtml/lop.htm`.

2 M. Kümmling, P. Großmann, K. Nachtigall, J. Opitz, and R. Weiß. A state-of-the-art realization of cyclic railway timetable computation. *Public Transport*, 7(3):281–293, 2015. `doi:10.1007/s12469-015-0108-5`.

3 Christian Liebchen. *Periodic timetable optimization in public transport.* PhD thesis, Technische Universtität Berlin, 2006.

4 Christian Liebchen and Rolf H. Möhring. The modeling power of the periodic event scheduling problem: Railway timetables – and beyond. In Frank Geraets, Leo Kroon, Anita Schoebel, Dorothea Wagner, and Christos D. Zaroliagis, editors, *Algorithmic Methods for Railway Optimization*, volume 4359 of *Lecture Notes in Computer Science*, pages 3–40. Springer Berlin Heidelberg, 2007.

5 Christian Liebchen and Leon Peeters. Integral cycle bases for cyclic timetabling. *Discrete Optimization*, 6:98–109, 2009.

6 Christian Liebchen and Elmar Swarat. The Second Chvatal Closure Can Yield Better Railway Timetables. In Matteo Fischetti and Peter Widmayer, editors, *8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'08)*, volume 9 of *OpenAccess Series in Informatics (OASIcs)*, Dagstuhl, Germany, 2008. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

7 Thomas Lindner. *Train Schedule Optimization in Public Rail Transport.* PhD thesis, Technische Universtität Braunschweig, 2000.

8 K. Nachtigall and J. Opitz. Solving periodic timetable optimisation problems by modulo simplex calculations. In Matteo Fischetti and Peter Widmayer, editors, *ATMOS'08*, volume 9, 2008.

9 Karl Nachtigall. *Periodic Network Optimization and Fixed Interval Timetables.* Habilitation thesis, Universtität Hildesheim, 1998.

10 Michiel A. Odijk. Construction of periodic timetables, part 1: A cutting plane algorithm. Technical Report 94-61, TU Delft, 1994.

11 Leon W. P. Peeters. *Cyclic Railway and Timetable Optimization.* PhD thesis, Erasmus Universiteit Rotterdam, 2003.

12 Alexander Schrijver. Routing and timetabling by topological search. *Documenta Mathematica*, Extra Volume ICM 1998:1–9, 1998.

13 P. Sels, T. Dewilde, D. Cattrysse, and P. Vansteenwegen. Reducing the passenger travel time in practice by the automated construction of a robust railway timetable. *Transportation Research Part B: Methodological*, 84:124–156, 2016. `doi:10.1016/j.trb.2015.12.007`.

14 Paolo Serafini and Walter Ukovich. A mathematical model for periodic scheduling problems. *SIAM Journal on Discrete Mathematics*, 2(4):550–581, 1989.

Mapping Polygons to the Grid with Small Hausdorff and Fréchet Distance*

Quirijn W. Bouts[1], Irina Kostitsyna[2], Marc van Kreveld[3], Wouter Meulemans[4], Willem Sonke[5], and Kevin Verbeek[6]

1 **Dept. of Mathematics and Computer Science, TU Eindhoven, Eindhoven, The Netherlands**
`q.w.bouts@tue.nl`
2 **Dept. of Mathematics and Computer Science, TU Eindhoven, Eindhoven, The Netherlands**
`i.kostitsyna@tue.nl`
3 **Dept. of Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands**
`m.j.vankreveld@uu.nl`
4 **giCentre, City University London, London, United Kingdom**
`wouter.meulemans@city.ac.uk`
5 **Dept. of Mathematics and Computer Science, TU Eindhoven, Eindhoven, The Netherlands**
`w.m.sonke@tue.nl`
6 **Dept. of Mathematics and Computer Science, TU Eindhoven, Eindhoven, The Netherlands**
`k.a.b.verbeek@tue.nl`

Abstract

We show how to represent a simple polygon P by a (pixel-based) grid polygon Q that is simple and whose Hausdorff or Fréchet distance to P is small. For any simple polygon P, a grid polygon exists with constant Hausdorff distance between their boundaries and their interiors. Moreover, we show that with a realistic input assumption we can also realize constant Fréchet distance between the boundaries. We present algorithms accompanying these constructions, heuristics to improve their output while keeping the distance bounds, and experiments to assess the output.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases grid mapping, Hausdorff distance, Fréchet distance, digital geometry

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.22

1 Introduction

Transforming the representation of objects from the real plane onto a grid has been studied for decades due to its applications in computer graphics, computer vision, and finite-precision computational geometry [14]. Two interpretations of the grid are possible: (i) the grid graph, consisting of vertices at all points with integer coordinates, and horizontal and vertical edges

* Research on the topic of this paper was initiated at the 1st Workshop on Applied Geometric Algorithms (AGA 2015) in Langbroek, The Netherlands, supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 639.023.208. NWO is supporting Q. W. Bouts, I. Kostitsyna, and W. Sonke under project no. 639.023.208, and K. Verbeek under project no. 639.021.541. W. Meulemans is supported by Marie Skłodowska-Curie Action MSCA-H2020-IF-2014 656741.

24th Annual European Symposium on Algorithms (ESA 2016).
Editors: Piotr Sankowski and Christos Zaroliagis;
Article No. 22; pp. 22:1–22:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

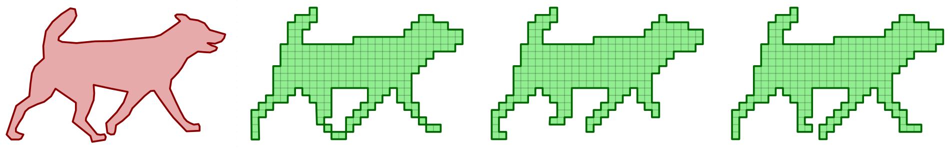


Figure 1 From left to right: input; symmetric-difference optimal result is not a grid polygon; grid polygon computed by our Fréchet algorithm; grid polygon computed by our Hausdorff algorithm.

between vertices at unit distance; (ii) the pixel grid, where the only elements are pixels (unit squares). In the latter, one can choose between 4-neighbor or 8-neighbor grid topology. In this paper we adopt the pixel grid view with 4-neighbor topology.

The issues involved when moving from the real plane to a grid begin with the definition of a line segment on a grid, known as a *digital straight segment* [18]. For example, it is already difficult to represent line segments such that the intersection between any pair is a connected set (or empty). In general, the challenge is to represent objects on a grid in such a way that certain properties of those objects in the real plane transfer to related properties on the grid; connectedness of the intersection of two line segments is an example of this.

While most of the research related to *digital geometry* has the graphics or vision perspective [17, 18], computational geometry has made a number of contributions as well. Besides finite-precision computational geometry [12, 14] these include snap rounding [11, 13, 16], the integer hull [4, 15], and consistent digital rays with small Hausdorff distance [10].

Mapping polygons. We consider the problem of representing a simple polygon P as a similar polygon in the grid (see Fig. 1). A *grid cycle* is a simple cycle of edges and vertices of the grid graph. A *grid polygon* is a set of pixels whose boundary is a grid cycle. This problem is motivated by schematization of country or building outlines and by nonograms.

The most well-known form of schematization in cartography is called a metro map, in which metro lines are shown in an abstract manner by polygonal lines whose edges typically have only four orientations. It is common to also depict region outlines with these orientations on such maps. It is possible to go one step further in schematization by using only integer coordinates for the vertices, which often aligns vertices vertically or horizontally, and leads to a more abstracted view. Certain types of cartograms like mosaic maps [8] are examples of maps following this visualization style. The version based on a square grid is often used to show electoral votes after elections. Another cartographic application of grid polygons lies in the schematization of building outlines [20].

Nonograms – also known as Japanese or picture logic puzzles – are popular in puzzle books, newspapers, and in digital form. The objective is to reconstruct a pixel drawing from a code that is associated with every row and column. The algorithmic problem of solving these puzzles is well-studied and known to be NP-complete [6]. To generate a nonogram from a vector drawing, a grid polygon needs to be made on a coarse grid. We are interested in the generation of grid polygons from shapes like animal outlines, which could be used to construct nonograms. To our knowledge, two papers address this problem. Ortíz-García et al. [22] study the problem of generating a nonogram from an image; both the black-and-white and color versions are studied. Their approach uses image processing techniques and heuristics. Batenburg et al. [5] also start with an image, but concentrate on generating nonograms from an image with varying difficulty levels, according to some definition of difficulty.

Considering the above, our work also relates to image downscaling (e.g. [19]), though this usually starts from a raster image instead of continuous geometric objects. Kopf et al. [19]

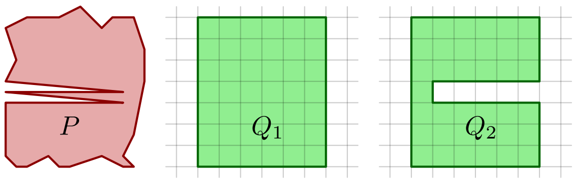


Figure 2 $d_H(P, Q_1)$ is small but $d_H(\partial P, \partial Q_1)$ is not. $d_H(P, Q_2)$ and $d_H(\partial P, \partial Q_2)$ are both small but the Fréchet distance $d_F(\partial P, \partial Q_2)$ is not.

apply their technique to vector images, stating that the outline remains connected where possible. In contrast to our work, the quality is not measured as the geometric similarity and the conditions necessary to guarantee a connected outline remain unexplored.

Similarity. There are at least three common ways of defining the similarity of two simple polygons: the symmetric difference[1], the Hausdorff distance [1], and the Fréchet distance [2]. The first does not consider similarity of the polygon boundaries, whereas the third usually applies to boundaries only. The Hausdorff distance between polygon interiors and between polygon boundaries both exist and are different measures; this distance can be directed or undirected. Let X and Y be two closed subsets of a metric space. The (directed) *Hausdorff distance* $d_H(X, Y)$ from X to Y is defined as the maximum distance from any point in X to its closest point in Y. The undirected version is the maximum of the two directed versions. To define the *Fréchet distance*, let X and Y be two curves in the plane. The Fréchet distance $d_F(X, Y)$ is the minimum leash length needed to let a man walk over X and a dog over Y, where neither may walk backwards (a formal definition can be found in [2]).

Contributions. In Section 2 we show that any simple polygon P admits a grid polygon Q with $d_H(P, Q) \leq \frac{1}{2}\sqrt{2}$ and $d_H(Q, P) \leq \frac{3}{2}\sqrt{2}$ on the unit grid. Furthermore, the constructed polygon satisfies the same bounds between the boundaries ∂P and ∂Q. This is not equivalent, since the point that realizes the maximum smallest distance to the other polygon may lie in the interior (Fig. 2). Our proof is constructive, but the construction often does not give intuitive results (Fig. 2, P and Q_2). Therefore, we extend our construction with heuristics that reduce the symmetric difference whilst keeping the Hausdorff distance within $\frac{3}{2}\sqrt{2}$. The Fréchet distance d_F [2] between two polygon boundaries is often considered to be a better measure for similarity. Unlike the Hausdorff distance, however, not every polygon boundary ∂P can be represented by a grid cycle with constant Fréchet distance. In Section 3 we present a condition on the input polygon boundary related to fatness (in fact, to κ-straightness [3]) and show that it allows a grid cycle representation with constant Fréchet distance. Finally, in Section 4 we evaluate how our algorithms perform on realistic input polygons.

2 Hausdorff distance

We consider the problem of constructing a grid polygon Q with small Hausdorff distance to P. Though minimizing the Hausdorff distance is NP-hard (Theorem 1, see [7]), we present an algorithm that achieves low, constant Hausdorff distance between both the boundaries and the interiors of the input polygon P and the resulting grid polygon Q. We first show

[1] The symmetric difference between two sets A and B is defined as the set $(A \setminus B) \cup (B \setminus A)$. When using symmetric difference as a quality measure, we actually mean the area of the symmetric difference.

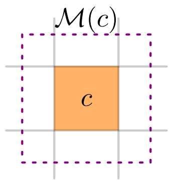


Figure 3 Module $\mathcal{M}(c)$ (dashed) of a cell c.

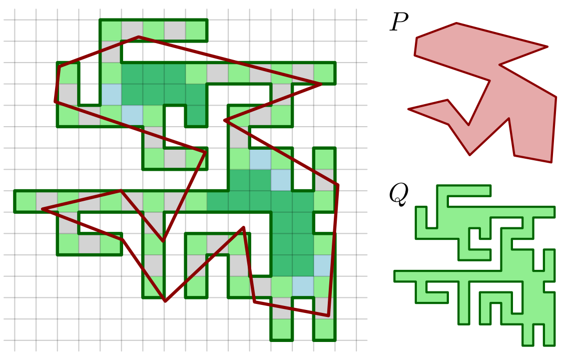


Figure 4 Example of the Hausdorff algorithm; the input and output are shown on the right. Colors: 🟩 Q_1, 🟩 Q_2, 🟦 Q_3, ⬜ Q_4.

how to construct such a grid polygon. Then, we provide an efficient algorithm to compute Q. Finally, we describe heuristics that can be used to improve the results in practice.

▶ **Theorem 1.** *Given a polygon P, it is NP-hard to decide whether there exists a grid polygon Q such that both $d_H(\partial P, \partial Q) \leq \frac{1}{2}$ and $d_H(\partial Q, \partial P) \leq \frac{1}{2}$.*

2.1 Construction

We represent the grid polygon Q as a set of cells (or pixels). We say that two cells are *adjacent* if they share a segment. If two cells share only a point, then they are *point-adjacent*. If two cells $c_1 \in Q$ and $c_2 \in Q$ are point-adjacent, and there is no cell $c \in Q$ that is adjacent to both c_1 and c_2, then c_1 and c_2 share a *point-contact*. We construct Q as the union of four sets Q_1, Q_2, Q_3, Q_4 (not necessarily disjoint). To define these sets, we define the *module* $\mathcal{M}(c)$ of a cell c as the 2×2-region centered at the center of c (see Fig. 3). Furthermore, we assume the rows and columns are numbered, so we can speak of even-even cells, odd-odd cells, odd-even cells, and even-odd cells. The four sets are defined as follows; see also Fig. 4.

Q_1: All cells c for which $\mathcal{M}(c) \subseteq P$.
Q_2: All even-even cells c for which $\mathcal{M}(c) \cap P \neq \emptyset$.
Q_3: For all cells $c_1, c_2 \in Q_1 \cup Q_2$ that share a point-contact, the two cells that are adjacent to both c_1 and c_2 are in Q_3.
Q_4: A minimal set of cells that makes Q connected, and where each cell $c \in Q_4$ is adjacent to two cells in Q_2 and $\mathcal{M}(c) \cap P \neq \emptyset$.

Set $Q_1 \cup Q_2$ is sufficient to achieve the desired Hausdorff distance. We add Q_3 to resolve point-contacts, and Q_4 to make the set Q simply connected (a polygon without holes). The lemmas below show that Q is indeed a grid polygon.

▶ **Lemma 2.** *The set $Q_1 \cup Q_2$ is hole-free, even when including point-adjacencies.*

Proof. For the sake of contradiction, let H be a maximal set of cells comprising a hole. Let set B contain all cells in $Q_1 \cup Q_2$ that surround H and are adjacent to a cell in H. Since Q_2 contains only even-even cells, every cell in $Q_2 \cap B$ is (point-)adjacent to two cells in $Q_1 \cap B$ (see Fig. 5). Hence, the outer boundary of the union of all modules of cells in $Q_1 \cap B$ is a single closed curve C. Since $C \subset P$ by the definition of Q_1, the interior of C must also be in P. Since all modules of cells in H lie completely inside C, they are also in P, so the cells in H must all be in Q_1. This contradicts that H is a hole. ◀

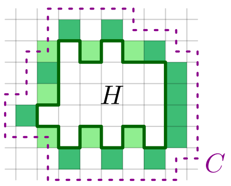


Figure 5 A hole in Q. Colors: 🟩 $Q_1 \cap B$; 🟩 $Q_2 \cap B$.

▶ **Lemma 3.** *The set Q is simply connected and does not contain point-contacts.*

Proof. Consider a point-contact between two cells $c_1, c_2 \in Q_1 \cup Q_2$ and a cell $c \notin Q_1 \cup Q_2$ that is adjacent to both c_1 and c_2 (so $c \in Q_3$). Since Q_2 contains only even-even cells, we may assume that $c_1 \in Q_1$. Recall that $\mathcal{M}(c_1) \subseteq P$ by definition. We may further assume that c_1 is an odd-odd cell, for otherwise a cell in Q_2 would eliminate the point-contact. Hence, all cells point-adjacent to c_1 are in $Q_1 \cup Q_2$, and thus c has three adjacent cells in $Q_1 \cup Q_2$. This implies that adding $c \in Q_3$ to $Q_1 \cup Q_2$ cannot introduce point-contacts or holes. Similarly, cells in Q_4 connect two oppositely adjacent cells in Q_2, and thus cannot introduce point-contacts (or holes, by definition). Combining this with Lemma 2 implies that Q is hole-free and does not contain point-contacts.

It remains to show that Q is connected, that is, the set Q_4 exists. Consider two cells $c_1, c_2 \in Q$. We show that c_1 and c_2 are connected in Q. We may further assume that $c_1, c_2 \in Q_2$, as cells in $Q_1 \cup Q_3 \cup Q_4$ must be adjacent or point-adjacent to a cell in Q_2. Let $p \in \mathcal{M}(c_1) \cap P$, $q \in \mathcal{M}(c_2) \cap P$ and consider a path π between p and q inside P. Every even-even cell c with $\mathcal{M}(c) \cap \pi \neq \emptyset$ must be in Q_2. Furthermore, the modules of even-even cells cover the plane. Every cell connecting a consecutive pair of even-even cells intersecting π satisfies the conditions of Q_4, and thus can be added to make c_1 and c_2 connected in Q. ◀

Upper bounds. To prove our bounds, note that $\mathcal{M}(c) \cap P \neq \emptyset$ for every cell $c \in Q$. This is explicit for cells in Q_1, Q_2, and Q_4. For cells in Q_3, note that these cells must be adjacent to a cell in Q_1, and thus contain a point in P.

▶ **Lemma 4.** $d_H(P, Q)$, $d_H(\partial P, \partial Q) \leq \frac{1}{2}\sqrt{2}$.

Proof. Let $p \in P$ and consider the even-even cell c such that $p \in \mathcal{M}(c)$. Since $c \in Q_2$, the distance $d_H(p, Q) \leq d_H(p, c) \leq \frac{1}{2}\sqrt{2}$. Now consider a point $p \in \partial P$. There is a 2×2-set of cells whose modules contain p. This set contains an even-even cell $c \in Q$ and an odd-odd cell $c' \notin Q$. The latter is true, because odd-odd cells in Q must be in Q_1. Therefore, the point q shared by c and c' must be in ∂Q. Thus, $d_H(p, \partial Q) \leq d_H(p, q) \leq \frac{1}{2}\sqrt{2}$. ◀

▶ **Lemma 5.** $d_H(Q, P)$, $d_H(\partial Q, \partial P) \leq \frac{3}{2}\sqrt{2}$.

Proof. Let q be a point in Q and let $c \in Q$ be the cell that contains q. Since $\mathcal{M}(c) \cap P \neq \emptyset$, we can choose a point $p \in \mathcal{M}(c) \cap P$. It directly follows that $d_H(q, P) \leq d_H(q, p) \leq \frac{3}{2}\sqrt{2}$. Now consider a point $q \in \partial Q$, and let $c \in Q$ and $c' \notin Q$ be two adjacent cells such that $q \in \partial c \cap \partial c'$. We claim that $(\mathcal{M}(c) \cup \mathcal{M}(c')) \cap \partial P \neq \emptyset$. If $c \notin Q_1$, then $\mathcal{M}(c) \not\subseteq P$. As furthermore $\mathcal{M}(c) \cap P \neq \emptyset$, we have that $\mathcal{M}(c) \cap \partial P \neq \emptyset$. On the other hand, if $c \in Q_1$, then $\mathcal{M}(c) \subseteq P$, so $\mathcal{M}(c') \cap P \neq \emptyset$. As furthermore $\mathcal{M}(c') \not\subseteq P$ (otherwise $c' \in Q_1$), we have that $\mathcal{M}(c') \cap \partial P \neq \emptyset$. Let $p \in (\mathcal{M}(c) \cup \mathcal{M}(c')) \cap \partial P$. Then $d_H(q, \partial P) \leq d_H(q, p) \leq \frac{3}{2}\sqrt{2}$. ◀

▶ **Theorem 6.** *For every simple polygon P a simply connected grid polygon Q without point-contacts exists such that $d_H(P, Q)$, $d_H(\partial P, \partial Q) \leq \frac{1}{2}\sqrt{2}$ and $d_H(Q, P)$, $d_H(\partial Q, \partial P) \leq \frac{3}{2}\sqrt{2}$.*

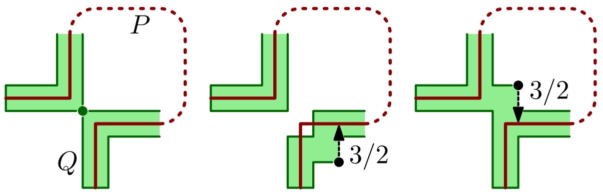


Figure 6 A polygon that does not admit a grid polygon with Hausdorff distance smaller than 3/2. The brown line signifies an infinitesimally thin polygon.

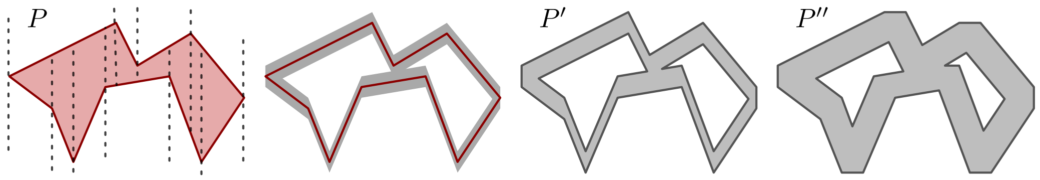


Figure 7 A simple polygon P with its vertical decomposition, and the construction of P' and P''.

Lower bound. Fig. 6 illustrates a polygon P for which no grid polygon Q exists with low $d(Q, P)$. A naive construction results in a nonsimple polygon (left). To make it simple, we can either remove a cell (center) or add a cell (right). Both methods result in $d_H(Q, P) \geq 3/2 - \epsilon$. Alternatively, we can fill the entire upper-right part of the grid polygon (not shown), resulting in a high $d_H(Q, P)$. This leads to the following theorem.

▶ **Theorem 7.** *For any $\epsilon > 0$, there exists a polygon P for which no grid polygon Q exists with $d(Q, P) < 3/2 - \epsilon$.*

In the L_∞ metric, the lower bound of $3/2 - \epsilon$ given in Fig. 6 also holds. A straightforward modification of the upper-bound proofs can be used to show that the Hausdorff distance is at most 3/2 in the L_∞ metric. In other words, our bounds are tight under the L_∞ metric.

2.2 Algorithm

To compute a grid polygon for a given polygon P with n edges, we need to determine the cells in the sets Q_1–Q_4. This is easy once we know which cells intersect ∂P. One way to do this is to trace the edges of P in the grid. The time this takes is proportional to the number of crossings between cells and ∂P. Let us denote the number of grid cells that intersect ∂P by b. Clearly, there are simple polygons with $\Theta(nb)$ polygon boundary-to-cell crossings. We show how to achieve a time bound of $O(n + B)$, where B is the number of cells in the output. The key idea is to first compute the Minkowski sum of ∂P with a square of side length 2 and use that to quickly find the cells intersecting ∂P.

To compute this Minkowski sum we first compute the vertical decomposition of ∂P, see Fig. 7. For every of the $O(n)$ quadrilaterals, determine the parts that are within vertical distance 1 from the bounding edges. The result P' is a simple polygon with holes with a total of $O(n)$ edges, and $\partial P \subset P'$. We compute the horizontal decomposition of every hole and the exterior of P' and determine all parts that are within horizontal distance 1 from the bounding edges. We add this to P', giving P''. These steps take $O(n)$ time if we use Chazelle's triangulation algorithm [9]. Essentially, the above steps constitute computing the Minkowski sum of ∂P with a square of side length 2, centered at the origin and axis-parallel.

▶ **Lemma 8.** *For any cell c, at most four edges of P'' intersect its boundary twice.*

Proof. For any edge of P'', by construction, the whole part vertically above or below it over distance at least 2 is inside P'', and the same is true for left or right. For any edge e that intersects the boundary of c twice, one side of that edge is fully in the interior of P'', and hence, cannot contain other edges of P''. Hence, e can be charged uniquely to a corner of c. ◀

▶ **Corollary 9.** *The number of polygon boundary-to-cell crossings of P'' is $O(n+b)$, where b is the number of grid cells intersecting ∂P.*

By tracing the boundary of P'', we can identify all cells that intersect it. Then we can determine all cells that intersect the boundary of P, because these are the cells that lie fully inside P''. The modifications needed to find all cells whose module lies inside P are straightforward. In particular, we can find all cells whose module lies inside P, but have a neighbor for which this is not the case in $O(n+b)$ time. This allows us to find the $O(B)$ cells selected in step Q_1 in $O(n+B)$ time. Steps Q_2 and Q_3 are now straightforward as well.

We now have a number of connected components of chosen grid cells. No component has holes, and if there are k components, we can connect them into one with only $k-1$ extra grid cells. We walk around the perimeter of some component and mark all non-chosen cells adjacent to it. If a cell is marked twice, it is immediately removed from consideration. Cells that are marked once but are adjacent to two chosen cells will merge two different components. We choose one of them, then walk around the perimeter of the new part and mark the adjacent cells. Again, cells that are marked twice (possibly, both times from the new part, or once from the old and once from the new part) are removed from consideration. Continuing this process unites all components without creating holes.

▶ **Theorem 10.** *For any simple polygon P with n edges, we can determine a set of B cells that together form a grid polygon Q in $O(n+B)$ time, such that $d_H(P,Q)$, $d_H(\partial P, \partial Q) \le \frac{1}{2}\sqrt{2}$ and $d_H(Q,P)$, $d_H(\partial Q, \partial P) \le \frac{3}{2}\sqrt{2}$.*

2.3 Heuristic improvements

The grid polygon Q constructed in Section 2.1 does not follow the shape of P closely (see Fig. 4). Although the boundary of Q remains close to the boundary of P, it tends to zigzag around it due to the way it is constructed. As a result, the symmetric difference between P and Q is relatively high. We consider two modifications of our algorithm to reduce the symmetric difference between P and Q while maintaining a small Hausdorff distance:

1. We construct Q_4 with symmetric difference in mind.
2. We post-process the resulting polygon Q by adding, removing, or shifting cells.

Construction of Q_4. Instead of picking cells arbitrarily when constructing Q_4 we improve the construction with two goals in mind: (1) to directly reduce the symmetric difference between P and Q, and (2) to enable the post-processing to be more effective. To that end, we construct Q_4 by repeatedly adding the cell c (not introducing holes) that has the largest overlap with P. These cells are the ones that reduce the symmetric difference between P and Q the most.

Post-processing. After computing the grid polygon Q, we allow three operations to reduce the symmetric difference: (1) adding a cell, (2) removing a cell, and (3) shifting a cell to a neighboring position. These operations are applied iteratively until there is no operation that can reduce the symmetric difference. Every operation must maintain the following conditions:

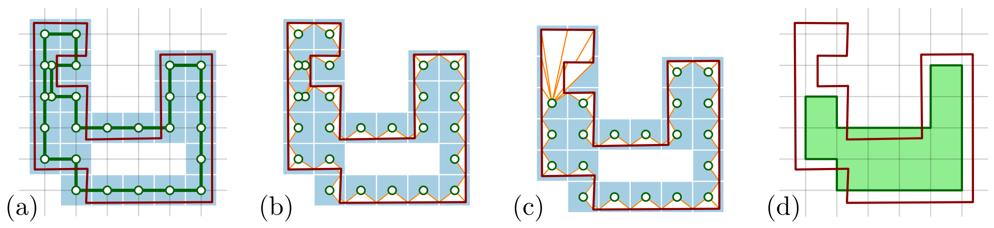


Figure 8 Constructing Q for the upper bound on the Fréchet distance. (a) Input polygon on the grid and the squares it visits (shaded); initial state of C with revisited vertices slightly offset for legibility. (b) Initial mapping μ (white triangles) between the vertices of C and ∂P. (c) Removal of duplicate vertices in C, and its effect on μ. (d) Resulting cycle represents a grid polygon.

(1) Q is simply connected, and (2) the Hausdorff distance between P (∂P) and Q (∂Q) is small. For the second condition we allow a slight relaxation with regard to the bounds of Lemma 4: $d_H(P, Q)$ and $d_H(\partial P, \partial Q)$ can be at most $\frac{3}{2}\sqrt{2}$ (like $d_H(Q, P)$ and $d_H(\partial Q, \partial P)$). This relaxation gives the post-processing more room to reduce the symmetric difference.

3 Fréchet distance

The Fréchet distance d_F between two curves is generally considered a better measure for similarity than the Hausdorff distance. For an input polygon P, we consider computing a grid polygon Q such that $d_F(\partial P, \partial Q)$ is bounded by a small constant. We study under what conditions on ∂P this is possible and prove an upper and lower bound. However, if ∂P zigzags back and forth within a single row of grid cells, any grid polygon must have a large Fréchet distance: the grid is too coarse to follow ∂P closely. To account for this in our analysis, we introduce a realistic input model, as explained below.

Narrow polygons. For $a, b \in \partial P$, we use $|ab|_{\partial P}$ to denote the perimeter distance, i.e., the shortest distance from a to b along ∂P. We define *narrowness* as follows.

▶ **Definition 11.** A polygon P is (α, β)*-narrow*, if for any two points $a, b \in \partial P$ with $|ab| \leq \alpha$, $|ab|_{\partial P} \leq \beta$.

Given a value for α, we refer to the minimal β as the α-narrowness of a polygon. We assume $\alpha < \beta$, to avoid degenerately small polygons. We note that narrowness is a more forgiving model than straightness [3]. A polygon P is κ*-straight* if for any two points $a, b \in \partial P$, $|ab|_{\partial P} \leq \kappa \cdot \|a - b\|$. A κ-straight polygon is $(\alpha, \kappa\alpha)$-narrow for any α, but not the other way around. In particular, a finite polygon that intersects itself (or comes infinitesimally close to doing so) has a bounded narrowness, whereas its straightness becomes unbounded.

Upper bound. With our realistic input model in place, we can bound the Fréchet distance needed for a grid polygon from above. In particular, we prove the following theorem.

▶ **Theorem 12.** *Given a* $(\sqrt{2}, \beta)$*-narrow polygon* P *with* $\beta \geq \sqrt{2}$*, there exists a grid polygon* Q *such that* $d_F(\partial P, \partial Q) \leq (\beta + \sqrt{2})/2$.

Proof. To prove the claimed upper bound, we construct Q via a grid cycle C that defines ∂Q. The construction is illustrated in Fig. 8. We define the *square* of a grid-graph vertex v to be the 1×1-square centered on v. Let C be the cyclic chain of vertices whose square is intersected by ∂P, in the order in which ∂P visits them. We define a mapping μ between the

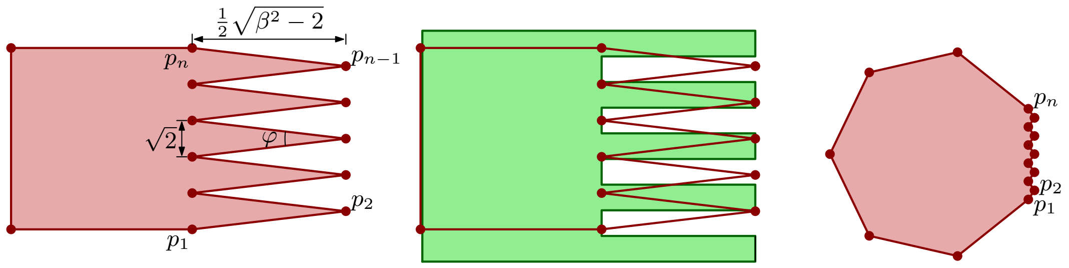


Figure 9 Polygon P (left) for which any grid polygon will have high Fréchet distance (center); polygon P for $\beta < 2$ (right).

vertices of C and ∂P. In particular, for each $c \in C$, let $\mu(c)$ be the "visit" of ∂P that led to c's existence in C, that is, the part of ∂P within the square of c. By construction, we have that $\|c - p_c\| \leq \sqrt{2}/2$ for all $c \in C$ and $p_c \in \mu(c)$. The visits $\mu(c)$ and $\mu(c')$ for two consecutive vertices, c and c', in C intersect in a point (or, in degenerate cases, in a line segment) that lies on the common boundary of the squares of c and c'; let p denote such a point. For any point σ on the line segment between c and c', we have that $\|\sigma - p\| \leq \max\{\|c-p\|, \|c'-p\|\} \leq \sqrt{2}/2$, as the Euclidean distance is convex (i.e., its unit disk is a convex set). Hence, μ describes a continuous mapping on ∂P and acts as a witness for $d_F(\partial P, C) \leq \sqrt{2}/2$.

However, C may contain duplicates and thus not describe a grid polygon Q. We argue here that we can remove the duplicates and maintain μ in such a way that it remains a witness to prove that $d_F(\partial P, C) \leq (\beta + \sqrt{2})/2$. Let c and c' be two occurrences in C of the same vertex v. Let $p \in \mu(c)$ and $p' \in \mu(c')$, both in the square of v. As they lie within the same square, $\|p - p'\| \leq \sqrt{2}$ and hence we know that $|pp'|_{\partial P} \leq \beta$. Hence, at least one of the two subsequences of C strictly in between c and c' maps via μ to a part of ∂P that has length at most β. We pick one such subsequence and remove it as well as c' from C. We concatenate to $\mu(c)$ the mapped parts of ∂P from the removed vertices. As the length of the mapped parts is bounded by β, the maximal distance between any point on these mapped parts is $\beta/2 + \sqrt{2}/2$. Hence, after removing all duplicates, we are left with a cycle C, with μ as a witness to testify that $d_F(\partial P, C) \leq (\beta + \sqrt{2})/2$.

If C contains at least three vertices, it describes a grid polygon and we are done. However, if C consists of at most two vertices, then it does not describe a grid polygon. We can extend C easily into a 4-cycle for which the bound still holds (see [7] for details). ◀

The proof of the theorem readily leads to a straightforward algorithm to compute such a grid polygon. The construction poses no restrictions on the order in which to remove duplicates and the decisions are based solely on the lengths of $\mu(v)$. Hence, the algorithm runs in linear time by walking over P to find C and handling duplicates as they arise.

Lower bound. To show a lower bound, we construct a $(\sqrt{2}, \beta)$-narrow polygon P for which there is no grid polygon with Fréchet distance smaller than $\frac{1}{4}\sqrt{\beta^2-2}$ to P, for any $\beta > \sqrt{2}$. First, construct a polygonal line $L = (p_1, \ldots, p_n)$, where $n = 2\left\lceil \frac{1}{4}\sqrt{\beta^2-2} \right\rceil + 1$. Vertex p_i is $(0, i/2)$ if i is odd and $(\frac{1}{2}\sqrt{\beta^2-2},\ i/\sqrt{2})$ otherwise. Now, consider a regular k-gon with side length $(n-1)/\sqrt{2}$ and $k \geq 4$ such that its interior angles are at least $\varphi = \arccos(1 - 4/\beta^2)$. Assume the k-gon has a vertical edge on the right-hand side. We replace this edge by L to construct our polygon P. Fig. 9 shows a polygon for $k = 4$ ($\beta \geq 2$) and for $k = 7$ ($\beta < 2$).

The two lemmas below readily imply our lower bound on the Fréchet distance. We omit proof of the first, but details can be found in [7].

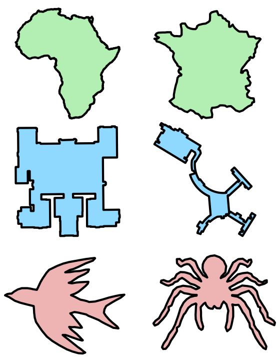


Figure 10 The input categories.

▶ **Lemma 13.** *The constructed polygon P described above is $(\sqrt{2}, \beta)$-narrow.*

▶ **Lemma 14.** *For constructed polygon P and any grid polygon Q, $d_F(\partial P, \partial Q) \geq \frac{1}{4}\sqrt{\beta^2 - 2}$.*

Proof. We show this by contradiction: assume that a grid polygon Q exists with $d_F(\partial P, \partial Q) = \varepsilon < \frac{1}{4}\sqrt{\beta^2 - 2}$. For any vertex p_i of P, there must be a point $q_i \in \partial Q$ (not necessarily a vertex) such that $\|p_i - q_i\| < \varepsilon$. Moreover, these points $q_1, \ldots, q_n$ need to appear on ∂Q in order. Equivalently, if we draw disks with radius ε centered at $p_1, \ldots, p_n$, curve ∂Q needs to visit these disks in order.

The disks centered at $p_1, p_3, \ldots, p_n$ never intersect the disks centered at $p_2, p_4, \ldots, p_{n-1}$. In particular, the disks centered at $p_1, p_3, \ldots, p_n$ are all to the left of the vertical line $v \colon x = \frac{1}{4}\sqrt{\beta^2 - 2}$, and all disks centered at $p_2, p_4, \ldots, p_{n-1}$ are all to the right of this line. Hence, between q_1 and q_2, ∂Q must contain at least one horizontal line segment crossing line v to the right, and between q_2 and q_3 there must be at least one horizontal segment crossing v to the left, and so on until we reach q_n. Since Q is simple, this requires that the difference between the maximum and the minimum y-coordinate of the these horizontal segments on ∂Q is at least $n - 1$. The y-difference between p_1 and p_n is only $(n-1)/\sqrt{2}$. This implies $d_F(\partial P, \partial Q) \geq n - 1 - (n-1)/\sqrt{2} > \frac{1}{4}\sqrt{\beta^2 - 2}$ and thus contradicts our assumption. ◀

▶ **Theorem 15.** *For any $\beta > \sqrt{2}$, there exists a $(\sqrt{2}, \beta)$-narrow polygon P such that $d_F(\partial P, \partial Q) \geq \frac{1}{4}\sqrt{\beta^2 - 2}$ holds for any grid polygon Q.*

4 Experiments

Here, we apply our algorithms to a set of polygons that can be encountered in practice. We investigate the performance of the Hausdorff algorithm and its heuristics as well as the Fréchet algorithm. Moreover, we consider the effects of grid resolution and the placement of the input. Full details on the experiments can be found in [7].

Data set. We use a set of 34 polygons: 14 territorial outlines (countries, provinces, islands), 11 building footprints and 9 animal silhouettes (see Fig. 10 for six examples). We scale all input polygons such that their bounding box has area r; we call r the resolution. Unless stated otherwise, we use $r = 100$. This scaling is used to eliminate any bias introduced from comparing different resolutions.

Table 1 Normalized symmetric difference, as an increase percentage w.r.t. optimal, of the algorithms. Note that "optimal" here means optimal for the symmetric difference when not insisting on a connected set of cells. For the Hausdorff algorithm, results for the various heuristic improvements are shown. In the second row, *None* means that no postprocessing heuristic was used; A, R and S mean additions, removals and shifts, respectively. In the third row, ✓ and ✗ indicate whether Q_4 was chosen arbitrarily (✗) or using the symmetric difference heuristic (✓).

	Optimal	**Hausdorff**						**Fréchet**
postproc.		*None*		A/R		$A/R/S$		
Q_4 *heur.*		✗	✓	✗	✓	✗	✓	
Maps	0.223	+316 %	+238 %	+39 %	+3 %	+11 %	+3 %	+23 %
Buildings	0.257	+270 %	+197 %	+47 %	+9 %	+21 %	+8 %	+17 %
Animals	0.333	+246 %	+188 %	+60 %	+12 %	+29 %	+11 %	+8 %

4.1 Symmetric difference

We start our investigation by measuring the symmetric difference between the input and output polygon. If the symmetric difference is small, this indicates that the output is similar to the input. We normalize the symmetric difference by dividing it by the area of the input polygon. The results of our algorithms depend on the position of the input polygon relative to the grid. Hence, for every input polygon we computed the average normalized symmetric difference over 20 random placements.

Computing a (simply connected) grid polygon that minimizes symmetric difference is NP-hard [21]. Hence, as a baseline for our comparison, we compute the set of cells with the best possible symmetric difference by simply taking all cells that are covered by the input polygon for at least 50 %. This set of cells is optimal with respect to symmetric difference but may not be simply connected. It can hence be thought of as a lower bound.

Overview. In Table 1, we compare the Fréchet algorithm and the various instantiations of the Hausdorff algorithm in terms of the (normalized) symmetric difference. The second column lists the average symmetric difference of the symmetric-difference optimal solution, calculated as described above. The other columns are hence given as a percentage representing the increase with respect to the optimal value. We aggregated the results per input type.

The table tells us that, with the use of heuristics, the Hausdorff algorithm gets quite close to the optimal symmetric difference, while still bounding the Hausdorff distance and guaranteeing a grid polygon. The Fréchet algorithm is performing more poorly in comparison, though interestingly performs *better* on the animal contours.

Fig. 11 shows three solutions for one of the input polygons: symmetric-difference optimal, Fréchet algorithm and Hausdorff algorithm with heuristics. The symmetric-difference optimal solution looks like the input, but consists of multiple disconnected polygons. The result of the Fréchet algorithm is a single grid polygon, but the algorithm cuts off narrow parts. The result of the Hausdorff algorithm is also a single grid polygon, but does not have to cut off parts when input is narrow.

Below, we examine the effect of the different heuristics for the Hausdorff algorithm to explain their success. Moreover, we show that the performance of the Fréchet algorithm is highly dependent on the grid resolution.

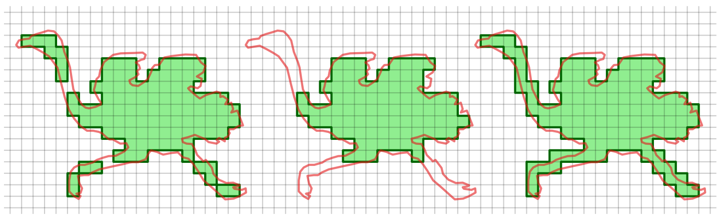


Figure 11 Example outputs for the symmetric-difference optimal algorithm (left), the Fréchet algorithm (center) and the Hausdorff algorithm (right). Note that the first does not yield a grid polygon.

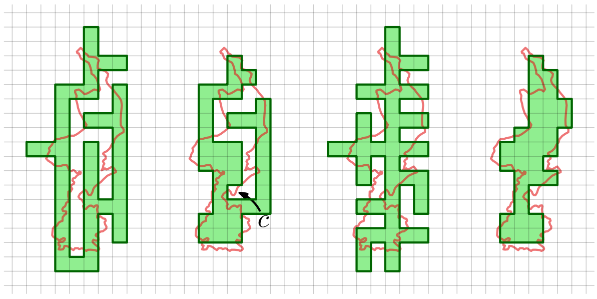


Figure 12 Without the heuristic for the Q_4 construction (a), the algorithm gets stuck in the post processing phase (b). The smart Q_4 construction gives a better starting point (c) resulting in the desired shape (d).

Hausdorff heuristics. Table 1 shows that using the heuristic for Q_4 makes a tremendous difference, especially if a postprocessing heuristic is used as well. Fig. 12 illustrates this finding with four results on the same input. In (a–b) Q_4 is chosen arbitrarily and the resulting shape does not look like the input – even after postprocessing. In particular, the postprocessing heuristic cannot progress further: the cell marked c cannot be added to Q since that would increase the symmetric difference. In (c–d) Q_4 is chosen using the heuristic; it provides a better initial solution which allows the postprocessing to create a nice result.

In the postprocessing heuristic, allowing or disallowing shifts can influence the result. See for example Fig. 13. Without shifts, the heuristic cannot move the connection between the two ends of the input polygon to the correct location as it would first need to increase the symmetric difference. With a series of diagonal shifts this can be achieved. Our experiments show that in practice allowing shifts indeed decreases the symmetric difference. However, the effect is only marginal if we use the heuristic for the Q_4 construction. Hence, we conclude that shifts only significantly improve the result if Q_4 is chosen badly.

Resolution and placement. While developing our algorithm we noticed that not just the grid resolution but also the placement of the input polygon effected the symmetric difference. Hence we set up experiments to investigate these factors. First we tested how much the resolution influences the symmetric differences. In Table 2, the results are shown, averaged over all 34 inputs. As expected, for all algorithms, the normalized symmetric difference decreases when the resolution increases.

To investigate how much the results of our algorithms depend on the input placement, we compared the minimal, maximal and average symmetric difference over 20 runs of the algorithms. The polygons were placed randomly for each run, but per polygon the same 20 positions were used for all three algorithms. We found that the difference between the

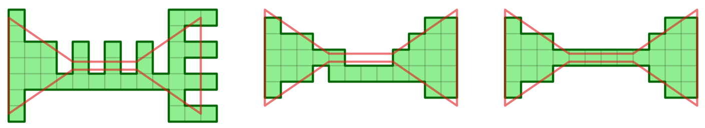


Figure 13 Without allowing shifts, the post-processing phase cannot move the cells in the middle to coincide with the input polygon. With shifts, this is possible.

Table 2 Normalized symmetric difference for the various algorithms on five resolutions.

	$r = 100$	$r = 225$	$r = 400$	$r = 625$	$r = 900$
Optimal	0.263	0.188	0.147	0.119	0.101
Hausdorff	0.282	0.201	0.155	0.123	0.103
Fréchet	0.306	0.227	0.184	0.148	0.122

minimum and the maximum symmetric difference for each algorithm / polygon combination is rather large. We hence concluded that placement can have a significant effect on the achieved symmetric difference. Hence, if the application permits us to choose the placement, it is advisable to do so to obtain the best possible result. This leads to an interesting open question of whether we can algorithmically optimize the placement, to avoid the need to find a good placement with trial and error. In the upcoming analysis, we also consider the effect of resolution and placement, with respect to the Fréchet distance.

4.2 Fréchet analysis

Theorem 12 predicts an upper bound on the Fréchet distance based on $\sqrt{2}$-narrowness. However, if the points defining the narrowness lie within different squares of grid vertices, this bound may be naive. Moreover, it assumes a worst-case detour, going away in a thin triangle to maximize the distance between the detour and a doubly-visited cell. Hence, the algorithm has the potential to perform better, depending on the actual geometry and its placement with respect to the grid. Here, we discuss our investigation of these effects.

Procedure. We use all 34 polygons for our experiments. As we may expect the grid resolution to significantly affect results, we used 20 different resolutions. In particular, we use resolutions varying from 10 000 to 25, using $(100/s)^2$ with scale $s \in \{1, \ldots, 20\}$.

For each resolution-polygon combination (case), we measure its $\sqrt{2}$-narrowness (see [7] for details on how to compute narrowness) and derive the predicted upper bound. Then, we run the Fréchet algorithm, using the 25 possible offsets in $\{0, 0.2, 0.4, 0.6, 0.8\}^2$, and measure the precise Fréchet distance between input and output. We keep track of three summary statistics for each case: the minimum (best), average ("expected") and maximum (worst) measured Fréchet distance.

Effect of placement. We consider placement with respect to the grid (offset) to have a significant effect on the result computed for a polygon, if the difference between the maximal and minimal Fréchet distance over the 25 offsets is at least 2. Almost 30 % of cases exhibit such a significant effect, with the animal contours being particularly affected (35 % significant). Again, this raises the question of whether we can algorithmically determine a good placement.

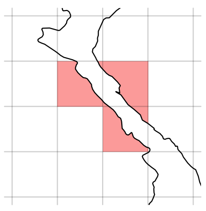


Figure 14 Red cells cause a cut-off and have high symmetric difference.

Upper bound quality. We define the performance as the measured Fréchet distance as a percentage of the upper bound. We consider a performance of 40 % significantly better than the upper bound. Using the best placement, over 95 % of cases perform significantly better. Averaging performance over placement, we still find such a majority (over 81 %). Interestingly, this drop is mostly due to the animal contours, of which only 63 % now perform significantly better. Thus, although we have a provable upper bound, we may typically expect our simple algorithm to perform significantly better than the upper bound. This holds even without any postprocessing to further optimize the result and when taking a random offset.

Effect of resolution. The influence of the resolution on the above results does not seem to exhibit a clear pattern. Nonetheless, resolution likely plays an important role in these results, but not as straightforward as either low or high resolution being more problematic. Instead, it is likely the most problematic resolutions are those at which the $\sqrt{2}$-narrowness of the polygon jumps as a new pair of edges comes within distance $\sqrt{2}$ of each other. However, an in-depth investigation of this is beyond the scope of this paper.

Heuristic improvement. In contrast to the Hausdorff algorithm, the Fréchet algorithm needs no heuristic improvement on inputs that are not too narrow. However, badly placed narrow polygons can be problematic: large parts of the polygon may be cut, greatly diminishing similarity. A solution may be to select an appropriate resolution (if our application permits us to). In our experiments the algorithm tends to perform well at resolutions where the symmetric-difference optimal solution is a single grid polygon. The advantage of our Fréchet algorithm is that it guarantees a grid polygon on all outputs and bounds the Fréchet distance.

Nonetheless, we may want to consider heuristic postprocessing to obtain a locally-optimal result. If we want to do this in terms of the symmetric difference, we may use similar techniques as for the Hausdorff algorithm. However, this does not perform well: the narrow strip that causes the Fréchet algorithm to perform badly tends to effect a high symmetric difference for the nearby grid cells (Fig. 14). As such, the result is already (close to) a local optimum in terms of the symmetric difference.

5 Conclusion

We presented two algorithms to map simple polygons to grid polygons that capture the shape of the polygon well. For measuring the distance between the input and the output, we considered the Hausdorff and the Fréchet distance. We achieved a constant bound on the Hausdorff distance; for the Fréchet distance we require a realistic input assumption to achieve a constant bound. We also evaluated our algorithms in practice. Although the Hausdorff algorithm does not produce great results directly, the algorithm achieves good results when combined with heuristic improvements. The Fréchet algorithm, on the other hand, struggles with narrow polygons, and it is not clear how to improve the results using heuristics. Designing an algorithm for the Fréchet distance that also works well in practice remains an interesting open problem. Another interesting open problem is to algorithmically optimize the placement of the input polygon, for the best results of both the Hausdorff and the Fréchet algorithm.

References

1 Helmut Alt, Bernd Behrends, and Johannes Blömer. Approximate matching of polygonal shapes. *Annals of Mathematics & Artificial Intelligence*, 13(3-4):251–265, 1995.

2 Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5:75–91, 1995.

3 Helmut Alt, Christian Knauer, and Carola Wenk. Comparison of distance measures for planar curves. *Algorithmica*, 38(1):45–58, 2003.

4 Ernst Althaus, Friedrich Eisenbrand, Stefan Funke, and Kurt Mehlhorn. Point containment in the integer hull of a polyhedron. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 929–933, 2004.

5 K. Joost Batenburg, Sjoerd Henstra, Walter A. Kosters, and Willem Jan Palenstijn. Constructing simple nonograms of varying difficulty. *Pure Mathematics and Applications (Pu. MA)*, 20:1–15, 2009.

6 Daniel Berend, Dolev Pomeranz, Ronen Rabani, and Ben Raziel. Nonograms: Combinatorial questions and algorithms. *Discrete Applied Mathematics*, 169:30–42, 2014.

7 Quirijn W. Bouts, Irina Kostitsyna, Marc van Kreveld, Wouter Meulemans, Willem Sonke, and Kevin Verbeek. Mapping polygons to the grid with small hausdorff and fréchet distance. *Computing Research Repository (arXiv)*, abs/1606.06660, 2016.

8 Rafael G. Cano, Kevin Buchin, Thom Castermans, Astrid Pieterse, Willem Sonke, and Bettina Speckmann. Mosaic drawings and cartograms. *Computer Graphics Forum*, 34(3):361–370, 2015.

9 Bernard Chazelle. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry*, 6:485–524, 1991.

10 Jinhee Chun, Matias Korman, Martin Nöllenburg, and Takeshi Tokuyama. Consistent digital rays. *Discrete & Computational Geometry*, 42(3):359–378, 2009.

11 Mark de Berg, Dan Halperin, and Mark H. Overmars. An intersection-sensitive algorithm for snap rounding. *Computational Geometry*, 36(3):159–165, 2007.

12 Olivier Devillers and Philippe Guigue. Inner and outer rounding of boolean operations on lattice polygonal regions. *Computational Geometry*, 33(1-2):3–17, 2006.

13 Michael T. Goodrich, Leonidas J. Guibas, John Hershberger, and Paul J. Tanenbaum. Snap rounding line segments efficiently in two and three dimensions. In *Proceedings of the 13th Annual Symposium on Computational Geometry (SoCG)*, pages 284–293, 1997.

14 Daniel H. Greene and F. Frances Yao. Finite-resolution computational geometry. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 143–152, 1986.

15 Warwick Harvey. Computing two-dimensional integer hulls. *SIAM Journal on Computing*, 28(6):2285–2299, 1999.

16 John Hershberger. Stable snap rounding. *Computational Geometry*, 46(4):403–416, 2013.

17 Reinhard Klette and Azriel Rosenfeld. *Digital Geometry – geometric methods for digital picture analysis*. Morgan Kaufmann, 2004.

18 Reinhard Klette and Azriel Rosenfeld. Digital straightness – a review. *Discrete Applied Mathematics*, 139(1-3):197–230, 2004.

19 Johannes Kopf, Ariel Shamir, and Pieter Peers. Content-adaptive image downscaling. *ACM Transactions on Graphics*, 32(6):Article No. 173, 2013.

20 Wouter Meulemans. *Similarity Measures and Algorithms for Cartographic Schematization*. PhD thesis, Technische Universiteit Eindhoven, 2014.

21 Wouter Meulemans. Discretized approaches to schematization. *Computing Research Repository (arXiv)*, abs/1606.06488, 2016.

22 Emilio G. Ortíz-García, Sancho Salcedo-Sanz, José M. Leiva-Murillo, Ángel M. Pérez-Bellido, and José Antonio Portilla-Figueras. Automated generation and visualization of picture-logic puzzles. *Computers & Graphics*, 31(5):750–760, 2007.

Hitting Set for Hypergraphs of Low VC-Dimension

Karl Bringmann[1], László Kozma[2], Shay Moran[3], and N. S. Narayanaswamy[4]

1 **Institut für Theoretische Informatik, ETH Zürich, Zürich, Switzerland**
`karl.bringmann@inf.ethz.ch`
2 **Department of Computer Science, Saarland University, Saarbrücken, Germany**
`kozma@cs.uni-saarland.de`
3 **Department of Computer Science, Technion-IIT, Israel, Microsoft Research, Hertzelia; and**
Max Planck Institute for Informatics, Saarbrücken, Germany
`shaymoran1@gmail.com`
4 **Department of Computer Science and Engineering, Indian Institute of Technology Madras, Madras, India**
`swamy@cse.iitm.ac.in`

Abstract

We study the complexity of the Hitting Set problem in set systems (hypergraphs) that avoid certain sub-structures. In particular, we characterize the classical and parameterized complexity of the problem when the Vapnik-Chervonenkis dimension (VC-dimension) of the input is small.

VC-dimension is a natural measure of complexity of set systems. Several tractable instances of Hitting Set with a geometric or graph-theoretical flavor are known to have low VC-dimension. In set systems of bounded VC-dimension, Hitting Set is known to admit efficient and almost optimal approximation algorithms (Brönnimann and Goodrich, 1995; Even, Rawitz, and Shahar, 2005; Agarwal and Pan, 2014).

In contrast to these approximation-results, a low VC-dimension does not necessarily imply tractability in the parameterized sense. In fact, we show that Hitting Set is $W[1]$-hard already on inputs with VC-dimension 2, even if the VC-dimension of the dual set system is also 2. Thus, Hitting Set is very unlikely to be fixed-parameter tractable even in this arguably simple case. This answers an open question raised by King in 2010. For set systems whose (primal or dual) VC-dimension is 1, we show that Hitting Set is solvable in polynomial time.

To bridge the gap in complexity between the classes of inputs with VC-dimension 1 and 2, we use a measure that is more fine-grained than VC-dimension. In terms of this measure, we identify a sharp threshold where the complexity of Hitting Set transitions from polynomial-time-solvable to NP-hard. The tractable class that lies just under the threshold is a generalization of Edge Cover, and thus extends the domain of polynomial-time tractability of Hitting Set.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes, F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases hitting set, VC-dimension

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.23

1 Introduction

Let $\mathcal{C}$ be a collection of subsets of a finite set X. We call the pair $(X, \mathcal{C})$ a *set system*.[1] A *hitting set* of $(X, \mathcal{C})$ is a subset of X that has non-empty intersection with all members of $\mathcal{C}$.

[1] Alternative names in the literature are *hypergraph* and *range space*.

The decision version of the Hitting Set problem asks, given a positive integer k, whether a set system has a hitting set of size at most k.

Hitting Set and its dual, Set Cover, are both ubiquitous and notoriously difficult problems. For an arbitrary set system $(X, \mathcal{C})$, Hitting Set is NP-hard to approximate [28, 3] with a multiplicative factor better than $c \cdot \log(|\mathcal{C}| \cdot |X|)$, for some constant $c > 0$.

Given a set system $\mathcal{F} = (X, \mathcal{C})$, and a set $A \subseteq X$, we define the *projection*[2] *of $\mathcal{F}$ on A* as $PR_{\mathcal{F}}(A) = \{R \cap A \mid R \in \mathcal{C}\}$. A set A is said to be *shattered* by $\mathcal{F}$ if $PR_{\mathcal{F}}(A) = 2^A$, i.e. the set of all subsets of A. The *Vapnik-Chervonenkis dimension* (or *VC-dimension*) of a set system $\mathcal{F}$, denoted $VC(\mathcal{F})$, is the cardinality of the largest set shattered by $\mathcal{F}$. VC-dimension was originally introduced in learning theory [39, 4], where it captures the sample complexity in the PAC model. Since its introduction, VC-dimension has seen many further applications both inside and outside learning theory (see e.g. [8, 31]) and it has become a standard measure of complexity of set systems.

Allowing a set system to have large VC-dimension means that less restrictions are placed on its structure, making it more difficult as a Hitting Set instance. In this paper we study both the classical and parameterized complexity of Hitting Set when the VC-dimension of the input set system is bounded.

Hitting Set and parameterized complexity. In parameterized complexity, a problem is called *fixed-parameter tractable* (FPT) with respect to a parameter[3] k, if there exists an algorithm that solves it in time $O(f(k) \cdot n^{O(1)})$ for an arbitrary function f (where n is the input size). Fixed-parameter tractability has emerged as a powerful tool to deal with hard combinatorial problems. We refer the reader to [12, 34, 15] for more details. Unfortunately, Hitting Set is $W[2]$-hard [15], and thus unlikely to be FPT, meaning that it is hopelessly difficult even from a parameterized perspective.

However, instances arising in various applications (e.g. in graph-theoretical or in geometric settings) often have special structure that can be algorithmically exploited. Indeed, the literature abounds with studies of problems - many of them FPT - that can be seen as special cases of Hitting Set.

Graph-theoretical examples of Hitting Set problems include Vertex Cover, Edge Cover, Feedback Vertex Set, and Dominating Set. In each of these problems the input set system is implicitly defined by an underlying graph G, with sets corresponding to the edges, vertices, cycles, and neighborhoods of G, respectively. The first three of these problems are well-known to be FPT (Edge Cover is even in P). Dominating Set remains $W[2]$-hard, but is FPT in certain families of graphs (see Table 1). Intuitively, Dominating Set is hard because it places very few restrictions on the input: Every set system whose incidence matrix is symmetric can be a Dominating Set instance. Special cases where Dominating Set is FPT include *biclique-free* graphs [35, 38] (a family that contains *bounded genus*, *planar*, *bounded treewidth*, and many other natural classes), *claw-free* graphs [25], and graphs *with girth at least five* [36]. The structure that makes these special cases of Dominating Set tractable can be described in terms of *forbidden patterns* in the adjacency matrix of G. For instance, biclique-freeness simply translates to the avoidance of an all-1s submatrix of a certain size. Our work continues this line of investigation: A VC-dimension smaller than d can be interpreted as the avoidance of every matrix with d columns that contain all 2^d different boolean vectors in its rows.

[2] Also known in the literature as the *trace* of a set system.
[3] In this paper we always use the standard parameter, i.e. the solution size k.

In geometric examples of Hitting Set, the input set system is defined by the incidences between (typically) low complexity geometric shapes, such as points, intervals, lines, disks, rectangles, hyperplanes, etc. VC-dimension is a natural and useful complexity measure for geometrically defined set systems [39, 4, 6, 23]. In Table 1 we list some representative examples of Hitting Set problems from the literature.

Hitting Set and VC-dimension. Given the difficulty of Hitting Set and the wealth of special cases that are FPT or polynomial-time solvable, it is natural to ask for a general structural property of set systems that guarantees tractability. Such a question has been successfully answered in the field of approximation algorithms: After a series of approximation-results for concrete geometric problems, the landmark result of Brönnimann and Goodrich [7] gave an almost optimal[4] approximation algorithm for Hitting Set on set systems with bounded VC-dimension. The algorithm has been further improved by Even at al. [14] and recently by Agarwal and Pan [1]. In this paper we consider this question from a parameterized viewpoint.

In general, the relevance of VC-dimension to Hitting Set has long been known: Low VC-dimension implies the existence of an ϵ-net of small size [23]. An ϵ-net can be seen as a relaxed form of hitting set in which we are only interested in hitting all sets whose size is at least an ϵ-fraction of the universe size. For set systems with low VC-dimension the size of the fractional hitting set is close to the size of the integral hitting set - this observation is the basis of the approximation-result of Brönnimann and Goodrich [7].

Dual VC-dimension. The *incidence matrix* of a set system $\mathcal{F} = (X, \mathcal{C})$ is a 0/1 matrix with columns indexed by elements of X, and rows indexed by members of $\mathcal{C}$. An entry (A, x) of the incidence matrix (where $A \in \mathcal{C}$ and $x \in X$) is 1 if $x \in A$, and 0 otherwise.

Given a set system $\mathcal{F}$, it is natural to consider its *dual* set system denoted $\mathcal{F}^T$, obtained by interchanging the roles of elements and sets (i.e. transposing the incidence matrix of the set system[5]). The Hitting Set problem on the dual set system is known as *Set Cover*. The VC-dimension of the dual set system, denoted $VC(\mathcal{F}^T)$ is a further natural parameter of set systems. It is well-known that if $VC(\mathcal{F}) = d$, then the inequality $VC(\mathcal{F}^T) < 2^{d+1}$ holds.

Our results. We study the classical and parameterized complexity of Hitting Set restricted to set systems with small VC-dimension. In light of Table 1, there is no clear separation at any value of the VC-dimension: Some FPT classes have unbounded VC dimension, while $W[1]$-hard classes with VC-dimension 3 are known[6]. However, an FPT result for Hitting Set restricted to VC-dimension 2 would generalize many known FPT results for special cases of Hitting Set. Hence, we study the existence of a small threshold value of VC-dimension, below which Hitting Set is tractable and at which it becomes intractable (both in the parameterized and in the classical sense). The program of finding such a dichotomy for the FPT complexity of Hitting Set in terms of the VC-dimension has also been proposed by King [26].

[4] As a further witness to the difficulty of Hitting Set, *almost optimal* here means a *logarithmic factor of the optimum*, i.e. $O(\log k)$. For more restricted geometric problems better approximation ratios are known, see e.g. [9, 33].

[5] The transposed incidence matrix may contain duplicate rows, contradicting the definition of a set system. It is safe to discard such duplicates, as this does not affect the VC-dimension or the Hitting Set solution.

[6] To the best of our knowledge, prior to our paper there were no $W[1]$-hard examples known with VC-dimension *or* dual VC-dimension lower than 3. In fact, we are not aware of $W[1]$-hard examples with *explicitly stated* VC-dimension lower than 4, see § B.

Table 1 Special cases of Hitting Set in the FPT literature, and their VC-dimension. For hardness results, the values for VC-dimension should be prefixed with "at least", for algorithmic results (P and FPT) with "at most". The results in the table are discussed in the Appendix § A and § B.

Graph problem	FPT status	VC-dimension
Edge Cover	P	2
Tree-Like Hitting Set [20]	P	∞
Vertex Cover	FPT	2
Dominating Set (claw-free) [25]	FPT	∞
Dominating Set (girth ≥ 5) [36]	FPT	2
Dominating Set (planar) [16]	FPT	4
Dominating Set ($K_{t,t}$-free) [35, 38]	FPT	$t + \lceil \log_2 t \rceil$ -1
Feedback Vertex Set [21, 10]	FPT	∞
Dominating Set (unit disk) [29]	$W[1]$-hard	3
Dominating Set (induced $K_{4,1}$-free) [25]	$W[1]$-hard	∞
Dominating Set (Δ-free) [36]	$W[2]$-hard	∞
Geometric problem	FPT status	VC-dimension
Line intervals	P	2
Halfplane arrangement in $\mathbb{R}^2$ [22]	P	3
Disjoint Rectangle Stabbing [24]	FPT	2
Pseudoline arrangement	FPT	2
Hyperplane arrangement in $\mathbb{R}^d$	FPT	$d+1$
Halfspace arrangement in $\mathbb{R}^3$ [§ C]	$W[1]$-hard	4
Collection of unit disks in $\mathbb{R}^2$ [18]	$W[1]$-hard	3
Collection of unit squares in $\mathbb{R}^2$ [18]	$W[1]$-hard	3
Rectangle Stabbing [11]	$W[1]$-hard	3

In this paper, we show the threshold of tractability to be at the (surprisingly low) value of 2, i.e., we prove $W[1]$-hardness of Hitting Set restricted to VC-dimension 2 (even if also the dual VC-dimension is 2). The phenomenon of a large gap between the complexity of set systems of VC-dimension 1 and set systems of VC-dimension 2 also occurs in other areas such as communication complexity, machine learning, and geometry [2, 32]. Moreover, assuming the Exponential Time Hypothesis (ETH) we obtain an almost matching lower bound for the trivial $n^{O(k)}$ algorithm. We prove this result in § 2.

▶ **Theorem 1.** *Hitting Set and Set Cover restricted to set systems $\mathcal{F} = (X, \mathcal{C})$ with $VC(\mathcal{F}) = VC(\mathcal{F}^T) = 2$ are $W[1]$-hard. Moreover, if any of these problems can be solved in time $f(k) \cdot |X|^{o(k/\log k)}$, where f is an arbitrary function and k is the solution size, then ETH fails.*

Note. Theorem 1 could be stated with $|X|$ replaced by $|\mathcal{C}|$ or $|\mathcal{C}| \cdot |X|$, which are perhaps more natural as a measure of input length. However, the Sauer-Perles-Shelah lemma (see e.g. [37]) states that if $VC(\mathcal{F}) = d$, then $|\mathcal{C}| \leq \sum_{j=0}^{d} \binom{|X|}{j}$. Therefore, $|X|$ and $|\mathcal{C}|$ are within a polynomial factor of each other, which allows us to use $|X|$.

The hardness result of Theorem 1 can be strengthened to set systems with symmetric incidence matrices, i.e. the result also holds for Dominating Set. The construction is more involved in that case, and we omit it in this version of the paper.

On the positive side, given a set system $\mathcal{F}$, if $VC(\mathcal{F}) = 1$ *or* $VC(\mathcal{F}^T) = 1$, we show that Hitting Set is in P. The proof is simple and self-contained (see § 3). The $VC(\mathcal{F}) = 1$ case was known prior to this work [26], but we are not aware of a published proof.

To bridge the rather large gap in complexity between set systems of VC-dimension 1 and 2, we use a finer parameterization which was also used in [2, 32]. For a pair of integers $\alpha, \beta \geq 1$, a set system $\mathcal{F} = (X, \mathcal{C})$ is an (α, β)-system if for any set $A \subseteq X$ with $|A| \leq \alpha$ the projection

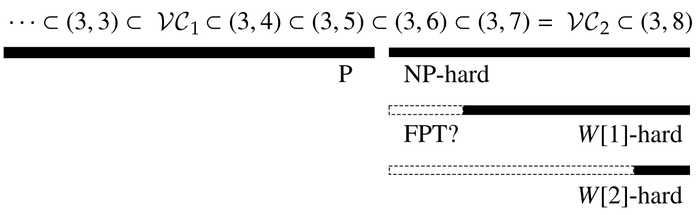

$\cdots \subset (3,3) \subset \mathcal{VC}_1 \subset (3,4) \subset (3,5) \subset (3,6) \subset (3,7) = \mathcal{VC}_2 \subset (3,8)$
P
NP-hard
FPT?
$W[1]$-hard
$W[2]$-hard

Figure 1 The complexity of Hitting Set when the VC-dimension is low.

$PR_{\mathcal{F}}(A)$ has cardinality at most β. In other words, a set system is an (α, β)-system, if every submatrix of its incidence matrix with α columns has at most β different vectors in its rows. Let $\mathcal{VC}_d$ denote the class of set systems with VC-dimension at most d. Observe that $\mathcal{VC}_{d-1}$ is equal to the class of $(d, 2^d - 1)$-systems. Moreover, every set system is a $(d, 2^d)$-system, for arbitrary $d \geq 1$. Hence, Hitting Set on $(3, 8)$-systems is the standard Hitting Set problem (without restrictions) and thus $W[2]$-hard.

The Sauer-Perles-Shelah Lemma can also be stated using this notation: Every $(d, 2^d - 1)$-system is a $(m, \sum_{j=0}^{d-1} \binom{m}{j})$-system for every $m \geq d$. In particular, every set system in $\mathcal{VC}_1$ is a $(3, 4)$-system. Further, we prove that every Edge Cover instance is a $(3, 5)$-system, but the reverse does not hold. Edge Cover is well-known to be solvable in polynomial time using matching techniques [17]. The next result (see § 3) extends the domain of polynomial-time solvability from Edge Cover to the larger class of $(3, 5)$-systems.

▶ **Theorem 2.** *Hitting Set on $(3, 5)$-systems is in P.*

The algorithm we present for proving Theorem 2 is fairly simple. However, its analysis is quite involved – revealing some of the combinatorial structure underlying $(3, 5)$-systems.

In contrast to $(3, 5)$-systems, it is not hard to see that there are $(3, 6)$-systems for which the Hitting Set problem is NP-hard.

▶ **Theorem 3.** *Hitting Set on $(3, 6)$-systems is NP-hard.*

This discussion yields a complete characterization of the complexity of Hitting Set on $(3, \beta)$-systems with a transition from polynomially-solvable to NP-hard between the β values of 5 and 6. Regarding the FPT status of the problem, the picture is almost complete, with the question of $(3, 6)$-systems remaining open. The results are illustrated in Figure 1.

Open questions. An immediate open question raised by our work is whether Hitting Set is FPT on $(3, 6)$-systems. The (α, β)-parameterization provides an ever finer hierarchy of set systems, as α increases. A more ambitious goal would be a full characterization of the complexity of Hitting Set in (α, β)-systems for $\alpha \geq 3$. We only have preliminary results in this direction. Finally, we leave open the question whether Hitting Set is even $W[2]$-hard on set systems of bounded VC-dimension.

Related work. Langerman and Morin [27] study the parameterized complexity of an abstract covering problem with a dimension parameter that has some connections to the VC-dimension. However, the results are not directly comparable with ours: The instances studied by Langerman and Morin can have arbitrarily large VC-dimension and are restricted by other conditions, whereas the instances we study have very low VC-dimension, but have no further constraints.

Notation. Consider a set system $\mathcal{F} = (X, \mathcal{C})$. Let $b_1, \dots, b_t \in X$ be distinct elements and $(p_1, \dots, p_t) \in \{0,1\}^t$. We say that $(b_1, \dots, b_t)$ *realizes the pattern* $p_1 \dots p_t$ if the set $\{b_i \mid p_i = 1\}$ is contained in $PR_{\mathcal{F}}(\{b_1, \dots, b_t\})$.

2 Hitting Set with VC-dimension 2 is W[1]-hard

In this section we prove the $W[1]$-hardness of Hitting Set and Set Cover on set systems of VC-dimension 2 and dual VC-dimension 2. The NP-hardness of this class was known, implied for example, by the NP-hardness of Vertex Cover.

▶ **Theorem 1** (restated). *Hitting Set and Set Cover restricted to set systems $\mathcal{F} = (X, \mathcal{C})$ with $VC(\mathcal{F}) = VC(\mathcal{F}^T) = 2$ are $W[1]$-hard. Moreover, if any of these problems can be solved in time $f(k) \cdot |X|^{o(k/\log k)}$, where f is an arbitrary function and k is the solution size, then ETH fails.*

In the remainder of this section we prove Theorem 1. Since Hitting Set on a set system $\mathcal{F}$ is equivalent to Set Cover on set system $\mathcal{F}^T$, it suffices to prove hardness of Hitting Set.

We reduce to Hitting Set from the Partitioned Subgraph Isomorphism problem: Given a host graph $G = (V, E)$ with a partitioning of the vertices $V = V_1 \cup \ldots \cup V_t$ and a pattern graph $H = ([t], F)$ with $|F| = k$, decide whether there are vertices $u_1 \in V_1, \dots, u_t \in V_t$ such that $u_i u_j \in E$ for every $ij \in F$. It is known that Partitioned Subgraph Isomorphism is W[1]-hard and cannot be solved in time $f(k) \cdot n^{o(k/\log k)}$, where $n = |V|$, and f is an arbitrary function, unless ETH fails [30].

We first show that we may assume the hard instance to have $t = k$, i.e. that the number of vertices and the number of edges in H are equal. Consider an arbitrary instance of Partitioned Subgraph Isomorphism. Since Partitioned Subgraph Isomorphism splits naturally over connected components of H, we may assume that H is connected. If $k > t$, add $k - t$ isolated vertices to H, and add the corresponding partitions containing isolated vertices $V_{t+1} = \{v_{t+1}\}, \dots, V_k = \{v_k\}$ to G, without changing the existence of a solution. Observe that the parameter k is unchanged. In the case when $k < t$, since H is connected, it follows that $k = t - 1$. We add two components to H: a clique on 4 vertices and an isolated vertex. To G we add the partitions $V_{t+1} = \{v_{t+1}\}, \dots, V_{t+5} = \{v_{t+5}\}$ such that $v_{t+1}, \dots, v_{t+4}$ form a clique, and v_{t+5} is an isolated vertex. After the transformation, H contains $k + 6 = t + 5$ edges and vertices. Furthermore, the equivalence of the solutions is preserved, and the parameter k (the number of edges in H), increases by a constant only.

For ease of notation we let $E \subseteq [n] \times [n]$ and write uv for an edge in E. Since G is undirected, the set E contains uv if and only if it contains vu. Similarly, $F \subseteq [k] \times [k]$ and $ij \in F$ if and only if $ji \in F$. We fix any ordering $<$ on V and the lexicographic[7] ordering $<$ on $V \times V$ and thus on E. We write $E_{ij} := E \cap (V_i \times V_j)$.

We construct an equivalent Hitting Set instance $\mathcal{F}$. We start by defining $\mathcal{F}$ and proving correctness, and later prove $VC(\mathcal{F}) = 2$ and $VC(\mathcal{F}^T) = 2$.

2.1 Construction of $\mathcal{F}$

We construct a set system $\mathcal{F} = (X, \mathcal{C})$ as follows. The elements of X are

$x^{\ell}_{i,u}$	for $i \in [k]$, $u \in V_i$, $\ell \in [2\deg_H(i)]$,
$y^{\ell}_{ij,uv}$	for $ij \in F$, $uv \in E_{ij}$, $\ell \in [5]$.

[7] $uv < wz \iff (u < w \lor (u = w \land v < z))$.

It will be convenient to structure these elements into disjoint *ground sets* $X_i^\ell = \{x_{i,u}^\ell \mid u \in V_i\}$ and $Y_{ij}^\ell = \{y_{ij,uv}^\ell \mid uv \in E_{ij}\}$. We will force each hitting set to pick exactly one element from every ground set; these elements will encode the desired copy of H (should it exist).

In the remainder we define the sets in $\mathcal{C}$. First we introduce the following sets of $\mathcal{C}$.

$$A_{i,u}^\ell = \{x_{i,v}^\ell \mid v < u\} \cup \{x_{i,v}^{\ell+1} \mid v \ge u\}, \qquad \text{for } i \in [k], u \in V_i,\ \ell \in [2\deg_H(i)],$$
$$B_{ij,uv}^\ell = \{y_{ij,wz}^\ell \mid wz < uv\} \cup \{y_{ij,wz}^{\ell+1} \mid wz \ge uv\}, \qquad \text{for } ij \in F,\ uv \in E_{ij},\ \ell \in [5].$$

Here, $x_{i,v}^{\ell+1}$ is to be interpreted as $x_{i,v}^1$ for $\ell = 2\deg_H(i)$, and $y_{ij,wz}^{\ell+1}$ as $y_{ij,wz}^1$ for $\ell = 5$, i.e. there is a wrap-around of the index ℓ. Note that the disjoint ground sets appear as sets $A_{i,u}^\ell$ (where u is the smallest vertex in V_i) and $B_{ij,uv}^\ell$ (where uv is the lexicographic smallest edge in E_{ij}). Hence, any hitting set of $\mathcal{F}$ contains at least one element of every ground set.

Note that the total number of ground sets is

$$k' = 5\,|F| + \sum_{i\in[k]} 2\deg_H(i) = 9k.$$

We set the number of vertices to be chosen in the hitting set to k', i.e. from now on we only consider hitting sets of size k' of $\mathcal{F}$. Since there are exactly k' ground sets, and they are mutually disjoint, it follows that any hitting set of $\mathcal{F}$ of size k' contains exactly one element $x_{i,u(i,\ell)}^\ell$ of any ground set X_i^ℓ, and exactly one element $y_{ij,e(ij,\ell)}^\ell$ of any ground set Y_{ij}^ℓ. Moreover, observe that hitting the set $A_{i,u}^\ell$ implies $u(i,\ell) < u \vee u(i,\ell+1) \ge u$.

This holds for all $u \in V$, and so $u(i,\ell) \le u(i,\ell+1)$ for all ℓ. Since there is a cyclic wrap-around of ℓ it follows that $u(i,\ell) = u(i,\ell+1)$ for all ℓ, and so let $u_i \in V_i$ such that $u(i,\ell) = u_i$ for all ℓ. Similarly, the sets $B_{ij,uv}^\ell$ ensure that $e(ij,\ell) = e_{ij}$ for all ℓ and some $e_{ij} = v_{ij}w_{ij} \in E_{ij}$.

Observe that the picked edges $e_{ij} = v_{ij}w_{ij}$ form a subgraph of G. This subgraph is isomorphic to H if we additionally ensure $u_i = v_{ij}$ and $u_j = w_{ij}$ for all $ij \in F$. To this end, we introduce the sets $C_{ij,u}$ and $C'_{ij,u}$ for $ij \in F$, $u \in V_i$. If ij is the d-th edge incident to vertex i in H, then we set

$$C_{ij,u} = \{x_{i,v}^{2d-1} \mid v < u\} \cup \{y_{ij,wz}^1 \mid w \ge u,\ z \in V_j\},$$
$$C'_{ij,u} = \{x_{i,v}^{2d} \mid v > u\} \cup \{y_{ij,wz}^2 \mid w \le u,\ z \in V_j\}.$$

Observe that this ensures $u_i = v_{ij}$ for all $ij \in F$. Indeed, fixing u_i the sets C_{ij,u_i} and C'_{ij,u_i} are only hit if we choose $y_{ij,v_{ij}w_{ij}}^1$ with $v_{ij} \ge u_i$ and $y_{ij,v_{ij}w_{ij}}^2$ with $v_{ij} \le u_i$.

We implement the remaining condition $u_j = w_{ij}$ indirectly by introducing the sets

$$D_{ij,uv} = \{y_{ij,wz}^3 \mid wz < uv\} \cup \{y_{ij,wz}^5 \mid wz > uv\} \cup \{y_{ji,vu}^4\}, \quad \text{for } ij \in F,\ i<j,\ uv \in E_{ij}.$$

This encodes the formula $e_{ij} \ne uv \vee e_{ji} = vu$ for all $uv \in E_{ij}$, and thus ensures that $v_{ij} = w_{ji}$ and $w_{ij} = v_{ji}$ for all $ij \in F$ with $i < j$ (and thus also for all $ij \in F$ without the condition $i < j$). This indirectly encodes the restriction $u_j = w_{ij}$, since $u_j = v_{ji}$ (by the sets of type $C_{ji,*}$ and $C'_{ji,*}$) and $v_{ji} = w_{ij}$ (by the sets of type $D_{ji,*}$).

In total, any hitting set of $\mathcal{F}$ of size k' yields a subgraph of G that is equal to H. It is easy to show that the inverse holds as well: If $u_1 \in V_1, \dots, u_k \in V_k$ induce a copy of H in G, then picking the elements x_{i,u_i}^ℓ and $y_{ij,u_iu_j}^\ell$ for all ij, ℓ yields a hitting set of $\mathcal{F}$ of size k'. This shows the correctness of our construction.

We show that $VC(\mathcal{F}) = VC(\mathcal{F}^T) = 2$ in the next two sections. Since $k' = O(k)$, $|\mathcal{F}| = n^{O(1)}$, and the construction of $\mathcal{F}$ can be done in polynomial time, W[1]-hardness of Hitting Set restricted to $VC(\mathcal{F}) = VC(\mathcal{F}^T) = 2$ follows, and any $f(k')|\mathcal{F}|^{o(k'/\log k')}$ algorithm for this problem would yield an $f(k)n^{o(k/\log k)}$ algorithm for Partitioned Subgraph Isomorphism, contradicting ETH.

2.2 VC-dimension 2

It is easy to see that in general $VC(\mathcal{F})$ can be at least 2, e.g., the elements $x_{i,1}^2, x_{i,2}^2$ are shattered by the sets $A_{i,1}^1$ (pattern 11), $A_{i,2}^1$ (pattern 01), $A_{i,2}^2$ (pattern 10), and any set of type B (pattern 00).

To prove that $\mathcal{F}$ has VC-dimension at most 2, we first argue that we can remove the single element $y_{ji,vu}^4$ from $D_{ij,uv}$, i.e. we replace any set $D_{ij,uv}$ by

$$D_{ij,uv}^* := D_{ij,uv} \setminus \{y_{ji,vu}^4\} = \{y_{ij,wz}^3 \mid wz < uv\} \cup \{y_{ij,wz}^5 \mid wz > uv\},$$

to obtain a set system $\mathcal{F}^*$. We claim that if there are elements a, b, c realizing the patterns $110, 101, 011, 111$ in F then these elements also realize these patterns in $\mathcal{F}^*$. Indeed, assume for the sake of contradiction that there are elements a, b, c realizing all of the patterns 110, 101, 011, and 111 in $\mathcal{F}$ but not in $\mathcal{F}^*$. Then without loss of generality, for some $ij \in F, uv \in E_{ij}$, $a = y_{ji,vu}^4$ and $b \in D_{ij,uv} \setminus \{a\} = D_{ij,uv}^*$. Now, there is only one set in $\mathcal{F}$ containing both a and b, namely $D_{ij,uv}$ (since $D_{ij,uv}$ is the only set which intersects both Y_{ji}^4 and $Y_{ij}^3 \cup Y_{ij}^5$). Thus, one of the patterns 110 and 111 is missing, contradicting the assumption that a, b, c realize all patterns 110, 101, 011, and 111. Hence, if we show that $\mathcal{F}^*$ contains no three elements realizing all patterns 110, 101, 011, and 111, then no three elements of $\mathcal{F}$ are shattered.

To this end, we first lift the ordering of V and the lexicographic ordering of E to orderings on the ground sets, i.e. for $u < v$ we set $x_{i,u}^\ell < x_{i,v}^\ell$ and for $uv < wz$ we set $y_{ij,uv}^\ell < y_{ij,wz}^\ell$. We use the following crucial observation about this ordering and $\mathcal{F}^*$.

▶ **Observation 4.** *Any set system in $\mathcal{F}^*$ intersects at most two ground sets. Any set system in $\mathcal{F}^*$ restricted to any ground set S forms an* interval *(with respect to the ordering on S). Moreover, for any pair of ground sets $S_1 \neq S_2$, the sets of $\mathcal{F}^*$ intersecting both S_1 and S_2 either all intersect in the smallest element of S_1 or all intersect in the largest element of S_1.*

With this observation at hand, consider any elements $a, b, c \in X$. We do a case distinction over the number of different ground sets that a, b, c are contained in.

(1) If a, b, c come from the same ground set S, then they are ordered in S, say $a < b < c$. Since each set of $\mathcal{F}^*$ forms an interval in S, there is no set of $\mathcal{F}^*$ containing a and c but not b.

(2) If a and b come from the same ground set S_1, say with $a < b$, and c comes from a different ground set S_2, then we consider the last part of Observation 4. If all sets of $\mathcal{F}^*$ containing elements of S_1 and S_2 contain the smallest element of S_1, then since these sets form an interval restricted to S_1, there is no set of $\mathcal{F}^*$ containing b and c but not a. We argue similarly if all sets of $\mathcal{F}^*$ containing elements of S_1 and S_2 contain the largest element of S_1.

(3) If a, b, c all come from different ground sets, then no set in $\mathcal{F}^*$ contains all three elements, since any set of $\mathcal{F}^*$ intersects at most two ground sets.

In all cases we showed that one of the patterns 110, 101, 011, and 111 is missing for any elements $a, b, c \in X$. This finishes the proof of $VC(\mathcal{F}) \leq 2$.

2.3 Dual VC-dimension 2

It is easy to see that in general the dual VC-dimension of $\mathcal{F}$ is at least 2, e.g., the sets $A_{i,1}^1, A_{i,2}^1$ are shattered by the elements $x_{i,2}^2$ (pattern 11), $x_{i,1}^2$ (pattern 10), $x_{i,1}^1$ (pattern 01), and any element of the form $y_{ij,uv}^\ell$ (pattern 00).

To show that the dual VC-dimension of $\mathcal{F}$ is at most 2, we first reduce to the set system $\mathcal{F}^*$ like in the previous section. Consider any sets $M_1, M_2, M_3 \in \mathcal{C}$ and assume for the sake of contradiction that they realize all of the patterns 110, 101, 011, and 111 in $\mathcal{F}$ but the corresponding sets M_1^*, M_2^*, M_3^* do not realize all patterns 110, 101, 011, 111 in $\mathcal{F}^*$. Without loss of generality, assume that M_1 is of the form $D_{ij,uv}$ and its element $y^4_{ji,vu}$ is also contained in M_2, so that $y^4_{ji,vu}$ induces one of the patterns 110 or 111. This yields that M_2 is of the form $B^\ell_{ji,wz}$ for appropriate $\ell \in [5], wz \in E_{ji}$. However, any such set has only one element in common with $D_{ij,uv}$, namely $y^4_{ji,vu}$. Thus, one of the patterns 110, 111 is missing, which is a contradiction. Hence, if we show that $\mathcal{F}^*$ contains no three sets realizing all patterns 110, 101, 011, and 111, then no three sets of $\mathcal{F}$ are shattered.

Consider any sets M_1^*, M_2^*, M_3^* of $\mathcal{F}^*$ and assume for the sake of contradiction that they realize all of the patterns 110, 101, 011, and 111. Restricted to any ground set S the sets M_1^*, M_2^*, M_3^* form intervals, and thus S cannot induce all four patterns 110, 101, 011, and 111 on M_1^*, M_2^*, M_3^* (as can be checked easily and follows from the proof of the well-known fact that intervals have dual VC-dimension 2).

Hence, without loss of generality there is a ground set S_1 with an element inducing the pattern 111 and another ground set S_2 with an element inducing the pattern 110 on M_1^*, M_2^*, M_3^*. Note that M_1^* and M_2^* are contained in $S_1 \cup S_2$, since every set of $\mathcal{F}^*$ intersects at most two ground sets. By Observation 4, since M_1^* and M_2^* intersect both S_1 and S_2, they both contain the smallest or largest element e_1 of S_1 and the smallest or largest element e_2 of S_2. In particular, restricted to S_1 we have without loss of generality $M_1^* \subseteq M_2^*$. Now, if M_3^* does not intersect S_2, then the pattern 101 is missing, since only elements of S_1 can be contained in both M_1^* and M_3^*, but any such element is also contained in M_2^*. Otherwise, M_3^* also contains e_1 and e_2, so that restricted to S_1 we have a linear ordering $M^*_{\pi(1)} \subseteq M^*_{\pi(2)} \subseteq M^*_{\pi(3)}$ and restricted to S_2 we have a linear ordering $M^*_{\sigma(1)} \subseteq M^*_{\sigma(2)} \subseteq M^*_{\sigma(3)}$ (for permutations π, σ). However, two linear orderings can only induce two of the patterns 110, 101, and 011. This contradicts M_1^*, M_2^*, M_3^* realizing all patterns 110, 101, 011, and 111, and finishes the proof of $VC(\mathcal{F}^T) \leq 2$.

3 Efficiently solvable classes of Hitting Set

In this section, we consider efficiently solvable special cases of Hitting Set. The following result can be seen a warmup for a similar but more involved argument in § 3.1.

▶ **Theorem 5.** *Hitting Set is polynomial-time solvable on set systems of VC-dimension* 1 *and on set systems of dual VC-dimension* 1.

Proof. Let $\mathcal{F} = (X, \mathcal{C})$ be a set system of VC-dimension 1. If every set in $\mathcal{C}$ has non-empty intersection with some $\{x, y\} \subseteq X$ then $\{x, y\}$ is a hitting set of size 2, and the minimal hitting set can be found by a brute-force search over all subsets of X of size 1 or 2.

Assume therefore that there is no pair $\{x, y\} \subseteq X$ which hits every set in $\mathcal{C}$. Let $x, y \in X$. We say that x *dominates* y if every set in $\mathcal{C}$ which contains y also contains x. Note that if x dominates y, then removing y from all sets in $\mathcal{C}$ does not affect the size of the minimum hitting set. Let $\{x, y\}$ be a two-element set which is contained in some set $A \in \mathcal{C}$. We claim that x dominates y or y dominates x. Indeed, (x, y) realizes the patterns 00 (by the first observation that no pair $\{x, y\}$ hits every set in $\mathcal{C}$) and 11 (since $\{x, y\} \subset A$). Since $\{x, y\}$ is not shattered, one of 01 and 10 must be missing – implying that one of x or y dominates the other. We proceed by repeatedly removing dominated elements, until we are left with singleton sets which immediately yields the minimum hitting set.

Now consider the case of dual VC-dimension 1. This condition implies that for every pair of sets $A, B \in \mathcal{C}$, at least one of the following holds: $A \subseteq B$, $B \subseteq A$, $A \cap B = \emptyset$, or $A \cup B = X$.

If there exist $A, B \in \mathcal{C}$ such that $A \subseteq B$, then we can consider the modified set system in which B is removed, without affecting the size of a minimal hitting set. Thus, we may assume that no set in $\mathcal{C}$ contains another set of $\mathcal{C}$. If the sets in $\mathcal{C}$ are all pairwise disjoint, then the minimum hitting set contains an arbitrary element from each set, and can easily be found. Thus, we can assume that there exist two sets $A, B \in \mathcal{C}$ such that $A \cup B = X$. Any other set $C \in \mathcal{C}$ intersects both A and B (otherwise it would be contained in one of them). From this we conclude that every $C \in \mathcal{C} \setminus \{A, B\}$ satisfies $C \cup A = X$, and $C \cup B = X$, or equivalently C must contain $B \setminus A$ and $A \setminus B$. It follows that the size of the minimum hitting set is at most 2, and thus can be computed in polynomial time. ◀

The Sauer-Perles-Shelah Lemma implies that set systems of VC-dimension 1 are $(k, k+1)$-systems for every k, and in particular they are $(3, 4)$-systems. Thus, a natural question is whether Hitting Set is polynomial-time solvable for every $(3, 4)$-system. We next show that the answer is yes, even for the more general case of $(3, 5)$-systems, thus extending Theorem 5.

3.1 (3,5)-systems

In this subsection we prove that Hitting Set on $(3, 5)$-systems is solvable in polynomial time. Before presenting the algorithm, we briefly observe that the class of $(3, 5)$-systems is a proper generalization of Edge Cover instances (i.e. where every element occurs in exactly two sets). More generally, an Edge Cover instance is a $(k, k + \lfloor k/2 \rfloor + 1)$-system for any $k \geq 1$. This is because the incidence matrix of an Edge Cover instance can have at most $2k$ one-entries in any k columns, and every collection of $k + \lfloor k/2 \rfloor + 2$ distinct k-vectors has at least $2k + 1$ one-entries. To see that Edge Cover instances are a proper subset of $(3, 5)$-systems, observe that in a $(k, k + \lfloor k/2 \rfloor + 1)$-system, an element can occur in an arbitrary number of sets.

▶ **Theorem 2** (restated). *Hitting set on $(3, 5)$-systems is in P.*

Let $\mathcal{F} = (X, \mathcal{C})$ be a $(3, 5)$-system. We present a polynomial-time algorithm which outputs a minimum hitting set for $\mathcal{F}$. First check whether $\emptyset \in \mathcal{C}$; if this is the case, then report "no solution". Otherwise perform the following preprocessing steps repeatedly, until none of the steps can be performed.

0. If $\mathcal{F}$ is not connected, i.e. there are set systems $(X_1, \mathcal{C}_1)$, $(X_2, \mathcal{C}_2)$ with disjoint X_1, X_2 and $\mathcal{F} = (X_1 \cup X_2, \mathcal{C}_1 \cup \mathcal{C}_2)$, then recursively solve $(X_1, \mathcal{C}_1)$ and $(X_2, \mathcal{C}_2)$ and return the union of the solutions.

1. If $\{x, y, z\} \subseteq X$, and the pattern 000 is not realized on (x, y, z), then a minimum hitting set is of size at most 3, and we find it by exhaustive search over all subsets of size at most 3.

2. If $\{x, y\} \subseteq X$, and the pattern 01 is not realized on (x, y), then remove y from X, as x dominates y (whenever y occurs, x also occurs).

3. If $A, B \in \mathcal{C}$ such that $A \subseteq B$, then remove B from $\mathcal{C}$, as whenever we hit A, we also hit B.

4. If there is a singleton set $\{x\} \in \mathcal{C}$, then add x to the solution, remove x from X and remove every set containing x from $\mathcal{C}$.

5. (only if steps $0, \ldots, 4$ cannot be applied) If $A, B, C \in \mathcal{C}$, and there is an element $x \in (A \cap B \cap C)$, then add x to the solution, remove x from X and remove every set containing x from $\mathcal{C}$.

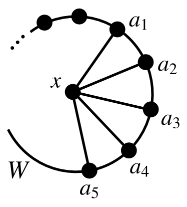


Figure 2 Illustration of the proof of Lemma 6.

Observe that after every preprocessing step the resulting set system is still a $(3,5)$-system. Moreover, after the preprocessing, every element of X is contained in exactly two sets of $\mathcal{C}$ (otherwise rule 2,4, or 5 is applicable). In other words, after preprocessing, $(X,\mathcal{C})$ is an instance of Edge Cover - such an instance can be solved in polynomial time by computing a maximum matching, and then augmenting with additional edges to cover the unmatched vertices [17]. The total asymptotic running time (including the time of the preprocessing) is dominated by the time needed to find a maximum matching in a graph with $|\mathcal{C}|$ vertices and $|X|$ edges.

The correctness of the algorithm hinges on the validity of the preprocessing steps. Note that only step 5 is not trivially valid. Theorem 2 thus follows from the following claim.

▶ **Lemma 6.** *If preprocessing steps $0,\ldots,4$ cannot be applied, and if there exists an element x contained in at least three sets of $\mathcal{C}$, then x is part of any minimum hitting set.*

Proof. We make use of the following claim that we prove later.

▶ **Lemma 7.** *If preprocessing steps $0,\ldots,4$ cannot be applied, then for any two sets $A,B\in\mathcal{C}$, we have $|A\cap B|\le 1$.*

Suppose that there is an element $x\in X$ contained in t sets of $\mathcal{C}$, where $t\ge 3$, and let $A_1,\ldots,A_t$ denote the sets containing x. Each of these sets must also contain some element other than x (by preprocessing step 4), so let $a_1\in A_1\setminus\{x\},\ldots,a_t\in A_t\setminus\{x\}$. Observe that since $t\ge 3$, Lemma 7 implies that for all $i\ne j$: $A_i\cap A_j=\{x\}$ and therefore, $a_1,\ldots,a_t$ are distinct.

The proof proceeds by showing that every hitting set that does not contain x must contain $a_1,\ldots,a_t$, and that replacing $a_2,\ldots,a_t$ by x preserves the property of being a hitting set.

For every a_i there exists a set $A_i'\in\mathcal{C}$ such that $a_i\in A_i'$ and $x\notin A_i'$, as otherwise a_i would have been deleted in step 2 of the preprocessing, as it is dominated by x.

We show that $A_i'=A_j'$ for all $i,j\le t$. Suppose first, towards contradiction, that there exist two indices i and j, such that $a_i\notin A_j'$. Let k be an index $(1\le k\le t)$ different from i and j. In this case, the triple (x,a_i,a_j) realizes the patterns 000 (by preprocessing step 1), 100 (from A_k), 110 (from A_i), 101 (from A_j), 001 (from A_j'), and either 011 or 010 (from A_i'), in both cases contradicting the hypothesis that $\mathcal{F}$ is a $(3,5)$-system. We conclude that for all indices i and j, $a_i\in A_j'$. Thus, we have $\{a_i,a_j\}\subseteq A_i'\cap A_j'$ for all i,j. From Lemma 7 we conclude that $A_i'=A_j'$. Let us denote $W=A_1'=\cdots=A_t'$.

Since the above reasoning holds for any element in $A_i\setminus\{x\}$, we even have $W\supset A_i\setminus\{x\}$ for all $i\le t$. Observe that $|A_i|=2$, for all $i\le t$, as otherwise W would intersect A_i in more than one element, contradicting Lemma 7. See Figure 2 for an illustration.

Suppose that there exists a set $Q\in\mathcal{C}$, such that $a_i\in Q$, and $Q\ne A_i$, and $Q\ne W$, and let j, k be two indices different from i. Note that since $|Q\cap W|,|Q\cap A_i|\le 1$, and $a_i\in Q\cap A_i$, and $a_i\in Q\cap W$, it follows that $a_j\notin Q$ and $x\notin Q$. Thus the triple (x,a_i,a_j) realizes the patterns 000 (by preprocessing step 1), 100 (from A_k), 110 (from A_i), 101 (from

A_j), 010 (from Q), and 011 (from W), contradicting the hypothesis that $\mathcal{F}$ is a $(3,5)$-system. Therefore, A_i and W are the only sets in $\mathcal{C}$ containing a_i.

Let H be a hitting set that does not contain x. Since each of the sets $A_1,\dots,A_t$ is of size 2, in order to hit them we must have $a_1,\dots,a_t \in H$. However, as a_i (for all i) is contained only in A_i and W, we can improve the solution by removing $a_2,\dots,a_t$ and adding x. In this way, all the sets containing the removed elements are still hit. This means that H is not a minimum hitting set, and thus preprocessing step 5 is justified. ◀

It remains to prove Lemma 7. We proceed via two intermediate claims.

▶ **Lemma 8.** *Let $\mathcal{F} = (X,\mathcal{C})$ be a $(3,5)$-system such that preprocessing steps $0,\dots,4$ cannot be applied. Let $Y \subseteq X$ with $|Y| = k \geq 4$. Then the following properties are equivalent. If they are satisfied, we say that $\mathcal{F}$ contains a B_k-system (induced by Y).*

- *$PR_{\mathcal{F}}(Y)$ contains all subsets of Y of size $k-1$,*
- *$PR_{\mathcal{F}}(Y)$ contains no set S with $0 < |S| \leq k-2$.*

Proof. To see that the second property implies the first, observe that all 01-patterns must be present (by preprocessing step 2), and these patterns can only be realized by having all sets of the form $Y \setminus \{y\}$ for $y \in Y$. To see that the first property implies the second, assume for contradiction that there is a set $S \in PR_{\mathcal{F}}(Y)$ with $0 < |S| \leq k-2$. Then S realizes pattern 100 on some y_1, y_2, y_3. Since patterns 110, 101, 011, 111 are realized (by the first property and $k \geq 4$) and 000 is realized (by preprocessing step 1), we obtain a contradiction. ◀

▶ **Lemma 9.** *Let $\mathcal{F} = (X,\mathcal{C})$ be a $(3,5)$-system such that preprocessing steps $0,\dots,4$ cannot be applied. If there are two sets $A, B \in \mathcal{C}$ with $|A \cap B| \geq 2$, then there exists a set $Q = \{x,y,z,t\} \subseteq X$ with the following properties:*

(i) *Q induces a B_4-system on $\mathcal{F}$, and*
(ii) *Q is a hitting set of $\mathcal{F}$.*

Proof.

(i) Consider two elements $x, y \in A \cap B$. By preprocessing step 3 there exist $z \in A \setminus B$ and $t \in B \setminus A$. On the triple (z,x,y) we realize 000 (by preprocessing step 1), 111 (by A), 011 (by B). The missing 01 patterns on (x,z), (y,z), (x,y), (y,x) can be realized with the assumption that $\mathcal{F}$ is a $(3,5)$-system, only if the remaining two patterns on (z,x,y) are 101 and 110. A similar argument shows that on the triple (t,x,y) the following five patterns are realized: 000 (by preprocessing step 1), 111 (by B), 011 (by A), and to obtain 01 on each pair: 101 and 110. Observe that if (x,y) realizes 00, 01, or 10, then the pattern on z and t is uniquely determined. This yields on the tuple (z,t,x,y) the following patterns: 0000, 1110, 1101 (by joining uniquely the patterns that realize 00, 01, and 10 on (x,y)). We need to realize 01 on both (z,t) and (t,z). The only way to achieve this is with the patterns 0111 and 1011 on (z,x,y,t). With this we conclude that $PR_{\mathcal{F}}(\{x,y,z,t\})$ contains all possible sets of size 3, satisfying the first condition of Lemma 8 and hence $\mathcal{F}$ contains a B_4 system induced by $\{x,y,z,t\}$.

(ii) Suppose for contradiction that $\{x,y,z,t\}$ is not a hitting set of $\mathcal{F}$. Pick $D, D' \in \mathcal{C}$ such that $D \cap \{x,y,z,t\} = \emptyset$, $D' \cap D \neq \emptyset$, and $D' \cap \{x,y,z,t\} \neq \emptyset$ (such D, D' exist since $\mathcal{F}$ is connected and $\{x,y,z,t\}$ is not a hitting set of $\mathcal{F}$). Let $s \in D \cap D'$. Observe that $|D' \cap \{x,y,z,t\}| \geq 3$ (by Lemma 8). Let x_1, x_2, x_3 be distinct elements from $\{x,y,z,t\}$ that belong to D'. Let $E \in \mathcal{C}$ such that E induces the pattern 110 on (x_1,x_2,x_3). Such an E exists since $\{x,y,z,t\}$ induce a B_4-system on $\mathcal{F}$. We consider two cases: (1) when

$s \in E$, and (2) when $s \notin E$. (1) On the triple (s, x_2, x_3), we have the patterns 000 (from the preprocessing), 111 (by D'), 100 (by D), 110 (by E), and to get 01 on (x_2, x_3) and 01 (s, x_2), we need at least two more patterns on (s, x_2, x_3), contradicting that $\mathcal{F}$ is a $(3, 5)$-system. (2) On the triple (s, x_1, x_2) we have the patterns 000 (from the preprocessing), 111 (by D'), 100 (by D), and 011 (from E). To realize 01 and 10 on (x_1, x_2), we need two more patterns on (s, x_1, x_2), contradicting that $\mathcal{F}$ is a $(3, 5)$-system. We reached a contradiction, proving that such a D cannot exist, and hence $\{x, y, z, t\}$ is a hitting set of $\mathcal{F}$. ◀

Suppose that preprocessing steps $0, \ldots, 4$ cannot be applied to $\mathcal{F}$, and there exist sets $A, B \in \mathcal{C}$ with $|A \cap B| \geq 2$. Then, from Lemma 9 it follows that there exists a hitting set $\{x, y, z, t\}$ of $\mathcal{F}$, such that $\{x, y, z, t\}$ induces a B_4-system in $\mathcal{F}$. From the definition of hitting set it follows that $PR_{\mathcal{F}}(\{x, y, z, t\})$ does not contain the empty set. From Lemma 8 it follows that $PR_{\mathcal{F}}(\{x, y, z, t\})$ does not contain a set of size 1. Hence, on the triple (x, y, z) the pattern 000 cannot be realized, contradicting the assumption that preprocessing step 1 cannot be applied.

This concludes the proof of Lemma 7 and the proof of correctness for the algorithm.

The following theorem gives a sharp threshold on the complexity of Hitting Set by showing the NP-hardness of Hitting Set on $(3, 6)$-set systems.

▶ **Theorem 3** (restated). *Hitting set on* $(3, 6)$*-systems is in NP-hard.*

The proof follows by considering the Hitting Set instance that corresponds to Vertex Cover in a triangle-free graph. Indeed, the following two observations establish NP-hardness and the $(3, 6)$-property: (i) Vertex Cover in triangle-free graphs is NP-hard. This can be seen by taking an arbitrary Vertex Cover instance and splitting every edge by adding two internal vertices. The resulting graph is triangle-free. Also, the size of its optimum vertex cover is the original plus the number of edges in the original graph, and (ii) in a triangle-free Vertex Cover instance, on any three elements (vertices) the pattern 111 and one of the patterns in $\{011, 110, 101\}$ are not realized.

References

1 Pankaj K. Agarwal and Jiangwei Pan. Near-linear algorithms for geometric hitting sets and set covers. In *30th Annual Symposium on Computational Geometry, SOCG'14, Kyoto, Japan, June 08 - 11, 2014*, page 271, 2014. doi:10.1145/2582112.2582152.

2 Noga Alon, Shay Moran, and Amir Yehudayoff. Sign rank, VC dimension and spectral gaps. *Electronic Colloquium on Computational Complexity (ECCC)*, 21:135, 2014. URL: http://eccc.hpi-web.de/report/2014/135.

3 Noga Alon, Dana Moshkovitz, and Shmuel Safra. Algorithmic construction of sets for k-restrictions. *ACM Transactions on Algorithms*, 2(2):153–177, 2006. doi:10.1145/1150334.1150336.

4 Anselm Blumer, A. Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Learnability and the Vapnik-Chervonenkis Dimension. *J. ACM*, 36(4):929–965, October 1989. doi:10.1145/76359.76371.

5 Nicolas Bousquet, Aurélie Lagoutte, Zhentao Li, Aline Parreau, and Stéphan Thomassé. Identifying codes in hereditary classes of graphs and vc-dimension. *CoRR*, abs/1407.5833, 2014. URL: http://arxiv.org/abs/1407.5833.

6 Olivier Bousquet, Stéphane Boucheron, and Gábor Lugosi. Introduction to statistical learning theory. In Olivier Bousquet, Ulrike von Luxburg, and Gunnar Rätsch, editors, *Advanced Lectures on Machine Learning: ML Summer Schools 2003, Canberra, Australia, February*

2 - 14, 2003, Tübingen, Germany, August 4 - 16, 2003, Revised Lectures, pages 169–207. Springer, Berlin, Heidelberg, 2004. doi:10.1007/978-3-540-28650-9_8.

7 Hervé Brönnimann and Michael T. Goodrich. Almost optimal set covers in finite vc-dimension. *Discrete & Computational Geometry*, 14(4):463–479, 1995. doi:10.1007/BF02570718.

8 Bernard Chazelle. *The discrepancy method - randomness and complexity.* Cambridge University Press, 2001.

9 Kenneth L. Clarkson and Kasturi R. Varadarajan. Improved approximation algorithms for geometric set cover. *Discrete & Computational Geometry*, 37(1):43–58, 2007. doi:10.1007/s00454-006-1273-8.

10 Frank K. H. A. Dehne, Michael R. Fellows, Michael A. Langston, Frances A. Rosamond, and Kim Stevens. An $o(2^{o(k)}n^3)$ FPT algorithm for the undirected feedback vertex set problem. *Theory Comput. Syst.*, 41(3):479–492, 2007. doi:10.1007/s00224-007-1345-z.

11 Michael Dom, MichaelR. Fellows, and FrancesA. Rosamond. Parameterized complexity of stabbing rectangles and squares in the plane. In Sandip Das and Ryuhei Uehara, editors, *WALCOM: Algorithms and Computation*, volume 5431 of *Lecture Notes in Computer Science*, pages 298–309. Springer Berlin Heidelberg, 2009. doi:10.1007/978-3-642-00202-1_26.

12 Rod G. Downey and Michael R. Fellows. *Parameterized Complexity.* Springer-Verlag, New York, 1999.

13 P. Erdős. Personal reminiscences and remarks on the mathematical work of Tibor Gallai. *Combinatorica*, 2(3):207–212, 1982. doi:10.1007/BF02579228.

14 Guy Even, Dror Rawitz, and Shimon Shahar. Hitting sets when the VC-dimension is small. *Inf. Process. Lett.*, 95(2):358–362, 2005. doi:10.1016/j.ipl.2005.03.010.

15 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2006.

16 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. doi:10.1137/S0097539702419649.

17 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

18 Panos Giannopoulos, Christian Knauer, and Sue Whitesides. Parameterized complexity of geometric problems. *Comput. J.*, 51(3):372–384, 2008. doi:10.1093/comjnl/bxm053.

19 Magdalene Grantson and Christos Levcopoulos. Covering a set of points with a minimum number of lines. In *Algorithms and Complexity, 6th Italian Conference, CIAC 2006, Rome, Italy, May 29-31, 2006, Proceedings*, pages 6–17, 2006. doi:10.1007/11758471_4.

20 Jiong Guo and Rolf Niedermeier. Exact algorithms and applications for tree-like weighted set cover. *J. Discrete Algorithms*, 4(4):608–622, 2006. doi:10.1016/j.jda.2005.07.005.

21 Jiong Guo, Rolf Niedermeier, and Sebastian Wernicke. Parameterized complexity of generalized vertex cover problems. In *Algorithms and Data Structures, 9th International Workshop, WADS 2005, Waterloo, Canada, August 15-17, 2005, Proceedings*, pages 36–48, 2005. doi:10.1007/11534273_5.

22 Sariel Har-Peled and Mira Lee. Weighted geometric set cover problems revisited. *JoCG*, 3(1):65–85, 2012. URL: http://jocg.org/index.php/jocg/article/view/77.

23 David Haussler and Emo Welzl. epsilon-nets and simplex range queries. *Discrete & Computational Geometry*, 2:127–151, 1987. doi:10.1007/BF02187876.

24 Pinar Heggernes, Dieter Kratsch, Daniel Lokshtanov, Venkatesh Raman, and Saket Saurabh. Fixed-parameter algorithms for cochromatic number and disjoint rectangle stabbing via iterative localization. *Inf. Comput.*, 231:109–116, 2013. doi:10.1016/j.ic.2013.08.007.

25 Danny Hermelin, Matthias Mnich, Erik Jan van Leeuwen, and Gerhard J. Woeginger. Domination when the stars are out. In *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part I*, pages 462–473, 2011. doi:10.1007/978-3-642-22006-7_39.

26 James King (http://cstheory.stackexchange.com/users/196/james king). Parameterized complexity of hitting set in finite vc-dimension. Theoretical Computer Science Stack Exchange. URL: http://cstheory.stackexchange.com/q/182.

27 Stefan Langerman and Pat Morin. Covering things with things. *Discrete & Computational Geometry*, 33(4):717–729, 2005. doi:10.1007/s00454-004-1108-4.

28 Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *J. ACM*, 41(5):960–981, September 1994. doi:10.1145/185675.306789.

29 Dániel Marx. Parameterized complexity of independence and domination on geometric graphs. In *Parameterized and Exact Computation, Second International Workshop, IWPEC 2006, Zürich, Switzerland, September 13-15, 2006, Proceedings*, pages 154–165, 2006. doi:10.1007/11847250_14.

30 Dániel Marx. Can you beat treewidth? In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, FOCS'07, pages 169–179, Washington, DC, USA, 2007. IEEE Computer Society. doi:10.1109/FOCS.2007.18.

31 Jiri Matousek. *Lectures on Discrete Geometry*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

32 Shay Moran, Amir Shpilka, Avi Wigderson, and Amir Yehudayoff. Teaching and compressing for low vc-dimension. *Proceedings of the 56th Annual IEEE Symposium on Foundations of Computer Science*, 2015.

33 Nabil H. Mustafa and Saurabh Ray. Improved results on geometric hitting set problems. *Discrete & Computational Geometry*, 44(4):883–895, 2010. doi:10.1007/s00454-010-9285-9.

34 Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*, volume 31 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, Oxford, 2006.

35 Geevarghese Philip, Venkatesh Raman, and Somnath Sikdar. Solving dominating set in larger classes of graphs: FPT algorithms and polynomial kernels. In *Algorithms - ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*, pages 694–705, 2009. doi:10.1007/978-3-642-04128-0_62.

36 Venkatesh Raman and Saket Saurabh. Short cycles make w-hard problems hard: Fpt algorithms for w-hard problems with no short cycles. *Algorithmica*, 52(2):203–225, 2008. doi:10.1007/s00453-007-9148-9.

37 Norbert Sauer. On the density of families of sets. *J. Comb. Theory, Ser. A*, 13(1):145–147, 1972. doi:10.1016/0097-3165(72)90019-2.

38 Jan Arne Telle and Yngve Villanger. FPT algorithms for domination in biclique-free graphs. In *Algorithms - ESA 2012 - 20th Annual European Symposium, Ljubljana, Slovenia, September 10-12, 2012. Proceedings*, pages 802–812, 2012. doi:10.1007/978-3-642-33090-2_69.

39 Vladimir N Vapnik and A Ya Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability & Its Applications*, 16(2):264–280, 1971.

A Complexity claims in Table 1

The exact definitions and the complexity results for the various Hitting Set instances can be found by following the respective references. Edge Cover and Vertex Cover are standard problems, described e.g. in [17]. The study of Hitting Set (a.k.a. *transversal* problem) on line intervals goes back to early work of Gallai [13]. The folklore polynomial-time algorithm follows directly from his combinatorial observations.

The fact that Hitting Set is FPT in set systems defined by pseudolines is folklore, and can easily be explained by the property that any two points are contained in at most one set (i.e. line). A generalization of this property holds for arrangements of hyperplanes in $\mathbb{R}^d$: Here any d points are contained in at most one hyperplane. Both properties are subsumed by the biclique-free property, or equivalently the avoidance of a submatrix consisting of all 1s[8]. The definition of the halfspace arrangement problem and a simple proof of hardness in three dimensions is included in § C.

B VC-dimension claims in Table 1

The computation of the VC-dimension is an easy exercise for most of the examples in Table 1. We mention that for some of the problems (especially those related to graphs), the value of the VC-dimension seems not to have been explicitly computed in the literature. In some cases this computation leads to approximation-results (via Brönnimann and Goodrich [7]) that match the best known approximation ratio obtained via other means. We give a brief overview of the examples listed in Table 1.

The set systems of Vertex Cover and Edge Cover instances are simple: Each set is of size 2, respectively, each element appears in 2 sets. In both cases it is easy to see that both the VC-dimension and the dual VC-dimension is at most 2.

In Tree-Like Hitting Set, the sets are restricted to be subtrees of a tree. Here we can shatter an arbitrary number of elements: Consider the set of all leaves of a tree, and pick any subset of the leaves. Observe that there is a subtree that contains exactly the picked set of leaves and no other leaves. A similar argument holds for the Feedback Vertex Set problem. In a complete graph, color half of the vertices blue, and observe that if we pick any set of blue vertices (possibly the empty set), there can be a cycle containing all the blue vertices of the chosen set and no other blue vertices.

The set system associated with the *Dominating Set* problem is the set of all closed vertex neighborhoods of a graph. Since the incidence matrix of this set system is a symmetric square matrix, the dual VC-dimension is the same as the VC-dimension.

Triangle-free graphs: The VC-dimension can be arbitrarily large. To see this, consider an independent set X of size n, and add 2^n further vertices, each connected to a different subset of X. Observe that X is shattered, while the graph is triangle-free.

A similar argument holds for *graphs free of induced* $K_{t,1}$, for $t \geq 3$. Consider a k-clique X and a 2^k-clique Y, and for each subset $X' \subseteq X$ (including the empty set), connect one vertex of Y to X', and to none of the vertices in $X \setminus X'$. Clearly X is shattered, and thus the VC-dimension is at least k. If the constructed graph contains an induced $K_{t,1}$ for $t \geq 3$, then at least two non-connected vertices of the induced subgraph must be both in X or both in Y. This is a contradiction, since X and Y are cliques.

Planar graphs: A simple case-analysis shows that if a set of five vertices is shattered by the closed vertex neighborhood of a graph, then the graph must contain $K_{3,3}$ or K_5 as a

[8] In the literature, the fixed-parameter tractability on biclique-free instances is shown for *Dominating Set*, but the result easily transfers to *Hitting Set*.

subgraph, and thus it cannot be planar. On the other hand, it is easy to construct a planar graph instance where a set of four vertices is shattered.

Graphs of girth at least 5: A simple case analysis shows that if 3 vertices are shattered, then the graph has a triangle or a cycle of length 4. On the other hand, 2 vertices can be shattered in this graph class. Therefore, the VC-dimension is 2 (see e.g. [5]). An immediate consequence of the boundedness of the VC-dimension is an $O(\log k)$-factor approximation algorithm for Dominating Set on this class of graphs, as a corollary of the result of Brönnimann and Goodrich [7]. A matching result was obtained by Raman and Saurabh [36] using sophisticated techniques.

The claim for *unit disk graphs* follows from simple geometric arguments (see e.g. [5]). For *graphs avoiding* $K_{t,t}$, the incidence matrix can not contain a t-by-t all-1s submatrix. It is easy to check that a matrix with $t + \lceil \log_2 t \rceil$ columns that contains all possible 0/1 vectors on its rows contains such a submatrix, whereas a similar matrix with one fewer columns does not. The claim on the VC-dimension follows.

For most geometric set systems in Table 1, the VC-dimension is well known from the computational geometry and learning theory literature.

Line intervals: Given three points on a line, no interval can contain the two outer points without containing the one in the middle. Thus the VC-dimension is at most 2. If 3 intervals share a common point, then one interval is in the union of the other two. This ensures that no three intervals can be shattered, thus the dual VC-dimension is at most 2 as well. Both values are tight.

Pseudolines: Since any two sets intersect at most once, a 2-by-2 submatrix of 1s can not exist in the incidence matrix. This implies that no 3 points can be shattered by the set system or by its dual. On the other hand, 2 points can be shattered by both set systems. The claim follows. The boundedness of the VC-dimension yields an $O(\log k)$-factor approximation algorithm for this problem. A similar result for a special case of the problem was obtained by Grantson and Levcopoulos [19] using different techniques.

Halfplanes: It is easy to show that not every subset of size 2 of a set of 4 points in the plane can be realized by halfplanes. On the other hand, for 3 points in general position every subset can be realized. Thus the VC-dimension is 3. Observe that the dual VC-dimension is 2, since three lines create at most 7 cells in the plane, therefore not all patterns on 3 sets can be realized. On the other hand, the 4 patterns on 2 sets can be realized.

The claims for *hyperplanes in* $\mathbb{R}^d$, *unit disks* and *unit squares* follow from similar geometric arguments and we omit them.

Rectangle Stabbing: In this problem the set system is defined by the incidences between a set of axis parallel rectangles (playing the role of sets), and a set of horizontal and vertical lines (playing the role of elements). Note that at most 4 lines can be shattered, as three lines with the same orientation can not be shattered (by the same argument as for intervals), thus the VC-dimension is at most 4 (this can be reached). However, the families of instances constructed in the hardness proof of Dom et al. [11] have VC-dimension 3. The same value is obtained for the dual VC-dimension. We omit the details. For *Disjoint Rectangle Stabbing* the VC-dimension is 2, by an argument similar to the one used for line intervals.

C Hardness of Hitting Set for halfspaces in $d \geq 3$

We prove now that the halfspace arrangement problem in R^d is $W[1]$-hard for $d \geq 3$. The halfspace arrangement problem is a special case of Hitting Set, defined as follows: The input has n points and n halfspaces in R^d, and a number k. The goal is to select k halfspaces such that each of the given points is contained in at least one of the k halfspaces. The special case for $d = 2$ appears in Table 1 as *halfplane arrangement*, and is known to be in P.

▶ **Theorem 10.** *In $R^d, d \geq 3$, the halfspace arrangement problem is NP-hard and even $W[1]$-hard.*

Proof. We reduce from Dominating Set on the intersection graphs of unit disks, which is known to be $W[1]$-hard [29]. First, observe that the problem stays $W[1]$-hard if we consider disks with unit radius r on the two-dimensional sphere $S^2 = \{(x_1, x_2, x_3) \in R^3 : x_1^2+x_2^2+x_3^2 = 1\}$. This is because we can embed any unit disk graph on a tiny part of the sphere S^2 that approximates the plane sufficiently well. Given the embedding with disk midpoints $p_1, ..., p_n$ on S^2 and the radius r, we want to find a dominating set of the intersection graph of these disks. This is equivalent to finding k indices $i_1, \ldots, i_k$ such that the disks of radius $2r$ around $p_{i_1}, .., p_{i_k}$ cover all points $p_1, \ldots, p_n$. We construct an equivalent instance of the halfspace arrangement problem as follows: The n points are $p_1, \ldots, p_n$. Let $0 < s < 1$. For $1 \leq i \leq n$ we add a halfspace $H_i = \{x \in R^3 : p_i.x \geq s\}$ with normal vector p_i. Crucially, observe that by setting s appropriately, $H_i \cap S^2$ is equal to the disk of radius $2r$ around p_i. Since all points p_i lie on S^2, it is equivalent whether we consider H_i or $H_i \cap S^2$ as sets (of the arrangement problem). ◀

New Algorithms, Better Bounds, and a Novel Model for Online Stochastic Matching[*][†]

Brian Brubach[1], Karthik Abinav Sankararaman[2], Aravind Srinivasan[3], and Pan Xu[4]

1 **Department of Computer Science, University of Maryland, College Park, USA**
`bbrubach@cs.umd.edu`

2 **Department of Computer Science, University of Maryland, College Park, USA**
`kabinav@cs.umd.edu`

3 **Department of Computer Science, University of Maryland, College Park, USA**
`srin@cs.umd.edu`

4 **Department of Computer Science, University of Maryland, College Park, USA**
`panxu@cs.umd.edu`

Abstract

Online matching has received significant attention over the last 15 years due to its close connection to Internet advertising. As the seminal work of Karp, Vazirani, and Vazirani has an optimal $(1 - 1/e)$ competitive ratio in the standard adversarial online model, much effort has gone into developing useful online models that incorporate some stochasticity in the arrival process. One such popular model is the "known I.I.D. model" where different customer-types arrive online from a known distribution. We develop algorithms with improved competitive ratios for some basic variants of this model with integral arrival rates, including: (a) the case of general weighted edges, where we improve the best-known ratio of 0.667 due to Haeupler, Mirrokni and Zadimoghaddam [11] to 0.705; and (b) the vertex-weighted case, where we improve the 0.7250 ratio of Jaillet and Lu [12] to 0.7299. We also consider two extensions, one is "known I.I.D." with non-integral arrival rate and stochastic rewards; the other is "known I.I.D." b-matching with non-integral arrival rate and stochastic rewards. We present a simple non-adaptive algorithm which works well simultaneously on the two extensions.

One of the key ingredients of our improvement is the following (offline) approach to bipartite-matching polytopes with additional constraints. We first add several valid constraints in order to get a good fractional solution $\mathbf{f}$; however, these give us less control over the structure of $\mathbf{f}$. We next *remove* all these additional constraints and randomly move from $\mathbf{f}$ to a feasible point on the matching polytope with all coordinates being from the set $\{0, 1/k, 2/k, \ldots, 1\}$ for a chosen integer k. The structure of this solution is inspired by Jaillet and Lu (*Mathematics of Operations Research*, 2013) and is a tractable structure for algorithm design and analysis. The appropriate random move preserves many of the removed constraints (approximately [exactly] with high probability [in expectation]). This underlies some of our improvements, and, we hope, could be of independent interest.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases Ad-Allocation, Online Matching, Randomized Algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.24

[*] A full version of the paper is available at `http://arxiv.org/abs/1606.063956`.
[†] Supported in part by NSF Awards CNS-1010789 and CCF-1422569, and a research award from Adobe, Inc.

1 Introduction

Applications to Internet advertising have driven the study of online matching problems in recent years [19]. In these problems, we consider a bipartite graph $G = (U, V, E)$ in which the set U is available offline while the vertices in V arrive online. Whenever some vertex v arrives, it must be matched immediately to at most one vertex in U. Each offline vertex u can be matched to at most one v or in the b-matching generalization, at most b vertices in V. In the context of Internet advertising, U is the set of advertisers, V is a set of impressions, and the edges E define the impressions that interest a particular advertiser. When v arrives, we must choose an available advertiser (if any) to match with it. Initially, we consider the case where $v \in V$ can be matched at most once. We later relax this condition to it being matched up to b times. Since advertising forms the key source of revenue for many large Internet companies, finding good matching algorithms and obtaining even small performance gains can have high impact. Additionally, bipartite matching is a fundamental combinatorial optimization problem. Hence, any improvements is interesting from a theoretical standpoint.

In the *stochastic known I.I.D.* model of arrival, we are given the bipartite graph in advance and each arriving vertex v is drawn with replacement from a known distribution on the vertices in V. This captures the fact that we often have background data about the impressions and can predict the frequency with which each type of impression will arrive. Edge-weighted matching [8] is a general model in the context of advertising: every advertiser gains a given revenue for being matched to a particular type of impression. Here, a *type* of impression refers to a class of users (e.g., a demographic group) who are interested in the same subset of advertisements. A special case of this model is vertex-weighted matching [1], where weights are associated only with the advertisers. In other words, a given advertiser has the same revenue generated for matching any of the user types interested in it.

In some modern business models, revenue is not generated upon matching advertisements, but only when a user *clicks* on the advertisement: this is the *pay-per-click* model. From background data, one can assign the probability of a particular advertisement being clicked by a type of user. Works including [20],[21] capture this notion by assigning a probability to each edge.

One unifying theme in most of our approaches is to use an LP benchmark with additional valid constraints that hold for the respective stochastic-arrival models, combined with some form of dependent rounding.

1.1 Related work

For readers not familiar with these problems, they are encouraged to first read parts of section 2 for formal definitions before getting into the related work. The study of online matching began with the seminal work of Karp, Vazirani, Vazirani [14], where they gave an optimal online algorithm for a version of the unweighted bipartite matching problem in which vertices arrive in adversarial order. Following that, a series of works have studied various related models. The book by Mehta [19] gives a detailed overview. The vertex-weighted version of this problem was introduced by Aggarwal, Goel and Karande [1], where they give an optimal $\left(1 - \frac{1}{e}\right)$ ratio for the adversarial arrival model. The edge-weighted setting has been studied in the adversarial model by Feldman, Korula, Mirrokni and Muthukrishnan [8], where they consider an additional relaxation of “free-disposal".

Beyond the adversarial model, these problems are studied under the name *stochastic matching*, where the online vertices either arrive in random order or are drawn I.I.D. from a known distribution. The works [5, 15, 16, 17] among others, study the random arrival order model; papers including [4, 9, 11, 12, 18, 6] study the I.I.D. arrival order model. Another variant of this problem is when the edges have stochastic rewards. Models with stochastic rewards have been previously studied by [20], [21] among others, but not in the known I.I.D. model.

Related Work in the Vertex-Weighted and Unweighted Settings: The vertex-weighted and unweighted settings have many results starting with Feldman, Mehta, Mirrokni and Muthukrishnan [9] who were the first to beat $1 - 1/e$ with a competitive ratio of 0.67 for the unweighted problem. This was improved by Manshadi, Gharan, and Saberi [18] to 0.705 with an adaptive algorithm. In addition, they showed that even in the unweighted variant with integral arrival rates, no algorithm can achieve a ratio better than $1 - e^{-2} \approx 0.86$. Finally, Jaillet and Lu [12] presented an adaptive algorithm which used a clever LP to achieve 0.725 and $1 - 2e^{-2} \approx 0.729$ for the vertex-weighted and unweighted problems, respectively.

Related Work in the Edge-Weighted Setting: For this model, Haeupler, Mirrokni, Zadimoghaddam [11] were the first to beat $1 - 1/e$ by achieving a competitive ratio of 0.667. They use a *discounted LP* with tighter constraints than the basic matching LP (a similar LP can be seen in 2.1) and they employ the *power of two choices* by constructing two matchings offline to guide their online algorithm.

Related Work in Online *b*-matching: In the model of b-matching, we assume each vertex u has a uniform capacity of b, where b is a parameter which is generally a large integral value. The model of unweighted b-matching can be viewed as a special case of Adwords or Display Ads. There is extensive literature for Adwords or Display Ads under various settings (see the book by Mehta [19]). In particular, [13] shows that their algorithm BALANCE is optimal for online b-matching under the adversarial model, which achieves a ratio of $1 - \frac{1}{(1+1/b)^b}$.

In this paper, we consider edge-weighted b-matching with stochastic rewards under the known I.I.D. model with arbitrary arrival rates. To the best of our knowledge, we are the first to consider this very general model. Devanur *et al* [7] gave an algorithm which achieves a ratio of $1 - 1/\sqrt{2\pi k}$ for the Adwords problem in the Unknown I.I.D. arrival model with knowledge of the optimal budget utilization and when the bid to budget ratios are at most $1/k$. Notice that even the problem of general edge-weighted b-matching with deterministic rewards cannot be captured in the Adwords model. Alaei *et al* [2] consider the Prophet-Inequality Matching problem, in which v arrives from a distinct (known) distribution $\mathcal{D}_t$, in each round t. They gave a $1 - 1/\sqrt{k+3}$ competitive algorithm, where k is the minimum capacity of u. They assume deterministic rewards however, and it is non-trivial to extend their result to the stochastic reward setting. In this paper, we present a very simple algorithm which achieves a ratio of $1 - b^{-1/2+\epsilon} - O(e^{-b^{2\epsilon}/3})$ for any given $\epsilon > 0$. It is worthwhile to see that our algorithm (5) can be trivially extended to the case where each vertex u has a distinct capacity b_u. The value of b in the final ratio would be replaced by $\min_{u \in U} b_u$.

2 Preliminaries

In the *Unweighted Online Known I.I.D. Stochastic Bipartite Matching* problem, we are given a bipartite graph $G = (U, V, E)$. The set U is available offline while the vertices v arrive online and are drawn with replacement from an I.I.D. distribution on V. For each $v \in V$, we are given an *arrival rate* r_v, which is the expected number of times v will arrive. With the exception of Sections 5 and 6, this paper will focus on the integral-arrival-rates setting where all $r_v \in \mathbb{Z}^+$. As described in [11], WLOG we can assume in this setting that $\forall v \in V, r_v = 1$. Let $n = \sum_{v \in V} r_v$ be the expected number of vertices arriving during the online phase.

In the **vertex-weighted** variant, every vertex $u \in U$ has a weight w_u and we seek a maximum weight matching. In the **edge-weighted** variant, every edge $e \in E$ has a weight w_e and we seek a maximum weight matching. In the **stochastic rewards** variant [1], additionally, each edge has a probability p_e and we seek to maximize the expected weight of the matching. In the **b-matching** model, every vertex in U can be matched upto b times. Throughout, we will use "WS" to refer to the worst case for various algorithms. Asymptotic assumption and notation: We will always assume n is large and analyze algorithms as n goes to infinity: e.g., if $x \leq 1-(1-2/n)^n$, we will just write this as "$x \leq 1-1/e^2$" instead of the more-accurate "$x \leq 1-1/e^2+o(1)$". These suppressed $o(1)$ terms will subtract at most $o(1)$ from our competitive ratios. Another fact to note is that the **competitive ratio** is defined slightly different than usual, for this set of problems (Similar to notation used in [19]). In particular, it is defined as $\frac{\mathbb{E}[ALG]}{\mathbb{E}[OPT]}$. Algorithms can be **adaptive** or **non-adaptive**. When v arrives, an adaptive algorithm can check which neighbors are still available to be matched, but a non-adaptive algorithm cannot.

2.1 LP Benchmark

We will use the following LP to upper bound the optimal offline solution and guide our algorithm. We will first show an LP for the unweighted variant, then describe changes for the vertex-weighted and edge-weighted settings. As usual, we have a variable f_e for each edge. Let $\partial(w)$ be the set of edges adjacent to a vertex $w \in U \cup V$ and let $f_w = \sum_{e \in \partial(w)} f_e$.

$$\begin{aligned}
\text{maximize} \quad & \sum_{e \in E} f_e && && (2.1)\\
\text{subject to} \quad & \sum_{e \in \partial(u)} f_e \leq 1 && \forall u \in U && (2.2)\\
& \sum_{e \in \partial(v)} f_e \leq 1 && \forall v \in V && (2.3)\\
& 0 \leq f_e \leq 1-1/e && \forall e \in E && (2.4)\\
& f_e + f_{e'} \leq 1-1/e^2 && \forall e, e' \in \partial(u), \forall u \in U && (2.5)
\end{aligned}$$

Variants: The objective function is: maximize $\sum_{u \in U} \sum_{e \in \partial(u)} f_e w_u$ in the vertex-weighted variant and maximize $\sum_{e \in E} f_e w_e$ in the edge-weighted variant.

Constraint 2.2 is the matching constraint for vertices in U. Constraint 2.3 is valid because each vertex in V has an arrival rate of 1. Constraint 2.4 is used in [18] and [11]. It captures the fact that the expected number of matches for any edge is at most $1-1/e$. This is valid for large n because the probability that a given vertex doesn't arrive after n rounds is $1/e$. Constraint 2.5 is similar to the previous one, but for pairs of edges. For any two neighbors of a given $u \in U$, the probability that neither of them arrive is $1/e^2$. Therefore, the sum of variables for any two distinct edges in $\partial(u)$ cannot exceed $1-1/e^2$. Notice that constraints 2.4 and 2.5 reduces the gap between the optimal LP solution and the performance

[1] The edge realization process is independent from one another. At each step, the algorithm "probes" the edge. With probability p_e the edge exists and with remaining probability it doesn't. Once realization of an edge is determined, it doesn't change for the rest of the algorithm

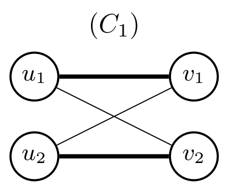


Figure 1 This cycle is the source of the negative result described by Jaillet and Lu [12]. Thick edges have $f_e = 2/3$ while thin edges have $f_e = 1/3$.

of the optimal online algorithm. In fact, without constraint 2.4, we cannot in general achieve a competitive ratio better than $1 - 1/e$.

2.2 Overview of vertex-weighted algorithm and contributions

A key challenge encountered by [12] was that their special LP could lead to length four cycles of type C_1 shown in Figure 1. In fact, they used this cycle to show that no algorithm could perform better than $1 - 2/e^2 \approx 0.7293$ using their LP. They mentioned that tighter LP constraints such as 2.4 and 2.5 in the LP from Section 2 could avoid this bottleneck, but they did not propose a technique to use them. Note that the $\{0, 1/3, 2/3\}$ solution produced by their LP was an essential component of their Random List algorithm.

We show a randomized rounding algorithm to construct a similar, simplified $\{0, 1/3, 2/3\}$ vector from the solution of a stricter benchmark LP. This allows for the inclusion of additional constraints, most importantly constraint 2.5. Using this rounding algorithm combined with tighter constraints, we will upper bound the probability of a vertex appearing in the cycle C_1 from Figure 1 at $2-3/e \approx 0.89$. (See Lemma 6) Additionally, we show how to deterministically break all other length four cycles which are not of type C_1 without creating any new cycles of type C_1. Finally, we describe an algorithm which utilizes these techniques to improve previous results in both the vertex-weighted and unweighted settings.

For this algorithm, we first solve the LP in Section 2 on the input graph. In Section 4, we show how to use the technique in sub-section 2.6 to obtain a sparse fractional vector. We then present a randomized online algorithm (similar to the one in [12]) which uses the sparse fractional vector as a guide to achieve a competitive ratio of 0.7299. Previously, there was gap between the best unweighted algorithm with a ratio of $1 - 2e^{-2}$ due to [12] and the negative result of $1 - e^{-2}$ due to [18]. We take a step towards closing that gap by showing that an algorithm can achieve $0.7299 > 1 - 2e^{-2}$ for both the unweighted and vertex-weighted variants with integral arrival rates.

2.3 Overview of edge-weighted algorithm and contributions

A challenge that arises in applying the *power of two choices* to this setting is when the same edge (u, v) is included in both matchings M_1 and M_2. In this case, the copy of (u, v) in M_2 can offer no benefit and a second arrival of v is wasted. To use an example from related work, Haeupler *et al.* [11] choose two matchings in the following way. M_1 is attained by solving an LP with constraints 2.2, 2.3 and 2.4 and rounding to an integral solution. M_2 is constructed by finding a maximum weight matching and removing any edges which have already been included in M_1. A key element of their proof is showing that the probability of an edge being removed from M_2 is at most $1 - 1/e \approx 0.63$.

The approach in this paper is to construct two or three matchings together in a correlated manner to reduce the probability that some edge is included in all matchings. We will show a general technique to construct an ordered set of k matchings where k is an easily adjustable parameter. For $k = 2$, we show that the probability of an edge appearing in both M_1 and M_2 is at most $1 - 2/e \approx 0.26$.

For the algorithms presented, we first solve an LP on the input graph. We then round the LP solution vector to a sparse integral vector and use this vector to construct a randomly ordered set of matchings which will guide our algorithm during the online phase. We begin Section 3 with a simple warm-up algorithm which uses a set of two matchings as a guide to achieve a 0.688 competitive ratio, improving the best known result for this problem. We follow it up with a slight variation that improves the ratio to 0.7 and a more complex 0.705-competitive algorithm which relies on a convex combination of a 3-matching algorithm and a separate *pseudo-matching* algorithm.

2.4 Overview of non-integral arrival rates with stochastic rewards contributions

This algorithm is presented in Section 5. We believe the known I.I.D. model with stochastic rewards is an interesting new direction motivated by the work of [20] and [21] in the adversarial model. We introduce a new, more general LP specifically for this setting and show that a simple algorithm using the LP solution directly can achieve a competitive ratio of $1 - 1/e$. In [21], it is shown that no randomized algorithm can achieve a ratio better than 0.62 $< 1 - 1/e$ in the adversarial model. Hence, achieving a $1 - 1/e$ for the i.i.d. model shows that this lower bound does not extend to this model.

In Section 6, we extend this simple algorithm[2] to the b-matching generalization of this problem where each offline vertex u can match with up to b arriving vertices. We show that our algorithm achieves a competitive ratio of at least $1 - b^{-1/2+\epsilon} - O(e^{-b^{2\epsilon}/3})$ for any given $\epsilon > 0$. Note that this result makes progress on Open Question 14 in the online matching and ad allocation survey [19] which asks about stochastic rewards in non-adversarial models.

2.5 Summary of our contributions

▶ **Theorem 1.** *For vertex-weighted online stochastic matching with integral arrival rates, online algorithm* VW *achieves a competitive ratio of at least* 0.7299.

▶ **Theorem 2.** *For edge-weighted online stochastic matching with integral arrival rates, there exists an algorithm which achieves a competitive ratio of at least* 0.7 *and algorithm* EW$[q]$ *with* $q = 0.149251$ *achieves a competitive ratio of at least* 0.70546.

▶ **Theorem 3.** *For edge-weighted online stochastic matching with arbitrary arrival rates and stochastic rewards, online algorithm* SM (4) *achieves a competitive ratio of* $1 - 1/e$.

▶ **Theorem 4.** *For edge-weighted online stochastic b-matching with arbitrary arrival rates and stochastic rewards, online algorithm* SM$_\mathsf{b}$ (5) *achieves a competitive ratio of at least* $1 - b^{-1/2+\epsilon} - O(e^{-b^{2\epsilon}/3})$ *for any given* $\epsilon > 0$.

[2] Recently, we have come to know that the result in Section 6 can be obtained as a special case of [3]. Our approach gives an alternative, and a simpler algorithm for this special case.

Table 1 Summary of Contributions.

Problem	Previous Work	This Paper
Edge-Weighted (Section 3)	0.667 [11]	0.705
Vertex-Weighted (Section 4)	0.725 [12]	0.7299
Unweighted	0.7293 [12]	0.7299
Non-integral Stochastic Rewards (Section 5)	N/A	$1 - e^{-1}$
b-matching, Stochastic Rewards (Section 6)	N/A	$1 - b^{-1/2+\epsilon} - O(e^{-b^{2\epsilon}/3})$

2.6 LP rounding technique

For the algorithms presented, we will first solve the benchmark LP in sub-section 2.1 for the input instance to get a fractional solution vector $\mathbf{f}$. We then round $\mathbf{f}$ to an integral solution $\mathbf{F}$ using a two step process we call $\mathsf{DR}[\mathbf{f}, k]$. The first step is to multiply $\mathbf{f}$ by k. The second step is to apply the dependent rounding techniques of Gandhi, Khuller, Parthasarathy and Srinivasan [10] to this new vector. In this paper, we will always choose k to be 2 or 3. This will help us handle the fact that a vertex in V may appear more than once, but probably not more than two or three times.

While dependent rounding is typically applied to values between 0 and 1, the useful properties extend naturally to our case in which kf_e may be greater than 1 for some edge e. To understand this process, it is easiest to imagine splitting each kf_e into two edges with the integer value $f'_e = \lfloor kf_e \rfloor$ and fractional value $f''_e = kf_e - \lfloor kf_e \rfloor$. The former will remain unchanged by the dependent rounding since it is already an integer while the latter will be rounded to 1 with probability f''_e and 0 otherwise. Our final value F_e would be the sum of those two rounded values. The two properties of dependent rounding we will use are:

1. **Marginal distribution:** For every edge e, let $p_e = kf_e - \lfloor kf_e \rfloor$. Then, $\Pr[F_e = \lceil kf_e \rceil] = p_e$ and $\Pr[F_e = \lfloor kf_e \rfloor] = 1 - p_e$.
2. **Degree-preservation:** For any vertex $w \in U \cup V$, let its fractional degree kf_w be $\sum_{e \in \partial(w)} kf_e$ and integral degree be the random variable $F_w = \sum_{e \in \partial(w)} F_e$. Then $F_w \in \{\lfloor kf_w \rfloor, \lceil kf_w \rceil\}$.

3 Edge-weighted matching with integral arrival rates

3.1 A simple 0.688-competitive algorithm

As a warm-up, we will describe a simple algorithm which achieves a competitive ratio of 0.688 and introduces key ideas in our approach. We begin by solving the LP in sub-section 2.1 to get a fractional solution vector $\mathbf{f}$ and applying $\mathsf{DR}[\mathbf{f}, 2]$ as described in Subsection 2.6 to get an integral vector $\mathbf{F}$. We construct a bipartite graph $G_{\mathbf{F}}$ with F_e copies of each edge e. Note that $G_{\mathbf{F}}$ will have max degree 2 since for all $w \in U \cup V$, $F_w \leq \lceil 2f_w \rceil \leq 2$ and therefore we can decompose it into two matchings using *Hall's Theorem*. Finally, we randomly permute the two matchings into an ordered pair of matchings, $[M_1, M_2]$. These matchings serve as a guide for the online phase of the algorithm, similar to [11].

The entire warm-up algorithm for the edge-weighted model, denoted by EW_0, is summarized in Algorithm 1.

```
Algorithm 1: [EW_0]
1 Construct and solve the benchmark LP in sub-section 2.1 for the input instance.
2 Let $\mathbf{f}$ be an optimal fraction solution vector. Call $\mathsf{DR}[\mathbf{f}, 2]$ to get an integral vector $\mathbf{F}$.
3 Create the graph $G_{\mathbf{F}}$ with $F_e$ copies of each edge $e \in E$ and decompose it into two
    matchings.
4 Randomly permute the matchings to get a random ordered pair of matchings, say
    $[M_1, M_2]$.
5 When a vertex $v$ arrives for the first time, try to assign $v$ to some $u_1$ if $(u_1, v) \in M_1$;
    when $v$ arrives for the second time, try to assign $v$ to some $u_2$ if $(u_2, v) \in M_2$.
6 When a vertex $v$ arrives for the third time or more, do nothing in that step.
```

3.1.1 Analysis of algorithm EW_0

We will show that EW_0 (Algorithm 1) achieves a competitive ratio of 0.688. Let $[M_1, M_2]$ be our randomly ordered pair of matchings. Note that there might exist some edge e which appears in both matchings if $f_e > 1/2$. Therefore, we consider three types of edges. We say an edge e is of type ψ_1, denoted by $e \in \psi_1$, iff e appears *only* in M_1. Similarly $e \in \psi_2$, iff e appears *only* in M_2 and $e \in \psi_b$, iff e appears in *both* M_1 and M_2.

Let P_1, P_2, P_b be the probabilities of getting matched for $e \in \psi_1$, $e \in \psi_2$, and $e \in \psi_b$ respectively. According to the result in Haeupler *et al.* [11], the respective values are shown as follows.

► **Lemma 5.** *(Proof details in Section 3 of [11]) Given M_1 and M_2, in the worst case* (1) $P_1 = 0.5808$*;* (2) $P_2 = 0.14849$ *and* (3) $P_b = 0.632$.

Proof. (Analysis for EW_0) Consider following two cases.

Case 1: $0 \leq f_e \leq 1/2$: By the marginal distribution property of dependent rounding, there can be at most one copy of e in $G_{\mathbf{F}}$ and the probability of including e in $G_{\mathbf{F}}$ is $2f_e$. Since an edge in $G_{\mathbf{F}}$ can appear in either M_1 or M_2 with equal probability 1/2, we have $\Pr[e \in \psi_1] = \Pr[e \in \psi_2] = f_e$. Thus, the ratio is $(f_e P_1 + f_e P_2)/f_e = P_1 + P_2 = 0.729$.

Case 2: $1/2 \leq f_e \leq 1 - 1/e$: Similarly, by marginal distribution, $\Pr[e \in \psi_b] = \Pr[F_e = \lceil 2f_e \rceil] = 2f_e - \lfloor 2f_e \rfloor = 2f_e - 1$. It follows that $\Pr[e \in \psi_1] = \Pr[e \in \psi_2] = (1/2)(1 - (2f_e - 1)) = 1 - f_e$. Thus, the ratio is $((1 - f_e)(P_1 + P_2) + (2f_e - 1)P_b)/f_e \geq 0.688$, where the WS is for an edge e with $f_e = 1 - 1/e$. ◄

3.2 A 0.7-competitive algorithm

In this section, we describe an improvement upon the previous warm-up algorithm to get a competitive ratio of 0.7. We start by making an observation about the performance of the warm-up algorithm. After solving the LP, let edges with $f_e > 1/2$ be called *large* and edges with $f_e \leq 1/2$ be called *small*. Let L and S, be the sets of large and small edges, respectively. Notice that in the previous analysis, small edges achieved a much higher competitive ratio of 0.729 versus 0.688 for large edges. This is primarily due to the fact that we may get two copies of a large edge in $G_{\mathbf{F}}$. In this case, the copy in M_1 has a better chance of being matched, since there is no edge which can block it, but the copy that is in M_2 has no chance of being matched.

To correct this imbalance, we make an additional modification to the f_e values *before* applying $\mathsf{DR}[\mathbf{f}, k]$. The rest of the algorithm is exactly the same. Let η be a parameter to be optimized later. For all large edges $\ell \in L$ such that $f_\ell > 1/2$, we set $f_\ell = f_\ell + \eta$. For all small edges $s \in S$ which are adjacent to some large edge, let $\ell \in L$ be the largest edge adjacent to s such that $f_\ell > 1/2$. Note that it is possible for e to have two large neighbors, but we only care about the largest one. We set $f_s = f_s\left(\frac{1-(f_\ell+\eta)}{1-f_\ell}\right)$.

In other words, we increase the values of large edges while ensuring that for all $w \in U \cup V$, $f_w \leq 1$ by reducing the values of neighboring small edges proportional to their original values. Note that it is not possible for two large edges to be adjacent since they must both have $f_e > 1/2$. For all other small edges which are not adjacent to any large edges, we leave their values unchanged. We then apply $\mathsf{DR}[\mathbf{f}, 2]$ to this new vector, multiplying by 2 and applying dependent rounding as before.

3.2.1 Analysis

We can now prove Theorem 2.

Proof. As in the warm-up analysis, we'll consider large and small edges separately

- $0 \leq f_s \leq \frac{1}{2}$: Here we have two cases
  - Case 1: s is not adjacent to any large edges. In this case, the analysis is the same as the warm-up algorithm and we still get a 0.729 competitive ratio for these edges.
  - Case 2: s is adjacent to some large edge ℓ. For this case, let f_ℓ be the value of the largest neighboring edge in the original LP solution. Then s achieves a ratio of

$$f_s\left(\frac{1-(f_\ell+\eta)}{1-f_\ell}\right)(0.1484+0.5803)/f_s = \left(\frac{1-(f_\ell+\eta)}{1-f_\ell}\right)(0.1484+0.5803)$$

  Note that for $f_\ell \in [0,1)$ this is a decreasing function with respect to f_ℓ. So the worst case is $f_\ell = 1-1/e$ and we have a ratio of

$$\left(\frac{1-(1-1/e+\eta)}{1-(1-1/e)}\right)(0.1484+0.5803) = \left(\frac{1/e-\eta}{1/e}\right)(0.1484+0.5803)$$

- $\frac{1}{2} < f_\ell \leq 1-\frac{1}{e}$: Here, the ratio is $((1-(f_\ell+\eta))(P_1+P_2)+(2(f_\ell+\eta)-1)P_b)/f_\ell$, where the WS is for an edge e with $f_\ell = 1-1/e$ since this is a decreasing function with respect to f_ℓ.

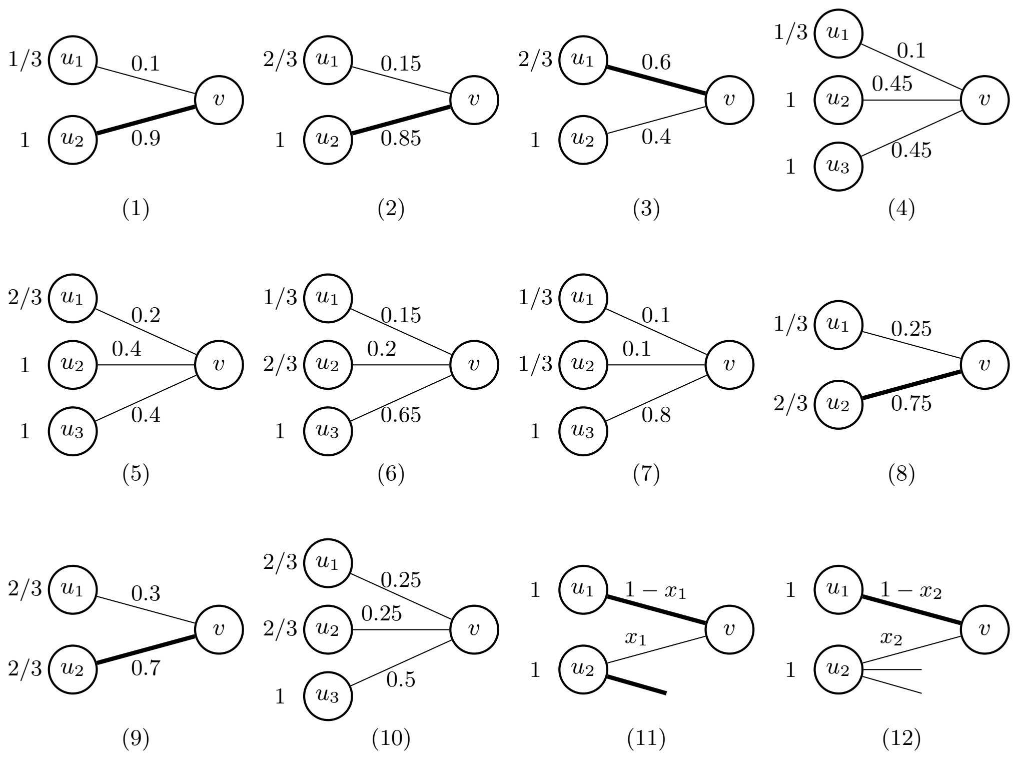
Choosing the optimal value of $\eta = 0.0142$, yields an overall competitive ratio of 0.7 for this new algorithm. ◀

3.3 A 0.705-competitive algorithm

The details of algorithm and the proof of Theorem 2 can be found in the full version of this paper.

4 Vertex-weighted stochastic I.I.D. matching with integral arrival rates

In this section, we will consider vertex-weighted online stochastic matching on a bipartite graph G under known $I.I.D.$ model with integral arrival rates. We will present an algorithm in which each u has a competitive ratio of at least 0.72998. Recall that after invoking $\mathsf{DR}[\mathbf{f}, 3]$,



1/3 u1 0.1 v 1 u2 0.9 (1)
2/3 u1 0.15 v 1 u2 0.85 (2)
2/3 u1 0.6 v 1 u2 0.4 (3)
1/3 u1 0.1 1 u2 0.45 v 1 u3 0.45 (4)
2/3 u1 0.2 1 u2 0.4 v 1 u3 0.4 (5)
1/3 u1 0.15 2/3 u2 0.2 v 1 u3 0.65 (6)
1/3 u1 0.1 1/3 u2 0.1 v 1 u3 0.8 (7)
1/3 u1 0.25 v 2/3 u2 0.75 (8)
2/3 u1 0.3 v 2/3 u2 0.7 (9)
2/3 u1 0.25 2/3 u2 0.25 v 1 u3 0.5 (10)
1 u1 1 − x1 x1 v 1 u2 (11)
1 u1 1 − x2 x2 v 1 u2 (12)

Figure 2 Illustration for second modification to $\mathbf{H}$. The value assigned to each edge represents the value after the second modification. Here, $x_1 = 0.2744$ and $x_2 = 0.15877$.

we can obtain a (*random*) integral vector $\mathbf{F}$ with $F_e \in \{0, 1, 2\}$. Define $\mathbf{H} = \mathbf{F}/3$ and let $G_{\mathbf{H}}$ be the graph induced by $\mathbf{H}$ and each edge takes the value $H_e \in \{0, 1/3, 2/3\}$.

In this section, we focus on the sparse graph $G_{\mathbf{H}}$. The main steps of the algorithm are:

1. Solve the vertex-weighted benchmark LP in sub-section 2.1. Let $\mathbf{f}$ be an optimal solution vector.
2. Invoke DR[$\mathbf{f}$, 3] to obtain an integral vector $\mathbf{F}$ and a fractional vector $\mathbf{H}$ with $\mathbf{H} = \mathbf{F}/3$.
3. Apply a series of modifications to $\mathbf{H}$ and transform it to another solution $\mathbf{H}'$. See sub-section 4.1.
4. Run the randomized list algorithm (RLA) [12] induced by $\mathbf{H}'$ on the graph $G_{\mathbf{H}}$. See the details in full version of this paper.

The WS for vertex-weighted case in [12] is shown in Figure 3, which arrives at node u with a competitive ratio of 0.725. From their analysis, we find node u_1 has a competitive ratio of at least 0.736. Hence, we *boost* the performance of u at the cost of u_1. In other words, we increase the value of $H_{(u,v1)}$ and decrease the value $H_{(u_1,v_1)}$. Case (10) and (11) in Figure 2 illustrates this. After this modification, the new WS for vertex-weighted is now the C_1 cycle shown in Figure 1. In fact, this is the WS for the unweighted case in [12]. However, Lemma 6 and the cycle breaking algorithm, implies that C_1 cycle can be avoided with probability at least $3/e - 1$. This helps us improve the ratio even for the unweighted case in [12].

▶ **Lemma 6.** *For any given $u \in U$, u appears in a C_1 cycle after* DR[$\boldsymbol{f}$, 3] *with probability at most $2 - 3/e$.*

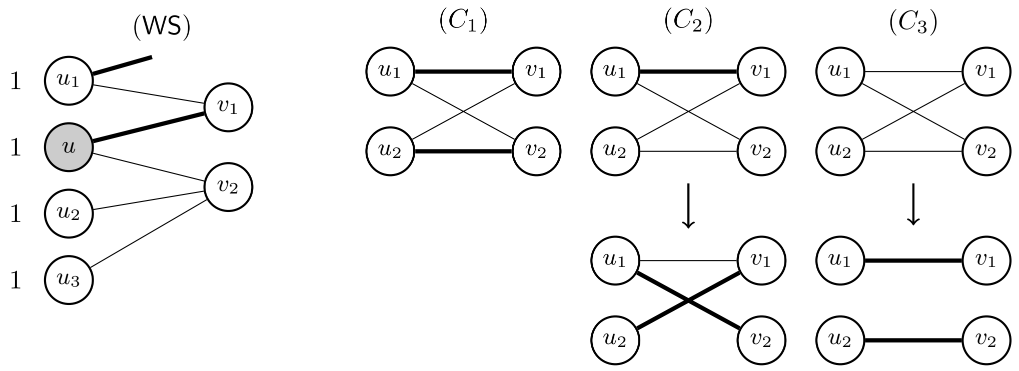


Figure 3 Left: The WS for Jaillet and Lu [12] for their vertex-weighted case. Right: The three possible types of cycles of length 4 after applying DR$[\mathbf{f}, 3]$. Thin edges have $H_e = 1/3$ and thick edges have $H_e = 2/3$.

Proof. Consider the graph $G_{\mathbf{H}}$ obtained after DR$[\mathbf{f}, 3]$. Notice that for some vertex u to appear in a C_1 cycle, it must have a neighboring edge with $H_e = 2/3$. Now we try to bound the probability of this event. It is easy to see that for some $e \in \partial(u)$ with $f_e \leq 1/3$, $F_e \leq 1$ after DR$[\mathbf{f}, 3]$, and hence $H_e = F_e/3 \leq 1/3$. Thus only those edges $e \in \partial(u)$ with $f_e > 1/3$ will possibly be rounded to $H_e = 2/3$. Note that, there can be at most two such edges in $\partial(u)$, since $\sum_{e \in \partial(u)} f_e \leq 1$. Hence, we have the following two cases.

Case 1: $\partial(u)$ contains only one edge e with $f_e > 1/3$. Let $q_1 = \Pr[H_e = 1/3]$ and $q_2 = \Pr[H_e = 2/3]$ after DR$[\mathbf{f}, 3]$. By DR$[\mathbf{f}, 3]$, we know that $\mathbb{E}[H_e] = \mathbb{E}[F_e]/3 = q_2(2/3) + q_1(1/3) = f_e$.
Notice that $q_1 + q_2 = 1$ and hence $q_2 = 3f_e - 1$. Since this is an increasing function of f_e and $f_e \leq 1 - 1/e$ from LP constraint 2.4, we have $q_2 \leq 3(1 - 1/e) - 1 = 2 - 3/e$.

Case 2: $\partial(u)$ contains two edges e_1 and e_2 with $f_{e_1} > 1/3$ and $f_{e_2} > 1/3$. Let q_2 be the probability that after DR$[\mathbf{f}, 3]$, either $H_{e_1} = 2/3$ or $H_{e_2} = 2/3$. Note that, these two events are mutually exclusive since $H_u \leq 1$. Using the analysis from case 1, it follows that $q_2 = (3f_{e_1} - 1) + (3f_{e_2} - 1) = 3(f_{e_1} + f_{e_2}) - 2$.
From LP constraint 2.5, we know that $f_{e_1} + f_{e_2} \leq 1 - 1/e^2$, and hence $q_2 \leq 3(1 - 1/e^2) - 2 < 2 - 3/e$. ◀

4.1 Two kinds of Modifications to H

The first modification is to break the cycles deterministically.

There are three possible cycles of length 4 in the graph $G_{\mathbf{H}}$, denoted C_1, C_2, and C_3. In [12], they give an efficient way to break C_2 and C_3, as shown in Figure 3. Cycle C_1 cannot be modified further and hence, is the bottleneck for their unweighted case. Notice that, while breaking the cycles of C_2 and C_3, new cycles of C_1 can be created in the graph. Since our randomized construction of solution $\mathbf{H}$ gives us control on the probability of cycles C_1 occurring, we would like to break C_2 and C_3 in a controlled way, so as to not create any new C_1 cycles. This procedure is summarized in Algorithm 2. The proof of Lemma 7 can be found in the full version of this paper.

▶ **Lemma 7.** *After applying Algorithm 2 to $G_{\mathbf{H}}$, we have (1) the value H_w is preserved for each $w \in U \cup V$; (2) no cycle of type C_2 or C_3 exists; (3) no new cycle of type C_1 is added.*

Algorithm 2: [Cycle breaking algorithm] Offline Phase

```
1 While there is some cycle of type $C_2$ or $C_3$, Do:
2 Break all cycles of type $C_2$.
3 Break one cycle of type $C_3$ and return to the first step.
```

The second modification is to decrease the rates of lists associated with those nodes u with $H_u = 1/3$ or $H_u = 2/3$ and increase the rates of lists associated with nodes u with $H_u = 1$. All details can be found in the full version. Let $\mathbf{H}'$ be the solution vector obtained by applying two kinds of modifications to $\mathbf{H}$. The algorithm for the vertex-weighted case, denoted by VW, is summarized below. The detailed analysis can be found in the full version of this paper.

Algorithm 3: VW [Vertex Weighted]

```
1 Construct and solve the LP in sub-section 2.1 for the input instance.
2 Invoke DR[$\mathbf{f}$, 3] to output $\mathbf{F}$ and $\mathbf{H}$. Apply the two kinds of modifications to morph $\mathbf{H}$
   to $\mathbf{H}'$.
3 Run RLA[$\mathbf{H}'$] on the graph $G_{\mathbf{H}}$.
```

5 Non-integral arrival rates with stochastic rewards

The setting here is strictly generalized over the previous sections in the following ways. Firstly, it allows an arbitrary arrival rate (say r_v) which can be fractional for each stochastic vertex v. Notice that, $\sum_v r_v = n$ where n is the total number of rounds.

Secondly, each $e = (v, u) \in E$ is associated with a value p_e, which indicates the probability that edge $e = (u, v)$ is present when we assign v to u. We assume this process is independent of the stochastic arrival of each v. We will show that the simple non-adaptive algorithm introduced in [11] can be extended to this general case. This achieves a competitive ratio of $(1 - \frac{1}{e})$. Note that Manshadi *et al.* [18] show that no non-adaptive algorithm can possibly achieve a ratio better than $(1 - 1/e)$ for the non-integral arrival rates, even for the case of all $p_e = 1$. Thus, our algorithm is an optimal non-adaptive algorithm for this model.

$$\max \quad \sum_{e \in E} w_e f_e p_e : \tag{5.1}$$

$$\text{s.t.} \quad \sum_{e \in \partial(u)} f_e p_e \le 1, \forall u \in U \tag{5.2}$$

$$\sum_{e \in \partial(v)} f_e \le r_v, \forall v \in V \tag{5.3}$$

We use a similar LP as [12] for the case of non-integral arrival rates. For each $e \in E$, let f_e be the probability that e gets matched in the offline optimal algorithm.

Our algorithm is summarized in Algorithm 4. Notice that the last constraint ensures that step 2 in the algorithm is valid. Let us now prove theorem 3.

Proof. Let $B(u, t)$ be the event that u is safe at beginning of round t and $A(u, t)$ to be the event that vertex u is matched during the round t conditioned on $B(u, t)$. From

```
Algorithm 4: SM
1 Construct and solve LP (5.1). WLOG assume $\{f_e | e \in E\}$ is an optimal solution.
2 When a vertex $v$ arrives, assign $v$ to each of its neighbor $u$ with a probability $\frac{f_{(u,v)}}{r_v}$.
```

the algorithm, we know $\Pr[A(u,t)] \le \sum_{v \sim u} \frac{r_v}{n} \frac{f_{u,v}}{r_v} p_e \le \frac{1}{n}$, which follows by $Pr[B(u,t)] = \Pr\left[\bigwedge_{i=1}^{t-1}(\neg A(u,i))\right] \ge \left(1-\frac{1}{n}\right)^{t-1}$.

Consider an edge $e = (u,v)$ in the graph. Notice that the probability that e gets matched in SM should be

$$\begin{aligned}\Pr[e \text{ is matched}] &= \sum_{t=1}^{n} \Pr[\text{v arrives at } t \text{ and } B(u,t)\ ] \cdot \frac{f_e p_e}{r_v} \\ &\ge \sum_{t=1}^{n} \left(1-\frac{1}{n}\right)^{t-1} \frac{r_v}{n} \frac{f_e p_e}{r_v} \ge \left(1-\frac{1}{e}\right) f_e p_e\,.\end{aligned}$$ ◀

6 Extension to b-matching with stochastic rewards

In this section, we further generalize the model in Section 5 to the case where each u in the offline set U has a uniform integral capacity b (i.e., each vertex u can be matched at most b times). Otherwise, we retain the same setting as Section 5; we allow non-integral arrival rates and stochastic rewards. We will generalize the simple algorithm used in the previous setting (i.e., Section 5) to this new setting. Consider the following updated LP:

$$\max \quad \sum_{e \in E} w_e f_e p_e : \tag{6.1}$$

$$\text{s.t.} \quad \sum_{e \in \partial(u)} f_e p_e \le b, \forall u \in U \tag{6.2}$$

$$\sum_{e \in \partial(v)} f_e \le r_v, \forall v \in V \tag{6.3}$$

We modify Algorithm 4 for the b-matching problem as shown in Algorithm 5. Let us now prove Theorem 4.

```
Algorithm 5: SM_b
1 Construct and solve LP (6.1). WLOG assume $\{f_e | e \in E\}$ is an optimal solution.
2 When a vertex $v$ arrives, assign $v$ to each of its neighbor $u$ with a probability $\frac{f_{(u,v)}}{r_v}$.
```

Proof. The proof is similar to that of Theorem 3. Let A_t be the number of times u has been matched at the beginning of round t.

Let $B(u,t)$ be the event that u is safe at the beginning of round t, which is defined as $A_t \le b-1$. For any given edge e, let X_e be the number of times that e gets matched over the n rounds. Thus we have

$$\mathbb{E}[X_e] = \sum_{t=1}^{n} \Pr[B(u,t)] \frac{r_v}{n} \frac{f_e}{r_v} p_e = \frac{f_e p_e}{n} \sum_{t=1}^{n} \Pr[A_t \le b-1]\,.$$

Now we upper bound the value of $\Pr[A_t \geq b]$. For each $1 \leq i \leq t$, let Z_i be the indicator random variable for u to be matched during round i. Thus $A_{t+1} = \sum_{i=1}^{t} Z_i$. Notice that for each i, we have

$$\mathbb{E}[Z_i] \leq \sum_{v \sim u} \frac{r_v}{n} \frac{f_{(u,v)}}{r_v} p_{(u,v)} \leq \frac{b}{n}.$$

It follows that for any $t \leq n(1-\tau)$ with $0 < \tau < 1$, we have $\mathbb{E}[A_{t+1}] \leq (1-\tau)b$. By applying Chernoff-Hoeffding bounds, we get $\Pr[A_{t+1} \geq b] \leq e^{-b\tau^2/3}$. Therefore

$$\begin{aligned}\mathbb{E}[X_e] &= \frac{f_e p_e}{n} \sum_{t=1}^{n} \Pr[A_t \leq b-1] \\ &\geq \frac{f_e p_e}{n} \sum_{t=1}^{n(1-\tau)} (1 - e^{-b\tau^2/3}) = f_e p_e (1-\tau)(1 - e^{-b\tau^2/3})\end{aligned}$$

For any given $\epsilon > 0$, choose $\tau = b^{-1/2+\epsilon}$ to get a competitive ratio of $1 - b^{-1/2+\epsilon} - O(e^{-b^{2\epsilon}/3})$. ◀

7 Conclusion and Future Directions

In this paper, we gave improved algorithms for the Edge-Weighted and Vertex-Weighted models. Previously, there was a gap between the best unweighted algorithm with a ratio of $1 - 2e^{-2}$ due to [12] and the negative result of $1 - e^{-2}$ due to [18]. We took a step towards closing that gap by showing that an algorithm can achieve $0.7299 > 1 - 2e^{-2}$ for both the unweighted and vertex-weighted variants with integral arrival rates. In doing so, we made progess on Open Questions 3 and 4 in the online matching and ad allocation survey [19]. This was possible because our approach of rounding to a simpler fractional solution allowed us to employ a stricter LP. For the edge-weighted variant, we showed that one can significantly improve the power of two choices approach by generating two matchings from the same LP solution. For the variant with edge weights, non-integral arrival rates, and stochastic rewards, we presented a $(1-1/e)$-competitive algorithm. This showed that the $0.62 < 1 - 1/e$ bound given in [21] for the adversarial model with stochastic rewards does not extend to the known I.I.D. model. Furthermore, we considered the online edge-weighted b-matching problem with stochastic rewards under the known IID setting. We gave a very simple non-adaptive algorithm which achieves a ratio of $1 - b^{-1/2+\epsilon} - O(e^{-b^{2\epsilon}/3})$ for any given $\epsilon > 0$.

A natural next step in the edge-weighted setting is to use an *adaptive* strategy. For the vertex-weighted problem, one can easily see that the stricter LP we use still has a gap. In addition, we only utilize fractional solutions $\{0, 1/3, 2/3\}$. However, dependent rounding gives solutions in $\{0, 1/k, 2/k, \ldots, \lceil k(1-1/e) \rceil / k\}$; allowing for random lists of length greater than three. Stricter LPs and longer lists could both yield improved results. In the stochastic rewards model with non-integral arrival rates, an open question is to either improve upon the $\left(1 - \frac{1}{e}\right)$ ratio or consider a simpler model with integral arrival rates and improve the ratio for this restricted model. Lastly, there is a gap between our result for b-matching with stochastic rewards and the results of [7] and [2] for similar problems with deterministic rewards. It would be nice to see a result for this problem that is $1 - O(k^{-1/2})$.

Acknowledgements. The authors would like to thank Aranyak Mehta and the anonymous reviewers for their valuable comments, which have significantly helped improve the presentation of this paper.

References

1 Gagan Aggarwal, Gagan Goel, Chinmay Karande, and Aranyak Mehta. Online vertex-weighted bipartite matching and single-bid budgeted allocations. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 1253–1264. SIAM, 2011.

2 Saeed Alaei, MohammadTaghi Hajiaghayi, and Vahid Liaghat. Online prophet-inequality matching with applications to ad allocation. In *Proceedings of the 13th ACM Conference on Electronic Commerce*, pages 18–35. ACM, 2012.

3 Saeed Alaei, MohammadTaghi Hajiaghayi, and Vahid Liaghat. The online stochastic generalized assignment problem. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques: 16th International Workshop, APPROX, and 17th International Workshop, RANDOM*, pages 11–25. Springer Berlin Heidelberg, 2013.

4 Bahman Bahmani and Michael Kapralov. Improved bounds for online stochastic matching. In *European Symposium on Algorithms (ESA)*, pages 170–181. Springer, 2010.

5 Nikhil R Devanur and Thomas P Hayes. The adwords problem: online keyword matching with budgeted bidders under random permutations. In *Proceedings of the 10th ACM conference on Electronic commerce*, pages 71–78. ACM, 2009.

6 Nikhil R. Devanur, Kamal Jain, Balasubramanian Sivan, and Christopher A. Wilkens. Near optimal online algorithms and fast approximation algorithms for resource allocation problems. In *Proceedings of the 12th ACM Conference on Electronic Commerce*, pages 29–38. ACM, 2011.

7 Nikhil R Devanur, Balasubramanian Sivan, and Yossi Azar. Asymptotically optimal algorithm for stochastic adwords. In *Proceedings of the 13th ACM Conference on Electronic Commerce*, pages 388–404. ACM, 2012.

8 Jon Feldman, Nitish Korula, Vahab Mirrokni, S Muthukrishnan, and Martin Pál. Online ad assignment with free disposal. In *Internet and network economics*, pages 374–385. Springer, 2009.

9 Jon Feldman, Aranyak Mehta, Vahab Mirrokni, and S Muthukrishnan. Online stochastic matching: Beating 1-1/e. In *Foundations of Computer Science (FOCS)*, pages 117–126. IEEE, 2009.

10 Rajiv Gandhi, Samir Khuller, Srinivasan Parthasarathy, and Aravind Srinivasan. Dependent rounding and its applications to approximation algorithms. *Journal of the ACM (JACM)*, 53(3):324–360, 2006.

11 Bernhard Haeupler, Vahab S. Mirrokni, and Morteza Zadimoghaddam. Online stochastic weighted matching: Improved approximation algorithms. In *Internet and Network Economics*, volume 7090 of *Lecture Notes in Computer Science*, pages 170–181. Springer Berlin Heidelberg, 2011.

12 Patrick Jaillet and Xin Lu. Online stochastic matching: New algorithms with better bounds. *Mathematics of Operations Research*, 39(3):624–646, 2013.

13 Bala Kalyanasundaram and Kirk R Pruhs. An optimal deterministic algorithm for online b-matching. *Theoretical Computer Science*, 233(1):319–325, 2000.

14 Richard M Karp, Umesh V Vazirani, and Vijay V Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 352–358. ACM, 1990.

15 Thomas Kesselheim, Klaus Radke, Andreas Tönnis, and Berthold Vöcking. An optimal online algorithm for weighted bipartite matching and extensions to combinatorial auctions. In *European Symposium on Algorithms (ESA)*, pages 589–600. Springer, 2013.

16 Nitish Korula and Martin Pál. Algorithms for secretary problems on graphs and hypergraphs. In *Automata, Languages and Programming*, pages 508–520. Springer, 2009.

17 Mohammad Mahdian and Qiqi Yan. Online bipartite matching with random arrivals: an approach based on strongly factor-revealing LPs. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 597–606. ACM, 2011.

18 Vahideh H Manshadi, Shayan Oveis Gharan, and Amin Saberi. Online stochastic matching: Online actions based on offline statistics. *Mathematics of Operations Research*, 37(4):559–573, 2012.

19 Aranyak Mehta. Online matching and ad allocation. *Foundations and Trends in Theoretical Computer Science*, 8(4):265–368, 2012.

20 Aranyak Mehta and Debmalya Panigrahi. Online matching with stochastic rewards. In *Foundations of Computer Science (FOCS)*, pages 728–737. IEEE, 2012.

21 Aranyak Mehta, Bo Waggoner, and Morteza Zadimoghaddam. Online stochastic matching with unequal probabilities. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2015.

Solving k-SUM Using Few Linear Queries

Jean Cardinal[*1], **John Iacono**[†2], **and Aurélien Ooms**[‡3]

1 **Université libre de Bruxelles (ULB), Brussels, Belgium**
jcardin@ulb.ac.be
2 **New York University, New York, USA**
esa2016submission@johniacono.com
3 **Université libre de Bruxelles (ULB), Brussels, Belgium**
aureooms@ulb.ac.be

Abstract

The k-SUM problem is given n input real numbers to determine whether any k of them sum to zero. The problem is of tremendous importance in the emerging field of complexity theory within P, and it is in particular open whether it admits an algorithm of complexity $O(n^c)$ with $c < \lceil \frac{k}{2} \rceil$. Inspired by an algorithm due to Meiser (1993), we show that there exist linear decision trees and algebraic computation trees of depth $O(n^3 \log^2 n)$ solving k-SUM. Furthermore, we show that there exists a randomized algorithm that runs in $\tilde{O}(n^{\lceil \frac{k}{2} \rceil + 8})$ time, and performs $O(n^3 \log^2 n)$ linear queries on the input. Thus, we show that it is possible to have an algorithm with a runtime almost identical (up to the +8) to the best known algorithm but for the first time also with the number of queries on the input a polynomial that is independent of k. The $O(n^3 \log^2 n)$ bound on the number of linear queries is also a tighter bound than any known algorithm solving k-SUM, even allowing unlimited total time outside of the queries. By simultaneously achieving few queries to the input without significantly sacrificing runtime vis-à-vis known algorithms, we deepen the understanding of this canonical problem which is a cornerstone of complexity-within-P.

We also consider a range of tradeoffs between the number of terms involved in the queries and the depth of the decision tree. In particular, we prove that there exist $o(n)$-linear decision trees of depth $\tilde{O}(n^3)$ for the k-SUM problem.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases k-SUM problem, linear decision trees, point location, ε-nets

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.25

1 Introduction

The k-SUM problem is defined as follows: given a collection of n real numbers decide whether any k of them sum to zero, where k is a constant. It is a fixed-parameter version of the subset-sum problem, a standard NP-complete problem. The k-SUM problem, and in particular the special case of 3SUM, has proved to be a cornerstone of the fine-grained complexity program aiming at the construction of a complexity theory for problems in P. In particular, there are deep connections between the complexity of k-SUM, the Strong

[*] Supported by the "Action de Recherche Concertée" (ARC) COPHYMA, convention number 4.110.H.000023.
[†] Research partially completed while on on sabbatical at the Algorithms Research Group of the Département d'Informatique at the Université Libre de Bruxelles with support from a Fulbright Research Fellowship, the Fonds de la Recherche Scientifique – FNRS, and NSF grants CNS-1229185, CCF-1319648, and CCF-1533564.
[‡] Supported by the Fund for Research Training in Industry and Agriculture (FRIA).

Exponential Time Hypothesis [30, 12], and the complexity of many other major problems in P [20, 7, 28, 29, 5, 2, 23, 25, 1, 3, 13].

It has been long known that the k-SUM problem can be solved in time $O(n^{\frac{k}{2}} \log n)$ for even k, and $O(n^{\frac{k+1}{2}})$ for odd k. Erickson [17] proved a near-matching lower bound in the k-linear decision tree model. In this model, the complexity is measured by the depth of a decision tree, every node of which corresponds to a query of the form $q_{i_1} + q_{i_2} + \cdots + q_{i_k} \leq^? 0$, where $q_1, q_2, \ldots, q_n$ are the input numbers. In a recent breakthrough paper, Grønlund and Pettie [23] showed that in the $(2k-2)$-linear decision tree model, where queries test the sign of weighted sums of up to $2k-2$ input numbers, only $O(n^{\frac{k}{2}}\sqrt{\log n})$ queries are required for odd values of k. In particular, there exists a 4-linear decision tree for 3SUM of depth $\tilde{O}(n^{\frac{3}{2}})$ (here the notation $\tilde{O}$ ignores polylogarithmic factors), while every 3-linear decision tree has depth $\Omega(n^2)$ [17]. This indicates that increasing the size of the queries, defined as the maximum number of input numbers involved in a query, can yield significant improvements on the depth of the minimal-height decision tree. Ailon and Chazelle [4] slightly extended the range of query sizes for which a nontrivial lower bound could be established, elaborating on Erickson's technique.

It has been well established that there exist nonuniform polynomial-time algorithms for the subset-sum problem. One of them was described by Meiser [26], and is derived from a data structure for point location in arrangements of hyperplanes using the bottom vertex decomposition. This algorithm can be cast as the construction of a linear decision tree in which the queries have non-constant size.

1.1 Our results

In Section 3, we show the existence of an n-linear decision tree of depth $\tilde{O}(n^3)$ for k-SUM using a careful implementation of Meiser's algorithm [26]. Although the high-level algorithm itself is not new, we refine the implementation and analysis for the k-SUM problem.[1] Meiser presented his algorithm as a general method of point location in m given n-dimensional hyperplanes that yielded a $\tilde{O}(n^4 \log m)$-depth algebraic computation tree; when viewing the k-SUM problem as a point location problem, m is $O(n^k)$ and thus Meiser's algorithm can be viewed as giving a $\tilde{O}(n^4)$-depth algebraic computation tree. We show that while the original algorithm was cast as a nonuniform polynomial-time algorithm, it can be implemented in the linear decision tree model with an $\tilde{O}(n^3)$ upper bound. Moreover, this result implies the same improved upper bound on the depth of algebraic computation trees for the k-SUM problem, as shown in Appendix B.

There are two subtleties to this result. The first is inherent to the chosen complexity model: even if the number of queries to the input is small (in particular, the degree of the polynomial complexity is invariant on k), the time required to *determine which queries should be performed* may be arbitrary. In a naïve analysis, we show it can be trivially bounded by $\tilde{O}(n^{k+2})$. In Section 4 we present an algorithm to choose which decisions to perform whereby the running time can be reduced to $\tilde{O}(n^{\frac{k}{2}+8})$. Hence, we obtain an $\tilde{O}(n^{\frac{k}{2}+8})$ time randomized algorithm in the RAM model expected to perform $\tilde{O}(n^3)$ linear queries on the input[2].

[1] After submitting this manuscript, we learned from a personal communication with Hervé Fournier that a similar analysis appears in his PhD thesis [18] (in French).

[2] Grønlund and Pettie [23] mention the algorithms of Meyer auf der Heyde [27] and Meiser [26], and state "(...) it was known that all k-LDT problems can be solved by n-linear decision trees with depth $O(n^5 \log n)$ [26], or with depth $O(n^4 \log(nK))$ if the coefficients of the linear function are integers with

Table 1 Complexities of our new algorithms for the k-SUM problem. The query size is the maximum number of elements of the input that can be involved in a single linear query. The number of blocks is a parameter that allows us to change the query size (see Section 5). The origin of the constant in the exponent of the time complexity is due to Lemma 9. We conjecture it can be reduced, though substantial changes in the analysis will likely be needed to do so.

	# blocks	query size	# queries	time
Theorem 3	1	n	$\tilde{O}(n^3)$	$\tilde{O}(n^{\lceil \frac{k}{2} \rceil + 8})$
Theorem 11	b	$k\lceil \frac{n}{b} \rceil$	$\tilde{O}(b^{k-4}n^3)$	$\tilde{O}(b^{\lfloor \frac{k}{2} \rfloor - 9}n^{\lceil \frac{k}{2} \rceil + 8})$
Corollary 12	$b = \Theta(\text{polylog}(n))$	$o(n)$	$\tilde{O}(n^3)$	$\tilde{O}(n^{\lceil \frac{k}{2} \rceil + 8})$
Corollary 13	$b = \Theta(n^{\alpha})$	$O(n^{1-\alpha})$	$\tilde{O}(n^{3+(k-4)\alpha})$	$\tilde{O}(n^{(1+\alpha)\frac{k}{2} + 8.5})$

The second issue we address is that the linear queries in the above algorithm may have size n, that is, they may use all the components of the input. The lower bound of Erickson shows that if the queries are of minimal size, the number of queries cannot be a polynomial independent of k such as what we obtain, so non-minimal query size is clearly essential to a drastic reduction in the number of queries needed. This gives rise to the natural question as to what is the relation between query size and number of queries. In particular, one natural question is whether queries of size less than n would still allow the problem to be solved using a number of queries that is a polynomial independent of k. We show that this is possible; in Section 5, we introduce a range of algorithms exhibiting an explicit tradeoff between the number of queries and their size. Using a blocking scheme, we show that we can restrict to $o(n)$-linear decision trees. We also give a range of tradeoffs for $O(n^{1-\alpha})$-linear decision trees. Although the proposed algorithms still involve nonconstant-size queries, this is the first time such tradeoffs are explicitly tackled. Table 1 summarizes our results.

2 Definitions and previous work

2.1 Definitions

We consider the k-SUM problem for $k = O(1)$. In what follows, we use the notation $[n] = \{ 1, 2, \dots, n \}$.

▶ **Problem** (k-SUM). *Given an input vector $q \in \mathbb{R}^n$, decide whether there exists a k-tuple $(i_1, i_2, \dots, i_k) \in [n]^k$ such that $\sum_{j=1}^{k} q_{i_j} = 0$.*

The problem amounts to deciding in n-dimensional space, for each hyperplane H of equation $x_{i_1} + x_{i_2} + \cdots + x_{i_k} = 0$, whether q lies on, above, or below H. Hence this indeed amounts to locating the point q in the arrangement formed by those hyperplanes. We emphasize that the set of hyperplanes depends only on k and n and not on the actual input vector q.

Linear degeneracy testing (k-LDT) is a generalization of k-SUM where we have arbitrary rational coefficients[3] and an independent term in the equations of the hyperplanes.

▶ **Problem** (k-LDT). *Given an input vectors $q \in \mathbb{R}^n$ and $\alpha \in \mathbb{Q}^n$ and constant $c \in \mathbb{Q}$ decide whether there exists a k-tuple $(i_1, i_2, \dots, i_k) \in [n]^k$ such that $c + \sum_{j=1}^{k} \alpha_j q_{i_j} = 0$.*

absolute value at most K [27]. Unfortunately these decision trees are not efficiently constructible. The time required to determine *which* comparisons to make is exponential." We prove that the trees can have depth $\tilde{O}(n^3)$ and that the whole algorithm can run in randomized polynomial-time.

[3] The usual definition of k-LDT allows arbitrary *real* coefficients. However, the algorithm we provide for Lemma 8 needs the vertices of the arrangement of hyperplanes to have rational coordinates.

Our algorithms apply to this more general problem with only minor changes.

The *s-linear decision tree model* is a standard model of computation in which several lower bounds for k-SUM have been proven. In the decision tree model, one may ask well-defined questions to an oracle that are answered “yes” or “no.” For s-linear decision trees, a well-defined question consists of testing the sign of a linear function on at most s numbers $q_{i_1}, \dots, q_{i_s}$ of the input $q_1, \dots, q_n$ and can be written as

$$c + \alpha_1 q_{i_1} + \cdots + \alpha_s q_{i_s} \overset{?}{\leq} 0\,.$$

Each question is defined to cost a single unit. All other operations can be carried out for free but may not examine the input vector q. We refer to n-linear decision trees simply as linear decision trees.

In this paper, we consider algorithms in the standard integer RAM model with $\Theta(\log n)$-size words, but in which the input $q \in \mathbb{R}^n$ is accessible *only* via a linear query oracle. Hence we are not allowed to manipulate the input numbers directly. The complexity is measured in two ways: by counting the total number of queries, just as in the linear decision tree model, and by measuring the overall running time, taking into account the time required to determine the sequence of linear queries. This two-track computation model, in which the running time is distinguished from the query complexity, is commonly used in results on comparison-based sorting problems where analyses of both runtime and comparisons are of interest (see for instance [31, 10, 11]).

2.2 Previous Results

The seminal paper by Gajentaan and Overmars [20] showed the crucial role of 3SUM in understanding the complexity of several problems in computational geometry. Since then, there has been an enormous amount of work focusing on the complexity of 3SUM and this problem is now considered a key tool of complexity-within-P [20, 7, 28, 6, 29, 5, 2, 23, 25, 1, 3, 13]. The current conjecture is that no $O(n^{2-\delta})$-time algorithm exists for 3SUM. It has been known for long that k-SUM is $W[1]$-hard. Recently, it was shown to be $W[1]$-complete by Abboud et al. [1].

In Erickson [17], it is shown that we cannot solve 3SUM in subquadratic time in the 3-linear decision tree model:

▶ **Theorem 1** (Erickson [17]). *The optimal depth of a k-linear decision tree that solves the k-LDT problem is $\Theta(n^{\lceil \frac{k}{2} \rceil})$.*

The proof uses an adversary argument which can be explained geometrically. As we already observed, we can solve k-LDT problems by modeling them as point location problems in an arrangement of hyperplanes. Solving one such problem amounts to determining which cell of the arrangement contains the input point. The adversary argument of Erickson [17] is that there exists a cell having $\Omega(n^{\lceil \frac{k}{2} \rceil})$ boundary facets and in this model point location in such a cell requires testing each facet.

Ailon and Chazelle [4] study s-linear decision trees to solve the k-SUM problem when $s > k$. In particular, they give an additional proof for the $\Omega(n^{\lceil \frac{k}{2} \rceil})$ lower bound of Erickson [17] and generalize the lower bound for the s-linear decision tree model when $s > k$. Note that the exact lower bound given by Erickson [17] for $s = k$ is $\Omega((nk^{-k})^{\lceil \frac{k}{2} \rceil})$ while the one given by Ailon and Chazelle [4] is $\Omega((nk^{-3})^{\lceil \frac{k}{2} \rceil})$. Their result improves therefore the lower bound for $s = k$ when k is large. The lower bound they prove for $s > k$ is the following:

▶ **Theorem 2** (Ailon and Chazelle [4]). *The depth of an s-linear decision tree solving the k-LDT problem is*

$$\Omega\left(\left(nk^{-3}\right)^{\frac{2k-s}{2\lceil\frac{s-k+1}{2}\rceil}(1-\epsilon_k)}\right),$$

where $\epsilon_k > 0$ tends to 0 as $k \to \infty$.

This lower bound breaks down when $k = \Omega(n^{1/3})$ or $s \geq 2k$ and the cases where $k < 6$ give trivial lower bounds. For example, in the case of 3SUM with $s = 4$ we only get an $\Omega(n)$ lower bound.

As for upper bounds, Baran et al. [6] gave subquadratic Las Vegas algorithms for 3SUM on integer and rational numbers in the circuit RAM, word RAM, external memory, and cache-oblivious models of computation. The idea of their approach is to exploit the parallelism of the models, using linear and universal hashing.

More recently, Grønlund and Pettie [23] proved the existence of a linear decision tree solving the 3SUM problem using a strongly subquadratic number of linear queries. The classical quadratic algorithm for 3SUM uses 3-linear queries while the decision tree of Grønlund and Pettie uses 4-linear queries and requires $O(n^{3/2}\sqrt{\log n})$ of them. Moreover, they show that their decision tree can be used to get better upper bounds for k-SUM when k is odd.

They also provide two subquadratic 3SUM algorithms. A deterministic one running in $O(n^2/(\log n/\log\log n)^{2/3})$ time and a randomized one running in $O(n^2(\log\log n)^2/\log n)$ time with high probability. These results refuted the long-lived conjecture that 3SUM cannot be solved in subquadratic time in the RAM model.

Freund [19] and Gold and Sharir [21] later gave improvements on the results of Grønlund and Pettie [23]. Freund [19] gave a deterministic algorithm for 3SUM running in $O(n^2 \log\log n/\log n)$ time. Gold and Sharir [21] gave another deterministic algorithm for 3SUM with the same running time and shaved off the $\sqrt{\log n}$ factor in the decision tree complexities of 3SUM and k-SUM given by Grønlund and Pettie.

Meyer auf der Heide [27] gave the first point location algorithm to solve the knapsack problem in the linear decision tree model in polynomial time. He thereby answers a question raised by Dobkin and Lipton [15, 16], Yao [32] and others. However, if one uses this algorithm to locate a point in an arbitrary arrangement of hyperplanes the running time is increased by a factor linear in the greatest coefficient in the equations of all hyperplanes. On the other hand, the complexity of Meiser's point location algorithm is polynomial in the dimension, logarithmic in the number of hyperplanes and does not depend on the value of the coefficients in the equations of the hyperplanes. A useful complete description of the latter is also given by Bürgisser et al. [9] (Section 3.4).

3 Query complexity

In this section and the next, we prove the following first result.

▶ **Theorem 3.** *There exist linear decision trees of depth at most $O(n^3 \log^2 n)$ solving the k-SUM and the k-LDT problems. Furthermore, for the two problems there exists an $\tilde{O}(n^{\lceil\frac{k}{2}\rceil+8})$ Las Vegas algorithm in the RAM model expected to perform $O(n^3 \log^2 n)$ linear queries on the input.*

3.1 Algorithm outline

For a fixed set of hyperplanes $\mathcal{H}$ and given input vertex q in $\mathbb{R}^n$, Meiser's algorithm allows us to determine the cell of the arrangement $\mathcal{A}(\mathcal{H})$ that contains q in its interior (or that *is* q if q is a 0-cell of $\mathcal{A}(\mathcal{H})$), that is, the positions $\sigma(H,q) \in \{-,0,+\}$ of q with respect to all hyperplanes $H \in \mathcal{H}$. In the k-SUM problem, the set $\mathcal{H}$ is the set of $\Theta(n^k)$ hyperplanes with equations of the form $x_{i_1} + x_{i_2} + \cdots + x_{i_k} = 0$. These equations can be modified accordingly for k-LDT.

We use standard results on ε-nets. By combining a theorem due to Blumer et al. [8] with the results of Meiser [26][4], it is possible to construct an ε-net $\mathcal{N}$ for the range space defined by hyperplanes and simplices using a random uniform sampling on $\mathcal{H}$.

▶ **Theorem 4.** *For all real numbers $\varepsilon > 0, c \geq 1$, if we choose at least $cn^2 \log n\varepsilon^{-1} \log \varepsilon^{-1}$ hyperplanes of $\mathcal{H}$ uniformly at random and denote this selection $\mathcal{N}$ then for any simplex intersected by more than $\varepsilon|\mathcal{H}|$ hyperplanes of $\mathcal{H}$, with probability $1 - 2^{-\Omega(c)}$, at least one of the intersecting hyperplanes is contained in $\mathcal{N}$.*

The contrapositive states that if no hyperplane in $\mathcal{N}$ intersects a given simplex, then with high probability the number of hyperplanes of $\mathcal{H}$ intersecting the simplex is at most $\varepsilon|\mathcal{H}|$.

We can use this to design a prune and search algorithm as follows:

(A) construct an ε-net $\mathcal{N}$,
(B) compute the cell C of $\mathcal{A}(\mathcal{N})$ containing the input point q in its interior,
(C) construct a simplex S inscribed in C and containing q in its interior,
(D) recurse on the hyperplanes of $\mathcal{H}$ intersecting the interior of S.

Proceeding this way with a constant ε guarantees that at most a constant fraction ε of the hyperplanes remains after the pruning step, and thus the cumulative number of queries made to determine the enclosing cell at each step is $O(n^2 \log n \log |\mathcal{H}|)$ when done in a brute-force way. However, we still need to explain how to find a simplex S inscribed in C and containing q in its interior. This procedure corresponds to the well-known *bottom vertex decomposition* (or *triangulation*) of a hyperplane arrangement [22, 14].

3.2 Finding a simplex

In order to simplify the exposition of the algorithm, we assume, without loss of generality, that the input numbers q_i all lie in the interval $[-1, 1]$. This assumption is justified by observing that we can normalize all the input numbers by the largest absolute value of a component of q. One can then see that every linear query on the normalized input can be implemented as a linear query on the original input. A similar transformation can be carried out for the k-LDT problem. This allows us to use bounding hyperplanes of equations $x_i = \pm 1, i \in [n]$. We denote by $\mathcal{B}$ this set of hyperplanes. Hence, if we choose a subset $\mathcal{N}$ of the hyperplanes, the input point is located in a bounded cell of the arrangement $\mathcal{A}(\mathcal{N} \cup \mathcal{B})$. Note that $|\mathcal{N} \cup \mathcal{B}| = O(|\mathcal{N}|)$ for all interesting values of ε.

We now explain how to construct S under this assumption. The algorithm can be sketched as follows. (Recall that $\sigma(H,p)$ denotes the relative position of p with respect to the hyperplane H.)

[4] Note that Meiser used an older result due to Haussler and Welzl [24] and got an extra $\log n$ factor in the size of the ε-net.

Algorithm 1 (Constructing S)**.**

```
Input: A point $q$ in $[-1,1]^n$, a set $\mathcal{I}$ of hyperplanes not containing $q$, and a set $\mathcal{E}$ of
    hyperplanes in general position containing $q$, such that the cell
        $C = \{\, p\colon\ \sigma(H,p) = \sigma(H,q) \text{ or } \sigma(H,p) = 0 \text{ for all } H \in (\mathcal{I} \cup \mathcal{E})\,\}$
    is a bounded polytope. The value $\sigma(H,q)$ is known for all $H \in (\mathcal{I} \cup \mathcal{E})$.
Output: A simplex $S \in C$ that contains $q$ in its interior (if it is not a point), and all
    vertices of which are vertices of $C$.
0. If $|\mathcal{E}| = n$, return $q$.
1. Determine a vertex $\nu$ of $C$.
2. Let $q'$ be the projection of $q$ along $\vec{\nu q}$ on the boundary of $C$. Compute $\mathcal{I}_\theta \subseteq \mathcal{I}$, the
    subset of hyperplanes in $\mathcal{I}$ containing $q'$. Compute $\mathcal{I}_\tau \subseteq \mathcal{I}_\theta$, a maximal subset of those
    hyperplanes such that $\mathcal{E}' = \mathcal{E} \cup \mathcal{I}_\tau$ is a set of hyperplanes in general position.
3. Recurse on $q'$, $\mathcal{I}' = \mathcal{I} \setminus \mathcal{I}_\theta$, and $\mathcal{E}'$, and store the result in $S'$.
4. Return $S$, the convex hull of $S' \cup \{\, \nu \,\}$.
```

Step **0** is the base case of the recursion: when there is only one point left, just return that point. This step uses no query.

We can solve step **1** by using linear programming with the known values of $\sigma(H,q)$ as linear constraints. We arbitrarily choose an objective function with a gradient non-orthogonal to all hyperplanes in $\mathcal{I}$ and look for the optimal solution. The optimal solution being a vertex of the arrangement, its coordinates are independent of q, and thus this step involves no query at all.

Step **2** prepares the recursive step by finding the hyperplanes containing q'. This can be implemented as a ray-shooting algorithm that performs a number of comparisons between projections of q on different hyperplanes of $\mathcal{I}$ without explicitly computing them. In Appendix A, we prove that all such comparisons can be implemented using $O(|\mathcal{I}|)$ linear queries. Constructing $\mathcal{E}'$ can be done by solving systems of linear equations that do not involve q.

In step **3**, the input conditions are satisfied, that is, $q' \in [-1,1]^n$, $\mathcal{I}'$ is a set of hyperplanes not containing q', $\mathcal{E}'$ is a set of hyperplanes in general position containing q', C' is a d-cell of C and is thus a bounded polytope. The value $\sigma(H,q')$ differs from $\sigma(H,q)$ only for hyperplanes that have been removed from $\mathcal{I}$, and for those $\sigma(H,q') = 0$, hence we know all necessary values $\sigma(H,q')$ in advance.

Since $|\mathcal{I}'| < |\mathcal{I}|$, $|\mathcal{E}'| > |\mathcal{E}|$, and $|\mathcal{I} \setminus \mathcal{I}'| - |\mathcal{E}' \setminus \mathcal{E}| \geq 0$, the complexity of the recursive call is no more than that of the parent call, and the maximal depth of the recursion is n. Thus, the total number of linear queries made to compute S is $O(n|\mathcal{I}|)$.

Hence given an input point $q \in [-1,1]$, an arrangement of hyperplanes $\mathcal{A}(\mathcal{N})$, and the value of $\sigma(H,q)$ for all $H \in (\mathcal{N} \cup \mathcal{B})$, we can compute the desired simplex S by running Algorithm 1 on q, $\mathcal{I} = \{\, H \in (\mathcal{N} \cup \mathcal{B})\colon\ \sigma(H,q) \neq 0\,\}$, and $\mathcal{E} \subseteq (\mathcal{N} \cup \mathcal{B}) \setminus \mathcal{I}$. This uses $O(n^3 \log n)$ linear queries. Figure 1 illustrates a step of the algorithm.

3.3 Assembling the pieces

Let us summarize the algorithm
Algorithm 2.

```
Input: $q \in [-1,1]^n$
```

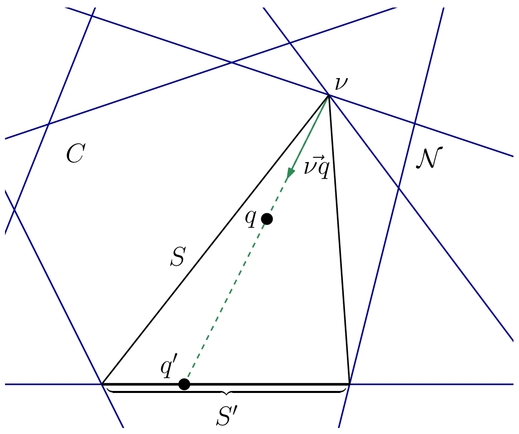


Figure 1 Illustration of a step of Algorithm 1.

1. Pick $O(n^2 \log n)$ hyperplanes of $\mathcal{H}$ at random and locate q in this arrangement. Call C the cell containing q.
2. Construct the simplex S containing q and inscribed in C, using Algorithm 1.
3. For every hyperplane of $\mathcal{H}$ containing S, output a solution.
4. Recurse on hyperplanes of $\mathcal{H}$ intersecting the interior of S.

The query complexity of step **1** is $O(n^2 \log n)$, and that of step **2** is $O(n^3 \log n)$. Steps **3** and **4** do not involve any query at all. The recursion depth is $O(\log |\mathcal{H}|)$, with $|\mathcal{H}| = O(n^k)$, hence the total query complexity of this algorithm is $O(n^3 \log^2 n)$. This proves the first part of Theorem 3.

We can also consider the overall complexity of the algorithm in the RAM model, that is, taking into account the steps that do not require any query, but for which we still have to process the set $\mathcal{H}$. Note that the complexity bottleneck of the algorithm are steps **3-4**, where we need to prune the list of hyperplanes according to their relative positions with respect to S. For this purpose, we simply maintain explicitly the list of all hyperplanes, starting with the initial set corresponding to all k-tuples. Then the pruning step can be performed by looking at the position of each vertex of S relative to each hyperplane of $\mathcal{H}$. Because in our case hyperplanes have only k nonzero coefficients, this uses a number of integer arithmetic operations on $\tilde{O}(n)$ bits integers that is proportional to the number of vertices times the number of hyperplanes. (For the justification of the bound on the number of bits needed to represent vertices of the arrangement see Appendix D.) Since we recurse on a fraction of the set, the overall complexity is $\tilde{O}(n^2|\mathcal{H}|) = \tilde{O}(n^{k+2})$. The next section is devoted to improving this running time.

4 Time complexity

Proving the second part of Theorem 3 involves efficient implementations of the two most time-consuming steps of Algorithm 2. In order to efficiently implement the pruning step, we define an intermediate problem, that we call the *double k-SUM* problem.

▶ **Problem** (double k-SUM). *Given two vectors $\nu_1, \nu_2 \in [-1,1]^n$, where the coordinates of ν_i can be written down as fractions whose numerator and denominator lie in the interval*

$[-M, M]$, enumerate all $i \in [n]^k$ such that

$$\left(\sum_{j=1}^{k} \nu_{1,i_j}\right)\left(\sum_{j=1}^{k} \nu_{2,i_j}\right) < 0.$$

In other words, we wish to list all hyperplanes of $\mathcal{H}$ intersecting the open line segment $\nu_1\nu_2$. We give an efficient output-sensitive algorithm for this problem.

▶ **Lemma 5.** *The double k-SUM problem can be solved in time $O(n^{\lceil \frac{k}{2} \rceil} \log n \log M + Z)$, where Z is the size of the solution.*

Proof. If k is even, we consider all possible $\frac{k}{2}$-tuples of numbers in ν_1 and ν_2 and sort their sums in increasing order. This takes time $O(n^{\frac{k}{2}} \log n)$ and yields two permutations π_1 and π_2 of $[n^{\frac{k}{2}}]$. If k is odd, then we sort both the $\lceil \frac{k}{2} \rceil$-tuples and the $\lfloor \frac{k}{2} \rfloor$-tuples. For simplicity, we will only consider the even case in what follows. The odd case carries through.

We let $N = n^{\frac{k}{2}}$. For $i \in [N]$ and $m \in \{1, 2\}$, let $\Sigma_{m,i}$ be the sum of the $\frac{k}{2}$ components of the ith $\frac{k}{2}$-tuple in ν_m, in the order prescribed by π_m.

We now consider the two $N \times N$ matrices M_1 and M_2 giving all possible sums of two $\frac{k}{2}$-tuples, for both ν_1 with the ordering π_1 and ν_2 with the ordering π_2.

We first solve the k-SUM problem on ν_1, by finding the sign of all pairs $\Sigma_{1,i} + \Sigma_{1,j}$, $i, j \in [N]$. This can be done in time $O(N)$ by parsing the matrix M_1, just as in the standard k-SUM algorithm. We do the same with M_2.

The set of all indices $i, j \in [N]$ such that $\Sigma_{1,i} + \Sigma_{1,j}$ is positive forms a staircase in M_1. We sweep M_1 column by column in order of increasing $j \in [N]$, in such a way that the number of indices i such that $\Sigma_{1,i} + \Sigma_{1,j} > 0$ is growing. For each new such value i that is encountered during the sweep, we insert the corresponding $i' = \pi_2(\pi_1^{-1}(i))$ in a balanced binary search tree.

After each sweep step in M_1 — that is, after incrementing j and adding the set of new indices i' in the tree — we search the tree to identify all the indices i' such that $\Sigma_{2,i'} + \Sigma_{2,j'} < 0$, where $j' = \pi_2(\pi_1^{-1}(j))$. Since those indices form an interval in the ordering π_2 when restricted to the indices in the tree, we can search for the largest i'_0 such that $\Sigma_{2,i'_0} < -\Sigma_{2,j'}$ and retain all indices $i' \le i'_0$ that are in the tree. If we denote by z the number of such indices, this can be done in $O(\log N + z) = O(\log n + z)$ time. Now all the pairs i', j' found in this way are such that $\Sigma_{1,i} + \Sigma_{1,j}$ is positive and $\Sigma_{2,i'} + \Sigma_{2,j'}$ is negative, hence we can output the corresponding k-tuples. To get all the pairs i', j' such that $\Sigma_{1,i} + \Sigma_{1,j}$ is negative and $\Sigma_{2,i'} + \Sigma_{2,j'}$ positive, we repeat the sweeping algorithm after swapping the roles of ν_1 and ν_2.

Every matching k-tuple is output exactly once, and every $\frac{k}{2}$-tuple is inserted at most once in the binary search tree. Hence the algorithm runs in the claimed time.

Note that we only manipulate rational numbers that are the sum of at most k rational numbers of size $O(\log M)$. ◀

Now observe that a hyperplane intersects the interior of a simplex if and only if it intersects the interior of one of its edges. Hence given a simplex S we can find all hyperplanes of $\mathcal{H}$ intersecting its interior by running the above algorithm $\binom{n}{2}$ times, once for each pair of vertices (ν_1, ν_2) of S, and take the union of the solutions. The overall running time for this implementation will therefore be $\tilde{O}(n^2(n^{\lceil \frac{k}{2} \rceil} \log M + Z))$, where Z is at most the number of intersecting hyperplanes and M is to be determined later. This provides an implementation of the pruning step in Meiser's algorithm, that is, step **4** of Algorithm 2.

▶ **Corollary 6.** *Given a simplex S, we can compute all k-SUM hyperplanes intersecting its interior in $\tilde{O}(n^2(n^{\lceil \frac{k}{2} \rceil} \log M + Z))$ time, where $\log M$ is proportional to the number of bits necessary to represent S.*

In order to detect solutions in step **3** of Algorithm 2, we also need to be able to quickly solve the following problem.

▶ **Problem** (multiple k-SUM). *Given d points $\nu_1, \nu_2, \ldots, \nu_d \in \mathbb{R}^n$, where the coordinates of ν_i can be written down as fractions whose numerator and denominator lie in the interval $[-M, M]$, decide whether there exists a hyperplane with equation of the form $x_{i_1} + x_{i_2} + \cdots + x_{i_k} = 0$ containing all of them.*

Here the standard k-SUM algorithm can be applied, taking advantage of the fact that the coordinates lie in a small discrete set.

▶ **Lemma 7.** *k-SUM on n integers $\in [-V, V]$ can be solved in time $\tilde{O}(n^{\lceil \frac{k}{2} \rceil} \log V)$.*

▶ **Lemma 8.** *Multiple k-SUM can be solved in time $\tilde{O}(dn^{\lceil \frac{k}{2} \rceil + 2} \log M)$.*

Proof. Let $\mu_{i,j}$ and $\delta_{i,j}$ be the numerator and denominator of $\nu_{i,j}$ when written as an irreducible fraction. We define

$$\zeta_{i,j} = \nu_{i,j} \prod_{(i,j) \in [d] \times [n]} \delta_{i,j} = \frac{\mu_{i,j} \prod_{(i',j') \in [d] \times [n]} \delta_{i',j'}}{\delta_{i,j}}.$$

By definition $\zeta_{i,j}$ is an integer and its absolute value is bounded by $U = M^{n^2}$, that is, it can be represented using $O(n^2 \log M)$ bits. Moreover, if one of the hyperplanes contains the point $(\zeta_{i,1}, \zeta_{i,2}, \ldots, \zeta_{i,n})$, then it contains ν_i. Construct n integers of $O(dn^2 \log M)$ bits that can be written $\zeta_{1,j} + U, \zeta_{2,j} + U, \ldots, \zeta_{d,j} + U$ in base $2Uk + 1$. The answer to our decision problem is "yes" if and only if there exists k of those numbers whose sum is $kU, kU, \ldots, kU$. We simply subtract the number $U, U, \ldots, U$ to all n input numbers to obtain a standard k-SUM instance on n integers of $O(dn^2 \log M)$ bits. ◀

We now have efficient implementations of steps **3** and **4** of Algorithm 2 and can proceed to the proof of the second part of Theorem 3.

Proof. The main idea consists of modifying the first iteration of Algorithm 2, by letting $\varepsilon = \Theta(n^{-\frac{k}{2}})$. Hence we pick a random subset $\mathcal{N}$ of $O(n^{k/2+2} \log n)$ hyperplanes in $\mathcal{H}$ and use this as an ε-net. This can be done efficiently, as shown in Appendix C.

Next, we need to locate the input q in the arrangement induced by $\mathcal{N}$. This can be done by running Algorithm 2 on the set $\mathcal{N}$. From the previous considerations on Algorithm 2, the running time of this step is

$$O(n|\mathcal{N}|) = \tilde{O}(n^{k/2+4}),$$

and the number of queries is $O(n^3 \log^2 n)$.

Then, in order to prune the hyperplanes in $\mathcal{H}$, we have to compute a simplex S that does not intersect any hyperplane of $\mathcal{N}$. For this, we observe that the above call to Algorithm 2 involves computing a sequence of simplices for the successive pruning steps. We save the description of those simplices. Recall that there are $O(\log n)$ of them, all of them contain the input q and have vertices coinciding with vertices of the original arrangement $\mathcal{A}(\mathcal{H})$. In order to compute a simplex S avoiding all hyperplanes of $\mathcal{N}$, we can simply apply Algorithm 1 on the set of hyperplanes bounding the intersection of these simplices. The running time and number of queries for this step are bounded respectively by $n^{O(1)}$ and $O(n^2 \log n)$.

Note that the vertices of S are not vertices of $\mathcal{A}(\mathcal{H})$ anymore. However, their coordinates lie in a finite set (see Appendix D)

▶ **Lemma 9.** *Vertices of S have rational coordinates whose fraction representations have their numerators and denominators absolute values bounded above by $C^{4n^5} n^{2n^5+n^3+\frac{n}{2}}$, where C is a constant.*

We now are in position to perform the pruning of the hyperplanes in $\mathcal{H}$ with respect to S. The number of remaining hyperplanes after the pruning is at most $\varepsilon n^k = O(n^{k/2})$. Hence from Corollary 6, the pruning can be performed in time proportional $\tilde{O}(n^{\lceil k/2 \rceil + 7})$.

Similarly, we can detect any hyperplane of $\mathcal{H}$ containing S using the result of Lemma 8 in time $\tilde{O}(n^{\lceil k/2 \rceil + 8})$. Note that those last two steps do not require any query.

Finally, it remains to detect any solution that may lie in the remaining set of hyperplanes of size $O(n^{k/2})$. We can again fall back on Algorithm 2, restricted to those hyperplanes. The running time is $\tilde{O}(n^{k/2+2})$, and the number of queries is still $O(n^3 \log^2 n)$.

Overall, the maximum running time of a step is $\tilde{O}(n^{\lceil \frac{k}{2} \rceil + 8})$, while the number of queries is always bounded by $O(n^3 \log^2 n)$. ◀

5 Query size

In this section, we consider a simple blocking scheme that allows us to explore a tradeoff between the number of queries and the size of the queries.

▶ **Lemma 10.** *For any integer $b > 0$, an instance of the k-SUM problem on $n > b$ numbers can be split into $O(b^{k-1})$ instances on at most $k\lceil \frac{n}{b} \rceil$ numbers, so that every k-tuple forming a solution is found in exactly one of the subproblems. The transformation can be carried out in time $O(n \log n + b^{k-1})$.*

Proof. Given an instance on n numbers, we can sort them in time $O(n \log n)$, then partition the sorted sequence into b consecutive blocks $B_1, B_2, \ldots, B_b$ of equal size. This partition can be associated with a partition of the real line into b intervals, say $I_1, I_2, \ldots, I_b$. Now consider the partition of $\mathbb{R}^k$ into grid cells defined by the kth power of the partition $I_1, I_2, \ldots, I_b$. The hyperplane of equation $x_1 + x_2 + \cdots + x_k = 0$ hits $O(b^{k-1})$ such grid cells. Each grid cell $I_{i_1} \times I_{i_2} \times \cdots \times I_{i_k}$ corresponds to a k-SUM problem on the numbers in the set $B_{i_1} \cup B_{i_2} \cup \ldots \cup B_{i_k}$ (note that the indices i_j need not be distinct). Hence each such instance has size at most $k\lceil \frac{n}{b} \rceil$. ◀

Combining Lemma 10 and Theorem 3 directly yields the following.

▶ **Theorem 11.** *For any integer $b > 0$, there exists a $k\lceil \frac{n}{b} \rceil$-linear decision tree of depth $\tilde{O}(b^{k-4} n^3)$ solving the k-SUM problem. Moreover, this decision tree can be implemented as an $\tilde{O}(b^{\lfloor \frac{k}{2} \rfloor - 9} n^{\lceil \frac{k}{2} \rceil + 8})$ Las Vegas algorithm.*

The following two corollaries are obtained by taking $b = \Theta(\text{polylog}(n))$, and $b = \Theta(n^\alpha)$, respectively

▶ **Corollary 12.** *There exists an $o(n)$-linear decision tree of depth $\tilde{O}(n^3)$ solving the k-SUM problem. Moreover, this decision tree can be implemented as an $\tilde{O}(n^{\lceil \frac{k}{2} \rceil + 8})$ Las Vegas algorithm.*

▶ **Corollary 13.** *For any α such that $0 < \alpha < 1$, there exists an $O(n^{1-\alpha})$-linear decision tree of depth $\tilde{O}(n^{3+(k-4)\alpha})$ solving the k-SUM problem. Moreover, this decision tree can be implemented as an $\tilde{O}(n^{(1+\alpha)\frac{k}{2}+8.5})$ Las Vegas algorithm.*

Note that the latter query complexity improves on $\tilde{O}(n^{\frac{k}{2}})$ whenever $\alpha < \frac{k-6}{2k-8}$ and $k \geq 7$. By choosing $\alpha = \frac{k-6}{2k-8} - \frac{\beta}{k-4}$ we obtain $O(n^{1-\frac{k-6}{2k-8}+\frac{\beta}{k-4}})$-linear decision trees of depth $\tilde{O}(n^{\frac{k}{2}-\beta})$ for any $k \geq 7$. Hence for instance, we obtain $O(n^{\frac{3}{4}+\frac{\beta}{4}})$-linear decision trees of depth $\tilde{O}(n^{4-\beta})$ for the 8SUM problem.

References

1 Amir Abboud, Kevin Lewi, and Ryan Williams. Losing weight by gaining edges. In *European Symposium on Algorithms (ESA 2014)*, pages 1–12. Springer, 2014.

2 Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In *International Colloquium on Automata, Languages, and Programming (ICALP 2014)*, volume 8572 of *Lecture Notes in Computer Science*, pages 39–51. Springer, 2014.

3 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.

4 Nir Ailon and Bernard Chazelle. Lower bounds for linear degeneracy testing. *J. ACM*, 52(2):157–171, 2005.

5 Amihood Amir, Timothy M. Chan, Moshe Lewenstein, and Noa Lewenstein. On hardness of jumbled indexing. In *International Colloquium on Automata, Languages, and Programming (ICALP 2014)*, volume 8572 of *Lecture Notes in Computer Science*, pages 114–125. Springer, 2014.

6 Ilya Baran, Erik D. Demaine, and Mihai Patrascu. Subquadratic algorithms for 3SUM. *Algorithmica*, 50(4):584–596, 2008.

7 Gill Barequet and Sariel 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.

8 Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Learnability and the vapnik-chervonenkis dimension. *J. ACM*, 36(4):929–965, 1989.

9 Peter Bürgisser, Michael Clausen, and Mohammad Amin Shokrollahi. *Algebraic complexity theory*, volume 315 of *Grundlehren der mathematischen Wissenschaften*. Springer, 1997.

10 Jean Cardinal, Samuel Fiorini, Gwenaël Joret, Raphaël M Jungers, and J Ian Munro. An efficient algorithm for partial order production. *SIAM journal on computing*, 39(7):2927–2940, 2010.

11 Jean Cardinal, Samuel Fiorini, Gwenaël Joret, Raphaël M Jungers, and J Ian Munro. Sorting under partial information (without the ellipsoid algorithm). *Combinatorica*, 33(6):655–697, 2013.

12 Marco Carmosino, Jiawei Gao, Russell Impagliazzo, Ivan Mikhailin, Ramamohan Paturi, and Stefan Schneider. Nondeterministic extensions of the strong exponential time hypothesis and consequences for non-reducibility. *Electronic Colloquium on Computational Complexity (ECCC 2015)*, 22:148, 2015.

13 Timothy M. Chan and Moshe Lewenstein. Clustered integer 3SUM via additive combinatorics. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Symposium on Theory of Computing (STOC 2015)*, pages 31–40. ACM, 2015.

14 Kenneth L. Clarkson. A randomized algorithm for closest-point queries. *SIAM J. Comput.*, 17(4):830–847, 1988.

15 David P. Dobkin and Richard J. Lipton. On some generalizations of binary search. In *Symposium on Theory of Computing (STOC 1974)*, pages 310–316, 1974.

16 David P. Dobkin and Richard J. Lipton. A lower bound of the $\frac{1}{2}n^2$ on linear search programs for the knapsack problem. *J. Comput. Syst. Sci.*, 16(3):413–417, 1978.

17 Jeff Erickson. Lower bounds for linear satisfiability problems. *Chicago Journal of Theoretical Computer Science*, 1999.

18 Hervé Fournier. *Complexité et expressibilité sur les réels.* PhD thesis, École normale supérieure de Lyon, 2001.

19 Ari Freund. Improved subquadratic 3SUM. *Algorithmica*, 2015. To appear.

20 Anka Gajentaan and Mark H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Comput. Geom.*, 5:165–185, 1995.

21 O. Gold and M. Sharir. Improved bounds for 3SUM, k-SUM, and linear degeneracy. *ArXiv e-prints*, 2015. arXiv:1512.05279 [cs.DS].

22 Jacob E. Goodman and Joseph O'Rourke, editors. *Handbook of Discrete and Computational Geometry, Second Edition.* Chapman and Hall/CRC, 2004.

23 Allan Grønlund and Seth Pettie. Threesomes, degenerates, and love triangles. In *Foundations of Computer Science (FOCS 2014)*, pages 621–630. IEEE, 2014.

24 David Haussler and Emo Welzl. ε-nets and simplex range queries. *Discrete & Computational Geometry*, 2(1):127–151, 1987.

25 T. Kopelowitz, S. Pettie, and E. Porat. Higher lower bounds from the 3SUM conjecture. *ArXiv e-prints*, 2014. arXiv:1407.6756 [cs.DS].

26 S. Meiser. Point location in arrangements of hyperplanes. *Information and Computation*, 106(2):286–303, 1993.

27 Friedhelm Meyer auf der Heide. A polynomial linear search algorithm for the n-dimensional knapsack problem. *J. ACM*, 31(3):668–676, 1984.

28 Joseph S. B. Mitchell and Joseph O'Rourke. Computational geometry column 42. *Int. J. Comput. Geometry Appl.*, 11(5):573–582, 2001.

29 Mihai Patrascu. Towards polynomial lower bounds for dynamic problems. In *Symposium on Theory of Computing (STOC 2010)*, pages 603–610, 2010.

30 Mihai Patrascu and Ryan Williams. On the possibility of faster SAT algorithms. In *Symposium on Discrete Algorithms (SODA 2010)*, pages 1065–1075, 2010.

31 William L. Steiger and Ileana Streinu. A pseudo-algorithmic separation of lines from pseudo-lines. *Information Processing Letters*, 53(5):295–299, 1995.

32 Andrew Chi-Chih Yao. On parallel computation for the knapsack problem. *J. ACM*, 29(3):898–903, 1982.

A Keeping queries linear in Algorithm 1

In Algorithm 1, we want to ensure that the queries we make in step **2** are linear and that the queries we will make in the recursion step remain linear too.

▶ **Lemma 14.** *Algorithm 1 can be implemented so that it uses $O(n|I|)$ linear queries.*

Proof. Let us first analyze what the queries of step **2** look like. In addition to the input point q we are given a vertex ν and we want to find the projection q' of q in direction $\vec{\nu q}$ on the hyperplanes of $\mathcal{I}_\theta$. Let the equation of H_i be $\Pi_i(x) = c_i + d_i \cdot x = 0$ where c_i is a scalar and d_i is a vector. The projection of q along $\vec{\nu q}$ on a hyperplane H_i can thus be written[5]

[5] Note that we project from ν instead of q. We are allowed to do this since $\nu + \lambda_i \vec{\nu q} = q + (\lambda_i - 1)\vec{\nu q}$ and there is no hyperplane separating q from ν.

$\rho(q,\nu,H_i) = \nu + \lambda_i \vec{\nu q}$ such that $\Pi_i(\nu + \lambda_i \vec{\nu q}) = c_i + d_i \cdot \nu + \lambda_i d_i \cdot \vec{\nu q} = 0$. Computing the closest hyperplane amounts to finding $\lambda_\theta = \min_{\lambda_i > 0} \lambda_i$. Since $\lambda_i = -\frac{c_i + d_i \cdot \nu}{d_i \cdot \vec{\nu q}}$ we can test whether $\lambda_i > 0$ using the linear query[6] $-\frac{d_i \cdot \vec{\nu q}}{c_i + d_i \cdot \nu} >^? 0$. Moreover, if $\lambda_i > 0$ and $\lambda_j > 0$ we can test whether $\lambda_i < \lambda_j$ using the linear query $\frac{d_i \cdot \vec{\nu q}}{c_i + d_i \cdot \nu} <^? \frac{d_j \cdot \vec{\nu q}}{c_j + d_j \cdot \nu}$. Step **2** can thus be achieved using $O(1)$ $(2k)$-linear queries per hyperplane of $\mathcal{N}$.

In step **4**, the recursive step is carried out on $q' = \nu + \lambda_\theta \vec{\nu q} = \nu - \frac{c_\theta + d_\theta \cdot \nu}{d_\theta \cdot \vec{\nu q}} \vec{\nu q}$ hence comparing λ'_i to 0 amounts to performing the query $-\frac{d_i \cdot \vec{\nu q}'}{c_i + d_i \cdot \nu'} >^? 0$, which is not linear in q. The same goes for comparing λ'_i to λ'_j with the query $\frac{d_i \cdot \vec{\nu q}'}{c_i + d_i \cdot \nu'} <^? \frac{d_j \cdot \vec{\nu q}'}{c_j + d_j \cdot \nu'}$.

However, we can multiply both sides of the inequality test by $d_\theta \vec{\nu q}$ to keep the queries linear as shown below. We must be careful to take into account the sign of the expression $d_\theta \vec{\nu q}$, this costs us one additional linear query.

This trick can be used at each step of the recursion. Let $q^{(0)} = q$, then we have

$$q^{(s+1)} = \nu^{(s)} - \frac{c_{\theta_s} + d_{\theta_s} \cdot \nu^{(s)}}{d_{\theta_s} \cdot \vec{\nu q}^{(s)}} \vec{\nu q}^{(s)}$$

and $(d_{\theta_s} \cdot \vec{\nu q}^{(s)}) q^{(s+1)}$ yields a vector whose components are linear in $q^{(s)}$. Hence, $(\prod_{k=0}^{s} d_{\theta_k} \cdot \vec{\nu q}^{(k)}) q^{(s+1)}$ yields a vector whose components are linear in q, and for all pairs of vectors d_i and $\nu^{(s+1)}$ we have that $(\prod_{k=0}^{s} d_{\theta_k} \cdot \vec{\nu q}^{(k)})(d_i \cdot \vec{\nu q}^{(s+1)})$ is linear in q.

Hence at the sth recursive step of the algorithm, we will perform at most $|\mathcal{N}|$ linear queries of the type

$$-\left(\prod_{k=0}^{s-1} d_{\theta_k} \cdot \vec{\nu q}^{(k)}\right) \frac{d_i \cdot \vec{\nu q}^{(s)}}{c_i + d_i \cdot \nu^{(s)}} \overset{?}{>} 0$$

$|\mathcal{N}| - 1$ linear queries of the type

$$\left(\prod_{k=0}^{s-1} d_{\theta_k} \cdot \vec{\nu q}^{(k)}\right) \frac{d_i \cdot \vec{\nu q}^{(s)}}{c_i + d_i \cdot \nu^{(s)}} \overset{?}{<} \left(\prod_{k=0}^{s-1} d_{\theta_k} \cdot \vec{\nu q}^{(k)}\right) \frac{d_j \cdot \vec{\nu q}^{(s)}}{c_j + d_j \cdot \nu^{(s)}}$$

and a single linear query of the type

$$d_{\theta_{s-1}} \cdot \vec{\nu q}^{(s-1)} \overset{?}{<} 0.$$

In order to detect all hyperplanes H_i such that $\lambda_i = \lambda_\theta$ we can afford to compute the query $f(q) > g(q)$ for all query $f(q) < g(q)$ that we issue, and vice versa.

Note that, without further analysis, the queries can become n-linear as soon as we enter the $\frac{n}{k}$th recursive step. ◀

B Algebraic computation trees

We consider *algebraic computation trees*, whose internal nodes are labeled with arithmetic $(r \leftarrow o_1 \operatorname{op} o_2, \operatorname{op} \in \{+,-,\times,\div\})$ and branching $(z : 0)$ operations. We say that an algebraic computation tree T *realizes* an algorithm A if the paths from the root to the leaves of T correspond to the execution paths of A on all possible inputs $q \in \mathbb{R}^n$, where n is fixed. A leaf is labeled with the output of the corresponding execution path of A. Such a tree is

[6] Note that if $c_i + d_i \cdot \nu = 0$ then $\lambda_i = 0$, we can check this beforehand for free.

well-defined if any internal node labeled $r \leftarrow o_1 \,\mathrm{op}\, o_2$ has outdegree 1 and is such that either $o_k = q_i$ for some i or there exists an ancestor $o_k \leftarrow x \,\mathrm{op}\, y$ of this node, and any internal node labeled $z : 0$ has outdegree 3 and is such that either $z = q_i$ for some i or there exists an ancestor $z \leftarrow x \,\mathrm{op}\, y$ of this node. In the algebraic computation tree model, we define the complexity $f(n)$ of an algorithm A to be the minimum depth of a well-defined computation tree that realizes A for inputs of size n.

In the algebraic computation tree model, we only count the operations that involve the input, that is, members of the input or results of previous operations involving the input. The following theorem follows immediately from the analysis of the linearity of queries

▶ **Theorem 15.** *The algebraic computation tree complexity of k-LDT is $\tilde{O}(n^3)$.*

Proof. We go through each step of Algorithm 2. Indeed, each k-linear query of step **1** can be implemented as $O(k)$ arithmetic operations, so step **1** has complexity $O(|\mathcal{N}|)$. The construction of the simplex in step **2** must be handled carefully. What we need to show is that each n-linear query we use can be implemented using $O(k)$ arithmetic operations. It is not difficult to see from the expressions given in Appendix A that a constant number of arithmetic operations and dot products suffice to compute the queries. A dot product in this case involves a constant number of arithmetic operations because the d_i are such that they each have exactly k non-zero components. The only expression that involves a non-constant number of operations is the product $\prod_{k=0}^{s} d_{\theta_k} \cdot \vec{\nu q}^{(k)}$, but this is equivalent to $(\prod_{k=0}^{s-1} d_{\theta_k} \cdot \vec{\nu q}^{(k)})(d_{\theta_s} \cdot \vec{\nu q}^{(s)})$ where the first factor has already been computed during a previous step and the second factor is of constant complexity. Since each query costs a constant number of arithmetic operations and branching operations, step **2** has complexity $O(n|\mathcal{N}|)$. Finally, steps **3** and **4** are free since they do not involve the input. The complexity of Algorithm 2 in this model is thus also $O(n^3 \log n \log |\mathcal{H}|)$. ◀

C Uniform random sampling

Theorem 4 requires us to pick a sample of the hyperplanes uniformly at random. Actually the theorem is a little stronger; we can draw each element of $\mathcal{N}$ uniformly at random, only keeping distinct elements. This is not too difficult to achieve for k-LDT when the $\alpha_i, i \in [k]$ are all distinct: to pick a hyperplane of the form $\alpha_0 + \alpha_1 x_{i_1} + \alpha_2 x_{i_2} + \cdots + \alpha_k x_{i_k} = 0$ uniformly at random, we can draw each $i_j \in [n]$ independently and there are n^k possible outcomes. However, in the case of k-SUM, we only have $\binom{n}{k}$ distinct hyperplanes. A simple dynamic programming approach solves the problem for k-SUM. For k-LDT we can use the same approach, once for each class of equal α_i.

▶ **Lemma 16.** *Given $n \in \mathbb{N}$ and $(\alpha_0, \alpha_1, \ldots, \alpha_k) \in \mathbb{R}^{k+1}$, m independent uniform random draws of hyperplanes in $\mathbb{R}^n$ with equations of the form $\alpha_0 + \alpha_1 x_{i_1} + \alpha_2 x_{i_2} + \cdots + \alpha_k x_{i_k} = 0$ can be computed in time $O(mk^2 \log n)$ and preprocessing time $O(k^2 n)$.*

Proof. We want to pick an assignment $a = \{ (\alpha_1, x_{i_1}), (\alpha_2, x_{i_2}), \ldots, (\alpha_k, x_{i_k}) \}$ uniformly at random. Note that all x_i are distinct while the α_j can be equal.

Without loss of generality, suppose $\alpha_1 \leq \alpha_2 \leq \cdots \leq \alpha_k$. There is a bijection between assignments and lexicographically sorted k-tuples $((\alpha_1, x_{i_1}), (\alpha_2, x_{i_2}), \ldots, (\alpha_k, x_{i_k}))$.

Observe that x_{i_j} can be drawn independently of $x_{i_{j'}}$ whenever $\alpha_j \neq \alpha_{j'}$. Hence, it suffices to generate a lexicographically sorted $|\chi|$-tuple of x_i for each class χ of equal α_i.

Let $\omega(m, l)$ denote the number of lexicographically sorted l-tuples, where each element comes from a set of m distinct x_i. We have

$$\omega(m,l) = \begin{cases} 1 & \text{if } l = 0 \\ \sum_{i=1}^{m} \omega(i, l-1) & \text{otherwise.} \end{cases}$$

To pick such a tuple $(x_{i_1}, x_{i_2}, \dots, x_{i_l})$ uniformly at random we choose $x_{i_l} = x_o$ with probability

$$P(x_{i_l} = x_o) = \begin{cases} 0 & \text{if } o > m \\ \frac{\omega(o,l-1)}{\omega(m,l)} & \text{otherwise} \end{cases}$$

that we append to a prefix $(l-1)$-tuple (apply the procedure recursively), whose elements come from a set of o symbols. If $l = 0$ we just return the empty tuple.

Obviously, the probability for a given l-tuple to be picked is equal to $\frac{1}{\omega(m,l)}$.

Let X denote the partiton of the α_i into equivalence classes, then the number of assignments is equal to $\prod_{\chi \in X} \omega(n, |\chi|)$. (Note that for k-SUM this is simply $\omega(n,k)$ since there is only a single class of equivalence.) For each equivalence class χ we draw independently a lexicographically sorted $|\chi|$-tuple on n symbols using the procedure above. This yields a given assignment with probability $\frac{1}{\prod_{\chi \in X} \omega(n,|\chi|)}$. Hence, this corresponds to a uniform random draw over the assignments.

It is a well known fact that $\omega(n,k) = \binom{n+k-1}{k-1}$, hence each number we manipulate fits in $O(k \log n)$ bits, that is, $O(k)$ words. Moreover $\omega(n,k) = \omega(n-1,k) + \omega(n-1,k-1)$ so each $\omega(m,l)$ can be computed using a single addition on numbers of $O(k)$ words.

For given n and k, there are at most nk values $\omega(m,l)$ to compute, and for a given k-LDT instance, it must be computed only once. One way to perform the random draws is to compute the cumulative distribution functions of the discrete distributions defined above, then to draw x_{i_l}, we use binary search to find a generated random integer of $O(k)$ words in the cumulative distribution function. Computing the values $\omega(m,l)$ and all cumulative distributions functions can be done as a preprocessing step in $O(k^2 n)$ time. Assuming the generation of a random sequence of words takes linear time, performing a random draw takes time $O(k^2 \log n)$. ◀

D Proof of Lemma 9

▶ **Theorem 17** (Cramer's rule). *If a system of n linear equations for n unknowns, represented in matrix multiplication form $Ax = b$, has a unique solution $x = (x_1, x_2, \dots, x_n)^T$ then, for all $i \in [n]$,*

$$x_i = \frac{\det(A_i)}{\det(A)}$$

where A_i is A with the ith column replaced by the column vector b.

▶ **Lemma 18** (Meyer auf der Heide[27]). *The absolute value of the determinant of an $n \times n$ matrix $M = M_{i=1\dots n, j=1\dots n}$ with integer entries is an integer that is at most $C^n n^{\frac{n}{2}}$, where C is the maximum absolute value in M.*

Proof. The determinant of M must be an integer and is the volume of the hyperparalleliped spanned by the row vectors of M, hence

$$|\det(M)| \leq \prod_{i=1}^{n} \sqrt{\sum_{j=1}^{n} M_{i,j}^2} \leq \left(\sqrt{nC^2}\right)^n \leq C^n n^{\frac{n}{2}}.$$

◀

▶ **Lemma 19.** *The determinant of an $n \times n$ matrix $M = M_{i=1\dots n, j=1\dots n}$ with rational entries can be represented as a fraction whose numerators and denominators absolute values are bounded above by $(ND^{n-1})^n n^{\frac{n}{2}}$ and D^{n^2} respectively, where N and D are respectively the maximum absolute value of a numerator and a denominator in M.*

Proof. Le $\delta_{i,j}$ denote the denominator of $M_{i,j}$. Multiply each row M_i of M by $\prod_j \delta_{i,j}$. Apply Lemma 18. ◀

We can now proceed to the proof of Lemma 9.

Proof. Coefficients of the hyperplanes of the arrangement are constant rational numbers, those can be changed to constant integers (because each hyperplane has at most k nonzero coefficients). Let C denote the maximum absolute value of those coefficients.

Because of Theorem 17 and Lemma 18, vertices of the arrangement have rational coordinates whose numerators and denominators absolute values are bounded above by $C^n n^{\frac{n}{2}}$.

Given simplices whose vertices are vertices of the arrangement, hyperplanes that define the faces of those simplices have rational coefficients whose numerators and denominators absolute values are bounded above by $C^{2n^3} n^{n^3+\frac{n}{2}}$ by Theorem 17 and Lemma 19. (Note that some simplices might be not fully dimensional, but we can handle those by adding vertices with coordinates that are not much larger than that of already existing vertices).

By applying Theorem 17 and Lemma 19 again, we obtain that vertices of the arrangement of those new hyperplanes (and thus vertices of S) have rational coefficients whose numerators and denominators absolute values are bounded above by $C^{4n^5} n^{2n^5+n^3+\frac{n}{2}}$. ◀

Optimal Staged Self-Assembly of General Shapes*

Cameron Chalk[1], Eric Martinez[2], Robert Schweller[3], Luis Vega[4], Andrew Winslow[5], and Tim Wylie[6]

1 **Department of Computer Science, University of Texas – Rio Grande Valley, Brownsville, USA**
`cameron.chalk01@utrgv.edu`

2 **Department of Computer Science, University of Texas – Rio Grande Valley, Brownsville, USA**
`eric.m.martinez02@utrgv.edu`

3 **Department of Computer Science, University of Texas – Rio Grande Valley, Brownsville, USA**
`robert.schweller@utrgv.edu`

4 **Department of Computer Science, University of Texas – Rio Grande Valley, Brownsville, USA**
`luis.a.vega01@utrgv.edu`

5 **Département d'Informatique, Université Libre de Bruxelles, Brussels, Belgium**
`awinslow@ulb.ac.be`

6 **Department of Computer Science, University of Texas – Rio Grande Valley, Brownsville, USA**
`timothy.wylie@utrgv.edu`

Abstract

We analyze the number of stages, tiles, and bins needed to construct $n \times n$ squares and scaled shapes in the staged tile assembly model. In particular, we prove that there exists a staged system with b bins and t tile types assembling an $n \times n$ square using $\mathcal{O}(\frac{\log n - tb - t\log t}{b^2} + \frac{\log\log b}{\log t})$ stages and $\Omega(\frac{\log n - tb - t\log t}{b^2})$ are necessary for almost all n. For a shape S, we prove $\mathcal{O}(\frac{K(S) - tb - t\log t}{b^2} + \frac{\log\log b}{\log t})$ stages suffice and $\Omega(\frac{K(S) - tb - t\log t}{b^2})$ are necessary for the assembly of a scaled version of S, where $K(S)$ denotes the Kolmogorov complexity of S. Similarly tight bounds are also obtained when more powerful *flexible* glue functions are permitted. These are the first staged results that hold for all choices of b and t and generalize prior results. The upper bound constructions use a new technique for efficiently converting each both sources of system complexity, namely the tile types and mixing graph, into a "bit string" assembly.

1998 ACM Subject Classification F.1.1. Models of Computation

Keywords and phrases Tile self-assembly, 2HAM, aTAM, DNA computing, biocomputing

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.26

1 Introduction

The *staged* self-assembly model is a generalization of the *two-handed* [1, 4, 7, 8] or *hierarchical* [5, 12] tile self-assembly models. In tile self-assembly, system monomers are unit squares with edge labels that collide randomly and attach permanently if abutting edge labels match sufficiently. This simple model is an abstraction of a DNA-based molecular implementation at

* Research supported in part by National Science Foundation Grants CCF-1117672, CCF-1555626, and CCF-1422152.

the nanoscale [13, 21] and is computationally universal [21]. The staged variant is motivated by experimental settings, where parallelism and mixing can be achieved (e.g. test tubes). Liquid-handling robots have been used to perform complex mixing instructions in the lab [15], similar to the mixing algorithms of staged self-assembly systems.

The staged model [8] extends the two-handed model by carrying out separate assembly processes in multiple *bins*. Assembly in each bin begins with *input assemblies* previously assembled in other bins. These bins are stratified into stages, and a *mix graph* specifies which bins in the previous stage supply each bin with input assemblies. The *output* of a staged self-assembly system is the set of assemblies produced in the bins of the final stage.

A common goal in the design of self-assembling systems is the construction of a desired shape. Here we consider the design of *efficient* systems with minimal complexity for a given shape. Three metrics exist for staged systems: the number of distinct tile types used in the system (*tile complexity*), the maximum number of bins used in any stage (*bin complexity*), and the number of stages (*stage complexity*). Efficient construction for various classes of shapes [8, 10] and patterns [9, 22] have been considered, and further extensions and variants of the staged self-assembly model have also been studied [1, 3, 11, 16, 17, 18].

Our results. Here we study the two classic benchmarks for the efficiency a tile self-assembly model: the assembly of $n \times n$ squares and arbitrary shapes (with scaling permitted). Previous works [19, 20, 2] achieved matching upper and lower (univariate) bounds on the minimum complexity of systems that assemble these shape classes in the very first tile assembly model [21]. Here we give nearly matching upper and lower (trivariate) bounds for assembling these shapes in the staged model; our results are summarized in Table 1.

For a given number of tile types t and bins b, we prove that any $n \times n$ square is constructed by a system with $\mathcal{O}(\frac{\log n - t\log t - tb}{b^2} + \frac{\log\log b}{\log t})$ stages and a scaled version[1] of any shape S is assembled by a system with $\mathcal{O}(\frac{K(S) - t\log t - tb}{b^2} + \frac{\log\log b}{\log t})$ stages, where $K(S)$ denotes the Kolmogorov complexity of S with respect to some fixed universal Turing machine. We pair these results with nearly matching lower bounds, proving that for almost all natural numbers n^2, $\Omega(\frac{\log n - t\log t - tb}{b^2})$ stages are needed to assemble an $n \times n$ square, and for all shapes S, $\Omega(\frac{K(S) - t\log t - tb}{b^2})$ stages are needed to assemble a scaled version of a given shape S.

We further explore the stage complexity of these shapes within the *flexible glue* model of tile attachment [6] (where non-matching glue labels can have strength), and prove that $n \times n$ squares and scaled shapes can be assembled using $\mathcal{O}(\frac{\log n - t^2 - tb}{b^2} + \frac{\log\log b}{\log t})$ and $\mathcal{O}(\frac{K(S) - t^2 - tb}{b^2} + \frac{\log\log b}{\log t})$ stages, respectively. We pair this with nearly matching lower bound stage complexities of $\Omega(\frac{\log n - t^2 - tb}{b^2})$ and $\Omega(\frac{K(S) - t^2 - tb}{b^2})$.

Our upper bounds both use a new technique to efficiently assemble *bit string pads*: constant-width assemblies with an exposed sequence of glues encoding a given bit string. This technique converts all three forms of system complexity (tile, bin, and stage) into bits of the string with only a constant-factor loss of information. In other words, the number of bits in the bit string pad rises linearly with the number of bits needed to specify the tile types and mix graph of the construction.

[1] The scale factor is proportional to the product of the time and space used by the fixed universal Turing machine to encode S using $K(S)$ bits.

[2] The fraction of values for which the statement holds reaches 1 in the limit as $n \to \infty$.

Table 1 The main results obtained in this work: upper and lower bounds on the number of stages of a staged self-assembly system with b bins and t tile types uniquely assembling $n \times n$ squares and scaled shapes. $K(S)$ denotes the Kolmogorov complexity of a shape.

Standard Glue Stage Complexity Results

Shape	Upper Bound	Theorem	Lower Bound	Theorem
$n \times n$	$\mathcal{O}(\frac{\log n - t\log t - tb}{b^2} + \frac{\log\log b}{\log t})$	11	$\Omega(\frac{\log n - t\log t - tb}{b^2})$	12
Scaled shapes	$\mathcal{O}(\frac{K(S) - t\log t - tb}{b^2} + \frac{\log\log b}{\log t})$	13	$\Omega(\frac{K(S) - t\log t - tb}{b^2})$	14

Flexible Glue Stage Complexity Results

Shape	Upper Bound	Theorem	Lower Bound	Theorem
$n \times n$	$\mathcal{O}(\frac{\log n - t^2 - tb}{b^2} + \frac{\log\log b}{\log t})$	20	$\Omega(\frac{\log n - t^2 - tb}{b^2})$	21
Scaled shapes	$\mathcal{O}(\frac{K(S) - t^2 - tb}{b^2} + \frac{\log\log b}{\log t})$	22	$\Omega(\frac{K(S) - t^2 - tb}{b^2})$	23

Comparison with prior work. In providing a class of nearly optimal staged systems for any choice of bin and tile count, our results also generalize and improve on prior results. For instance, Theorem 11 implies construction of $n \times n$ squares using $\mathcal{O}(1)$ bins, $\mathcal{O}(\frac{\log n}{\log\log n})$ tile types, and $\mathcal{O}(1)$ stages, matching a result of [2] (up to constant factors). For flexible glues, this is improved to $\mathcal{O}(\sqrt{\log n})$ tile types, a result of [6]. The same theorem also yields constructions using $\mathcal{O}(1)$ bins, $\mathcal{O}(1)$ tile types, and $\mathcal{O}(\log n)$ stages (matching a result of [8]) or $\mathcal{O}(\sqrt{\log n})$ bins, $\mathcal{O}(1)$ tile types, and $\mathcal{O}(\log\log\log n)$ stages, substantially improving over the $\mathcal{O}(\log\log n)$ stages used in [8]. For constructing scaled shapes, Theorem 13 implies systems using $\mathcal{O}(1)$ bins, $\mathcal{O}(\frac{K(S)}{\log K(S)})$ tile types, and $\mathcal{O}(1)$ stages, a result of [20].

2 The Staged Assembly Model

Tiles. A *tile* is a non-rotatable unit square with each edge labeled with a *glue* from a set Σ. Each pair of glues $g_1, g_2 \in \Sigma$ has a non-negative integer *strength*, denoted $\text{str}(g_1, g_2)$. Every set Σ contains a special *null glue* whose strength with every other glue is 0. If the glue strengths do not obey $\text{str}(g_1, g_2) = 0$ for all $g_1 \neq g_2$, then the glues are *flexible*. Unless otherwise stated, we assume that glues are not flexible.

Configurations, assemblies, and shapes. A *configuration* is a partial function $A : \mathbb{Z}^2 \to T$ for some set of tiles T, i.e., an arrangement of tiles on a square grid. For a configuration A and vector $\vec{u} = \langle u_x, u_y \rangle \in \mathbb{Z}^2$, $A + \vec{u}$ denotes the configuration $f \circ A$, where $f(x, y) = (x + u_x, y + u_y)$. For two configurations A and B, B is a *translation* of A, written $B \simeq A$, provided that $B = A + \vec{u}$ for some vector $\vec{u}$. For a configuration A, the *assembly* of A is the set $\tilde{A} = \{B : B \simeq A\}$. The *shape* of an assembly $\tilde{A}$ is $\{\text{dom}(A) : A \in \tilde{A}\}$ where dom() is the domain of a configuration. A shape S' is a *scaled* version of shape S provided that for some $k \in \mathbb{N}$ and $D \in S$, $\bigcup_{(x,y) \in D} \bigcup_{(i,j) \in \{0,1,\ldots,k-1\}^2} (kx + i, ky + j) \in S'$.

Bond graphs and stability. For a configuration A, define the *bond graph* G_A to be the weighted grid graph in which each element of dom(A) is a vertex, and the weight of the edge between a pair of tiles is equal to the strength of the coincident glue pair. A configuration is *τ-stable* for $\tau \in \mathbb{N}$ if every edge cut of G_A has strength at least τ, and is *τ-unstable* otherwise. Similarly, an assembly is *τ-stable* provided the configurations it contains are τ-stable. Assemblies $\tilde{A}$ and $\tilde{B}$ are *τ-combinable* into an assembly $\tilde{C}$ provided there exist $A \in \tilde{A}$, $B \in \tilde{B}$, and $C \in \tilde{C}$ such that $A \cup B = C$ and $\tilde{C}$ is τ-stable.

Two-handed assembly and bins. We define the assembly process via bins. A bin is an ordered tuple (S, τ) where S is a set of *initial* assemblies and $\tau \in \mathbb{N}$ is the *temperature*. In this work, τ is always equal to 2. For a bin (S, τ), the set of *produced* assemblies $P'_{(S,\tau)}$ is defined recursively as follows:

1. $S \subseteq P'_{(S,\tau)}$.
2. If $A, B \in P'_{(S,\tau)}$ are τ-combinable into C, then $C \subseteq P'_{(S,\tau)}$.

A produced assembly is *terminal* provided it is not τ-combinable with any other producible assembly, and the set of all terminal assemblies of a bin (S, τ) is denoted $P_{(S,\tau)}$. That is, $P'_{(S,\tau)}$ represents the set of all possible supertiles that can assemble from the initial set S, whereas $P_{(S,\tau)}$ represents only the set of supertiles that cannot grow any further.

If all assemblies in $P'_{(S,\tau)}$ have finite size, then the assemblies in $P_{(S,\tau)}$ are *uniquely* produced by bin (S, τ). Unique production implies that every producible assembly can be repeatedly combined with others to form an assembly in $P_{(S,\tau)}$.

Staged assembly systems. An *r-stage b-bin mix graph M* is an acyclic r-partite digraph consisting of rb vertices $m_{i,j}$ for $1 \le i \le r$ and $1 \le j \le b$, and edges of the form $(m_{i,j}, m_{i+1,j'})$ for some i, j, j'. A *staged assembly system* is a 3-tuple $\langle M_{r,b}, \{T_1, T_2, \dots, T_b\}, \tau\rangle$ where $M_{r,b}$ is an r-stage b-bin mix graph, T_i is a set of tile types, and $\tau \in \mathbb{N}$ is the temperature. Given a staged assembly system, for each $1 \le i \le r$, $1 \le j \le b$, a corresponding bin $(R_{i,j}, \tau)$ is defined as follows:

1. $R_{1,j} = T_j$ (this is a bin in the first stage);
2. For $i \ge 2$, $R_{i,j} = \Big(\bigcup\limits_{k:\ (m_{i-1,k}, m_{i,j}) \in M_{r,b}} P_{(R_{(i-1,k)}, \tau_{i-1,k})} \Big)$.

Thus, bins in stage 1 are tile sets T_j, and each bin in any subsequent stage receives an initial set of assemblies consisting of the terminally produced assemblies from a subset of the bins in the previous stage as dictated by the edges of the mix graph.[3] The *output* of a staged system is the union of the set of terminal assemblies of the bins in the final stage.[4] The output of a staged system is *uniquely produced* provided each bin in the staged system uniquely produces its terminal assemblies.

3 Key Lemmas

Our results rely on two key lemmas. The first is an upper bound on the information content of a staged system that implies the lower bounds on system complexity. The second is a formal statement of the previously mentioned bit string pad construction.

▶ **Lemma 1.** *A staged system of fixed temperature τ with b bins, s stages, and t tile types can be specified using $\mathcal{O}(t \log t + sb^2 + tb)$ bits. Such a system with flexible glues can be specified using $\mathcal{O}(t^2 + sb^2 + tb)$ bits.*

[3] The original staged model [8] only considered $\mathcal{O}(1)$ distinct tile types, and thus for simplicity allowed tiles to be added at any stage (since $\mathcal{O}(1)$ extra bins could hold the individual tile types to mix at any stage). Because systems here may have super-constant tile complexity, we restrict tiles to only be added at the initial stage.

[4] This is a slight modification of the original staged model [8] in that there is no requirement of a final stage with a single output bin. It may be easier in general to solve problems in this variant of the model, so we consider it for lower bound purposes. However, all of our results apply to both variants of the model.

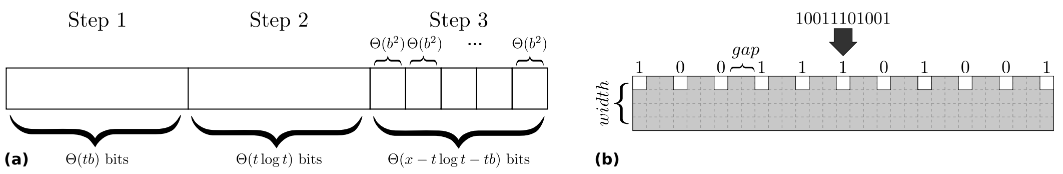


Figure 1 (a) The decomposition of a bit string pad's bits into those encoded by the three steps of a staged system with t tile types and b bins. (b) An example bit string $r = 10011101001$ encoded as a *width*-4 *gap*-2 11-*bit string pad* where the top glues correspond to the bits in r.

Proof. A staged system can be specified in four parts: the tile types, the glue function, the mix graph, and the assignment of tile types to stage-1 bins. We separately bound the number of bits required to specify each.

A set of t tile types has up to $4t$ glue types, so specifying each tile requires $\mathcal{O}(\log t)$ bits, and the entire tile set takes $\mathcal{O}(t \log t)$ bits. If the system does not have flexible glues, then the glue function can be specified in $\mathcal{O}(4t) = \mathcal{O}(t)$ bits, using $\mathcal{O}(\log \tau) = \mathcal{O}(1)$ bits per glue type to specify the glue's strength. If the system has flexible glues, then the glue function can be specified using $\mathcal{O}(1)$ bits per pairwise glue interaction and $\mathcal{O}((4t)^2) = \mathcal{O}(t^2)$ bits total.

The mix graph consists of bs nodes. Each pair of nodes in adjacent stages optionally share a directed edge pointing upwards. Thus specifying these edges takes $\mathcal{O}(b^2(s-1)) = \mathcal{O}(b^2 s)$ bits. The assignment of tile types to stage-1 bins requires one bit per each choice of tile type and bin, or $\mathcal{O}(tb)$ bits total.

Thus a staged system without flexible glues can be specified in $\mathcal{O}(t \log t + t + b^2 s + tb)$ bits, and otherwise in $\mathcal{O}(t \log t + t^2 + b^2 s + tb)$ bits. ◀

It immediately follows from Lemma 1 that for most bit strings of length x, any staged system with b bins and t tiles that encodes the bit string must have $\Omega(\frac{x-tb-t\log t}{b^2})$ stages with standard glues and $\Omega(\frac{x-tb-t^2}{b^2})$ stages with flexible glues.

The two main positive results of this work, efficient assembly of squares and general scaled shapes, both rely mainly on efficient assembly of *bit string pads*: assemblies that expose a sequence of north glues that encode a bit string. An example is shown in Figure 1(b). Squares and general scaled shapes are assembled by combining a universal set of "computation" tiles with efficiently assembled "input" bit string pads.

▶ **Definition 2** (bit string pad). A *width-k gap-f r-bit string pad* is a $k \times (f(r-1)+1)$ rectangular assembly with r glues from a set of two glue types $\{0,1\}$ exposed on the north face of the rectangle at intervals of length f, starting from the leftmost north edge. Unless otherwise specified, a bit string pad is gap-0. All remaining exposed glues on the north tile edges have some common label f. The remaining exposed south, east, and west tile edges have glues g_S, g_E, and g_W. A bit string pad *represents* a given string of r bits if the exposed "0" and "1" glues from left to right are equal to the given bit string.

Bit string pads are constructed by decomposing the pad into three subpads and constructing each in a separate step using a different source of system complexity (see Figure 1(a)):

- Step 1: $\Theta(tb)$ bits from assigning tile types to stage-1 bins (Section 4.2).
- Step 2: $\Theta(t \log t)$ bits from the tile types themselves as in [2, 6, 14, 20] (Section 4.3).
- Step 3: $\Theta(x - t\log t - tb)$ bits from the mix graph using a variant of "crazy mixing" [8] (Section 4.4).

These subpads are then combined into the complete pad. If flexible glues are permitted, Step 2 is modified as in [6] to achieve $\mathcal{O}(\frac{x-tb-t\log t}{b^2} + \frac{\log\log b}{\log t})$ stages.

▶ **Lemma 3.** *There exist constants $c, d \in \mathbb{N}$ such that, for any $t, b \in \mathbb{N}$ with $t > c$, $b > d$ and bit string S of length x, there exists a staged system with b bins, t tiles, and $\mathcal{O}(\frac{x-tb-t\log t}{b^2} + \frac{\log\log b}{\log t})$ stages that assembles a width-9 gap-$\Theta(\log b)$ l-bit string pad representing S.*

Proof. Let $t' = \frac{t-10}{4}$ and $b' = \frac{b-10}{4}$. Using an approach similar to that in Section 4.1, construct a length $1 \times 2\log\frac{b'-15}{9} + 2$ *filler assembly* using t' tile types and b' bins in $\mathcal{O}(\frac{\log\log b'}{\log t'})$ stages such that the assembly has glue e on its west edge (matching that of the east side of the bit string pads) and glue w on its east edge. Next, use Lemma 5 with t' tile types, b' bins, and $\mathcal{O}(\frac{\log\log b'}{\log t'})$ stages to construct a width-9 gap-$2\log\frac{b'-15}{9} + 2$ $\Theta(tb)$-bit string pad. Then use Lemma 9 with t' tile types, b' bins, and $\mathcal{O}(1)$ stages to construct a width-3, gap-$2\log\frac{b'-15}{9} + 2$, $\Theta(t\log t)$-bit string pad.

So far, $\Theta(tb) + \Theta(t\log t)$ bits have been encoded and so $\Theta(x - tb - t\log t)$ bits remain. Invoke Lemma 10 with t' tile types, b' bins, and $\mathcal{O}(\frac{x-tb-t\log t}{b^2} + \frac{\log\log b'}{\log t'})$ stages to construct a width-9 gap-$2\log\frac{b'-15}{9} + 2$ $\Theta(x - tb - t\log t)$-bit string pad. In one final stage, concatenate two bit string pads using the filler assembly and in one more stage concatenate the third.

By concatenating the length $\Theta(tb)$-bit string pad, the length $\Theta(t\log t)$-bit string pad, and the $\Theta(x - tb - t\log t)$-bit string pad, each separated by the $2\log\frac{b'-15}{9} + 2$ filler assembly, an x-bit string pad with $\mathcal{O}(\log b)$ spacing is constructed; use 10 additional tile types (in 10 bins) to "fill in" the portions of the assembly with width less than 9.

The total number of tile types and bins used are $4t' + 10 = t$ and $4b' + 10 = b$, respectively, with $4t'$ and $4b'$ used for the three bit string pads and one connector assembly and the remainder for filling in the pad to width 9. The total number of stages used is $\mathcal{O}(\frac{\log\log b'}{\log t'}) + \mathcal{O}(1) + \mathcal{O}(\frac{x-tb-t\log t}{b^2} + \frac{\log\log b'}{\log t'}) = \mathcal{O}(\frac{\log b}{t} + \frac{x-tb-t\log t}{b^2})$. ◀

The additive gap between the upper and lower bounds implied by these lemmas comes from the $\mathcal{O}(\frac{\log\log b}{\log t})$ additional stages used to construct some of the machinery needed to carry out the three steps of Lemma 3.

4 Bit String Pad Construction

As mentioned, bit string pads are assembled by combining three subpads constructed via separate and independent methods that utilize distinct sources of information complexity in a staged self-assembly system. Each subpad encodes a number of bits roughly proportional to the number needed to describe the corresponding portion of the staged system, i.e., an asympotitically optimal number of bits are encoded.

4.1 Wings

The additive gap in our upper and lower bounds come from a helpful subconstruction used in Steps 1 and 3 described here. This subconstruction assembles all 1-gapped width-2 bit string pads of a given length in separate bins:

▶ **Lemma 4.** *There exist constants $c, d \in \mathbb{N}$ such that, for any $t, b \in \mathbb{N}$ with $t > c$ and $b > d$, there is a staged self-assembly system with b bins, t tile types, and $\mathcal{O}(\frac{\log\log b}{\log t})$ stages that assembles all gap-1, width-2, $\log(b)$-bit string pads, each placed in a distinct bin.*

Due to space constraints, the proof of this and some later results are omitted. We give proof sketches instead. Let $\gamma = \lfloor\frac{t-6}{2}\rfloor$ and $\eta = \gamma + 1$. If $\gamma \geq 2\log(b)$, directly build all the bit string pads in $\mathcal{O}(1)$ stages. Otherwise, repeatedly apply a constant-stage "round" that

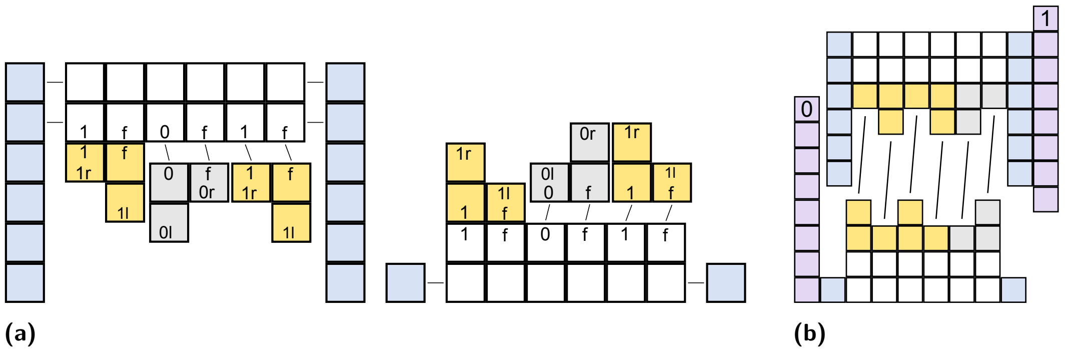


Figure 2 (a) The attachment of extra subassemblies onto bit string pads to create left and right wings. Each of the two size 3 subassemblies use 3 tiles to deterministically assemble the respective L shape in their own bins. **(b)** The attachment of two bit strips using matching wings. Note that the geometry attached to the sides of each wing prevent misaligned, non-matching wings to attach.

starts with all binary gadgets of a given length and yields all binary gadgets of a factor of η longer, starting with just two bit string pads encoding the two bit strings of length 1.

Use $\mathcal{O}(1)$ additional tile types, bins, and stages to augment the the bit string pads assembled by Lemma 4 into left and right *wings* (seen in the left and right portions of Figure 2(a)) that attach when the underlying bit strings are identical. These wings are used in Steps 1 and 3 to achieve ordered assembly of bit string subpads into larger bit string pads.

4.2 Step 1: encoding via initial tile-to-bin assignment

Recall that in a staged system, each of the system's b stage-1 bins is assigned a subset of t total tile types. Here we design an assignment that assembles a $\Theta(tb)$-bit string subpad of the final bit string pad using $\mathcal{O}(\log\log b/\log t)$ stages - enough to utilize the wings of Section 4.1. The assignment yields b bins that contain assemblies encoding distinct equal-length substrings of the $\Theta(tb)$ bits. These assemblies are then combined using wings.

▶ **Lemma 5.** *There exist $c, d \in \mathbb{N}$ such that, for all $t, b \in \mathbb{N}$ with $t > c$ and $b > d$ and bit string S of length $\Theta(tb)$, there is a staged self-assembly system with b bins, t tiles, and $\mathcal{O}(\frac{\log\log b}{\log t})$ stages that assembles a gap-$(2\log\lfloor\frac{b-15}{9}\rfloor + 2)$ $\Theta(tb)$-bit string pad representing S.*

See Figure 3 for a sketch of the idea. Let γ and β be constant fractions of t and b, respectively. Use γ tiles and β bins to construct all left and right $\log(\beta)$-bit wings according to Section 4.1. Also construct $\frac{\gamma}{2}$ constant-sized *bit strip* subassemblies that expose a 0 or 1 north glue and have wings attached to their right and left sides such that any $\frac{\gamma}{2}$-bit string pad can be assembled from $\frac{\gamma}{2}$ bit strips attached sequentially.

In each of β bins, assemble $\frac{\gamma}{2}$ bit strips into a distinct $\frac{\gamma}{2}$-bit string subpad of the desired pad. Combine these β subpads with wings that encode their locations in the pad, and then combine these "wing-labeled" subpads to assemble the complete $\Theta(tb)$-bit string pad. The number of stages used is $\mathcal{O}(\frac{\log\log b}{\log t})$ (for the wings, see Lemma 4) plus $\mathcal{O}(1)$ (the subpads of the desired pad).

4.3 Step 2: encoding via tile types

Here the goal is to design a collection of t tile types that encodes $\Theta(t\log t)$ bits. The solution is to utilize the base conversion approach of [2, 6, 14, 20]. In this approach, tile

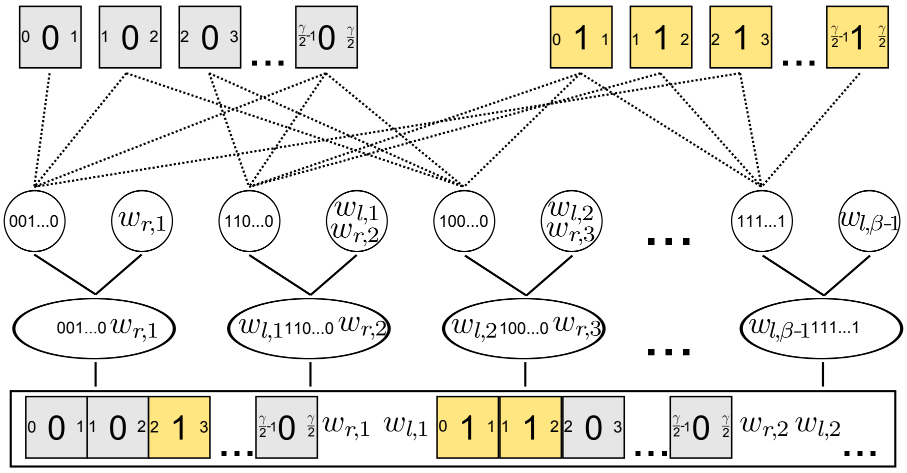


Figure 3 The creation of $\gamma\beta$-bit string pads. The squares labeled 0 and 1 represent bit strips. The dotted lines indicate tile to bin assignments before the first stage of the system; $w_{r,i}$ and $w_{l,i}$ represent the i^{th} right and left wings respectively.

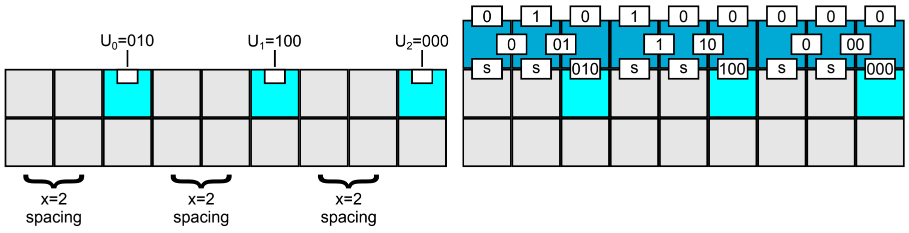


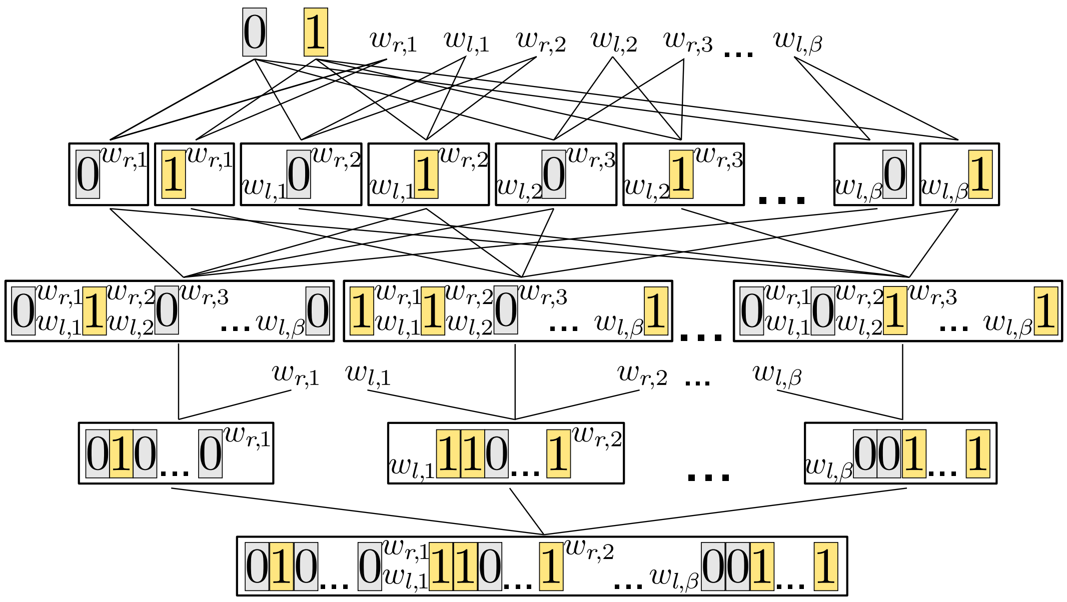
Figure 4 Left: a width-2 gap-log $z-1$ decompression pad representing a bit string $S = 010100000$ in base $z = 8$. Right: $\mathcal{O}(z)$ decompression tiles interact with the north glues of the decompression pad to combine into a width-3 bit string pad representing S in base 2.

types optimally encode integer values in a high base and then "decompressed" into a binary representation. In total, t tile types are used to encode (in a high base) and decompress (into a binary) $\Theta(t \log t)$ bits.

▶ **Definition 6** (decompression pad). For $k, r, x \in \mathbb{N}$ and $u = 2^x$, a width-k, r-digit, base-u decompression pad is a $k \times rx$ rectangular assembly with r glues from a set of $u-1$ glue types $\{0, 1, ..., u-1\}$ exposed on the north face of the rectangle at intervals of length $x-1$ and starting from the leftmost northern edge. All remaining glues on the north surface have a common type n. The remaining exposed south, east, and west tile edges have glues g_S, g_E, and g_W. A decompression pad *represents* a given string of digits in base u if the exposed glues from left to right, disregarding glues of type n, are equal to the given digit string in base u.

Consider the following example, also seen in Figure 4). Let $S = 010100000$ ($S = 240$ in base 8) be a bit string, with the goal of constructing a width-3 9-bit string pad representing S. First, build a decompression pad representing S in base 8 by combining 3 different $3 \times \log_2(8)$ blocks. Then convert the decompression pad into a bit string pad representing S using $\mathcal{O}(z)$ tile types.

▶ **Lemma 7.** *Given integers $x \geq 3$, $d \geq 1$ and $z = 2^x$, there exists a 1-stage, 1-bin staged self-assembly system that assembles a d-digit decompression pad of width-2 and base-z, using at most $5d + \log z - 2$ tile types.*



0
1
$w_{r,1}$ $w_{l,1}$ $w_{r,2}$ $w_{l,2}$ $w_{r,3}$ ... $w_{l,\beta}$

Figure 5 The creation of β^2-bit string pads using β wings and $\mathcal{O}(1)$ stages. The rectangles 0 and 1 represent bit strips that may attach wings on either side; $w_{r,i}$ and $w_{l,i}$ represent the i^{th} right and left wings respectively.

▶ **Lemma 8.** *Given integers $d \geq 3$, $x \geq 3$, $z = 2^x$, and bit string S of length $d\log(z)$, there exists a staged self-assembly system with 1 bin, $5d + 2z + \log z - 4$ tile types, and 1 stage that assembles a width-3 $d\log(z)$-bit string pad representing S.*

▶ **Lemma 9.** *There exists some constant $c \in \mathbb{N}$ such that, for any $t \geq c$ and bit string S of length $\Theta(t\log t)$, there exists a staged self-assembly system with 1 bin, t tile types, and 1 stage assembling a width-3 $\Theta(t\log t)$-bit string pad representing S.*

Omitted additional details are needed to convert these gap-0 pads to higher-gap pads consistent with those assembled in Section 4.4.

4.4 Step 3: encoding via mix graph

This step uses a mix graph to encode encodes a achieves the following efficient assembly:

▶ **Lemma 10.** *There exist $c, d \in \mathbb{N}$ such that, for any $t > c$ and $b > d$ and bit string S of length x, there is a staged self-assembly system with t bins, b tile types, and $\mathcal{O}(\frac{x}{b^2} + \frac{\log\log b}{\log t})$ stages that assembles a width-9 gap-$(2\log\frac{b-15}{9} + 2)$ x-bit string pad representing S.*

An overview of the construction is shown in Figure 5. Let γ and β represent some constant fractions of t and b respectively. Utilize γ tiles and β bins to construct all length-$\log_2(\beta)$ left and right wings according to Section 4.1 and denote the i^{th} left and wings by $w_{l,i}$ and $w_{r,i}$, respectively. Also construct two constant-sized *bit strip* subassemblies that expose a 0 or 1 north glue and allow wings to be attached to their right and left sides.

In the first stage and for all $1 \leq i \leq \beta$, mix $w_{r,i}$ and $w_{l,i-1}$ with bit strip 0 into a bin denoted b_i^0. Similarly, mix $w_{r,i}$ and $w_{l,i-1}$ with bit strip 1 into a bin denoted b_i^1 for a total of 2β bins.

In the second stage, selectively mix specific 0 or 1 winged bit strips to assemble specific β-bit string pads across β bins. Specifically, mix either b_i^0 or b_i^1 for each i across β bins for a total of β different β-bit string pads.

In the third stage, attach wings to each of the β-bit string pads. For each of the β bins, mix $w_{r,i}$ and $w_{l,i-1}$ into the bins such that $w_{r,1}$ is mixed with the first β bits of the desired β^2-bit string pad, $w_{r,2}$ and $w_{l,1}$ are mixed with the second β bits of the desired β^2-bit string pad, etc.

In the final stage, mix all β bins, each containing a β-bit string pads, into a common bin to create β^2-bit string pads. The wings ensure that the bit string pads attach in the desired order. Repeat this process $\frac{x}{\beta^2}$ times, each time concatenating the β^2-bit string pad onto each preceding bit string pad. In the end, a single x-bit string pad results. In total, $\mathcal{O}(\frac{\log\log b}{\log t})$ stages are used to construct the wings and $\mathcal{O}(\frac{x}{b^2})$ stages are used to assemble $\frac{x}{\beta^2}$ unique β^2-bit string pads. Thus this step has total stage complexity of $\mathcal{O}(\frac{x}{\beta^2} + \frac{\log\log\beta}{\log t}) = \mathcal{O}(\frac{x}{b^2} + \frac{\log\log b}{\log t})$.

5 Assembly of $n \times n$ Squares

Efficient assembly of $n \times n$ squares is obtained by combining bit string pads with a technique of Rothemund and Winfree [19]. Their technique utilizes a binary counting mechanism which constructs a length $\Theta(n)$ rectangle with $\Theta(\log n)$ width. The mechanism uses $\mathcal{O}(\log n)$ tile types to seed the counter at a certain value, and then $\mathcal{O}(1)$ tile types attach in a "zig-zag" pattern, where "zigs" copy the value from the row below and "zags" increment the the value by 1. Once the binary counter increments to its maximum value (a string of 1), the assembly stops growing. Two rectangles assembled this way can be combined to form a bounding box that is then filled to form a square. We utilize the bit string pad construction of Section 4 to efficiently assemble the seed for the binary counting mechanism, requiring only an additional $\mathcal{O}(1)$ tile types and 1 stage to perform the binary counting and square filling.

▶ **Theorem 11.** *There exist constants $b_0, t_0 \in \mathbb{N}$ such that for any $b, t, n \in \mathbb{N}$ with $b \geq b_0$, $t \geq t_0$, there exists a staged self-assembly system with b bins, t tile types, and $\mathcal{O}(\frac{\log n - t\log t - tb}{b^2} + \frac{\log\log b}{\log t})$ stages that uniquely produces an $n \times n$ square.*

Proof. Let c be the (constant) number of tile types used to implement the fixed-width "zig-zag" binary counting mechanism shown in [19]. Let $t' = t - c$, $b' = b - 2$, and $n' = \lceil \log n \rceil$. Let $m = 2^{n'-1} - (n-22)/2 - n'(2\log b' + 2)$. Using Lemma 3, construct two $\Theta(\log b)$-gap $\lceil \log m \rceil$-bit string pads encoding m, where each construction each uses b' bins, t' tile types and $\mathcal{O}(\frac{\lceil \log m \rceil - tb - t\log t}{b^2} + \frac{\log\log b}{\log t})$ stages. Figure 6 shows the construction, including modifications to the technique shown in [19].

On both pads, a small modification is made: the glues of the first and last bits are made unique and the first bit's glue strength is set to 2. This modification is necessary to implement a fixed-width binary counting mechanism as in [19] and uses $\mathcal{O}(1)$ additional tile types. Also, on the north-facing (east-facing) bit string pad, a unique strength-2 glue C2 is placed on the south (west) face of the pad's bottommost rightmost (topmost leftmost) tile. This special glue is used to combine the two pads with a unique tile type.

Note that the bit string pads assembled in Section 4 have substantial spacing between the exposed binary glues, but the counter of [19] has spacing 0. This is resolved by adding generic tiles which transfer information horizontally. These generic tiles use cooperative binding between a south-facing f glue (which matches the glue that spaces the bits on the bit string pad) and west/east glues representing the information to be passed horizontally across spacing of f glues. The tiles also expose a north-facing f glue to be used when the

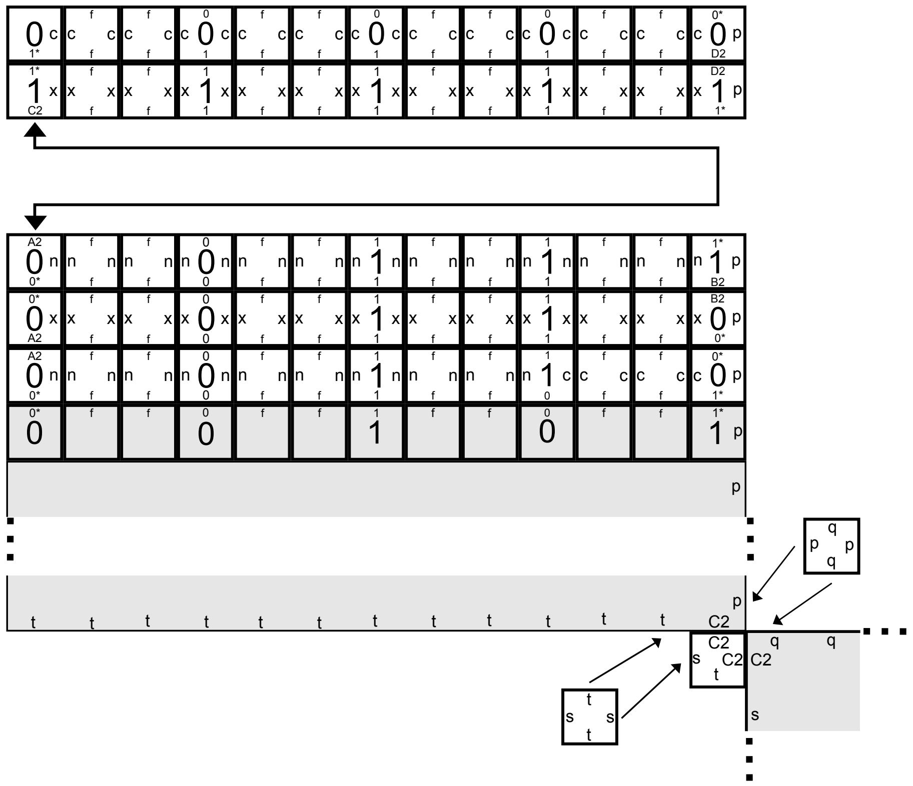


Figure 6 Constructing a counter seed. The bit string pads are shown in gray. Glues with a "2" in the string have strength-2, all other glues have strength 1.

information needs to be transferred across the spacing in the row above. Without loss of generality, rotated versions of these tiles are used in the east-growing counter.

The stage complexity of the system is $\mathcal{O}(\frac{\lceil \log n \rceil - tb - t \log t}{b^2} + \frac{\log \log b}{\log t})$. Note that the length of the bit string pads assembled according to Lemma 3 is dependent on b, the number of bins used to construct the bit string pad. If b is so large that the spacing between bits causes the width of the bit string pad to exceed n (roughly $\log b > n$), we instead directly construct the appropriate bit string pad with spacing 0 using $\mathcal{O}(\frac{\log \log b}{\log t})$ stages. ◀

The following lower bound is derived from Lemma 1 by observing that for almost all $n \in \mathbb{N}$, $\lfloor \log n \rfloor$ bits are needed to represent n.

▶ **Theorem 12.** *For any $b, t \in \mathbb{N}$ and almost all $n \in \mathbb{N}$, any staged self-assembly system which uses at most b bins and t tile types that uniquely assembles an $n \times n$ square must use $\Omega(\frac{\log n - t \log t - tb}{b^2})$ stages.*

6 Assembly of Scaled Shapes

Efficient assembly of arbitrary shapes (up to scaling) is achieved by combining bit string pads with the shape-building scheme of Soloveichik and Winfree [20]. Their construction uses two subsets of tile types: a varying set to encode the binary description of the target shape and a fixed set to decode the binary description and build the shape. We replace the first set with a bit string pad encoding the same information.

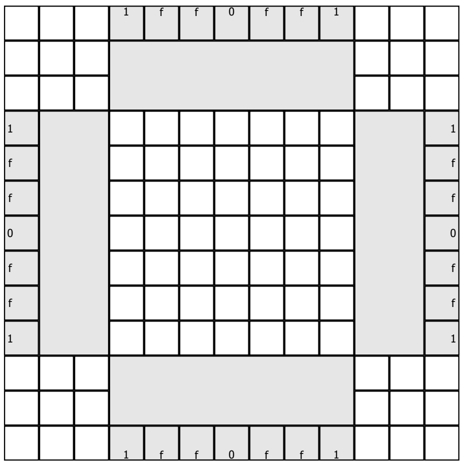


Figure 7 Construction of the modified seed block. Bit string pads are colored in gray. We concatenate four $K(S)$-bit string pads representing S.

▶ **Theorem 13.** *There exist constants $b_0, t_0 \in \mathbb{N}$ such that for any shape S of Kolmogorov complexity $K(S)$ and $b, t \in \mathbb{N}$ such that $b \geq b_0$ and $t \geq t_0$, there exists a staged self-assembly system with b bins, t tile types, and $\mathcal{O}(\frac{K(S)-t\log t-tb}{b^2} + \frac{\log\log b}{\log t})$ stages that uniquely produces S at some scale factor.*

Proof. Observe that the tile set described in [20] uniquely constructs the same terminal assembly, namely a *scaled* version of S where each cell is replaced by a square block of cells, when run at temperature 2 in the two-handed mixing model. It does so via a Kolmogorov-complexity-optimal Turing machine simulation of a machine that computes a spanning tree of the shape given a seed assembly or *seed block* encoding the shape. The simulation is then run as it "fills in" the shape, beginning with the seed block. Here a similar seed block is constructed and consists of four bit string pads, a square "core" and additional filler tiles.

Let $t' = \frac{t-c}{5}$ where c is the (constant) number of tile types required by [20] to carry out the simulation of a (fixed) universal Turing machine. Let $b' = \frac{b-1}{5}$.

Use the method of Lemma 3 to construct the modified seed block by assembling four different $K(S)$-bit string pads representing a program that outputs S, each using b' bins, t' tile types and $\mathcal{O}(\frac{K(S)-t'\log t'-t'b'}{b'^2} + \frac{\log\log b'}{\log t'})$ stages. These four pads (each with dimensions $(2K(S)\log K(S)+2) \times \mathcal{O}(1)$) are attached to the four sides of a $(2K(S)\log K(S)+2) \times (2K(S)\log K(S)+2)$ square constructed as in Theorem 11 using t' tile and b' bins in $\mathcal{O}(\frac{\log(2K(S)\log K(S)+2)-t'\log t'-t'b'}{b'^2} + \frac{\log\log b'}{\log t'})$ stages. An abstract figure of the completed seed block can be seen in Figure 7. The Turing simulation occurs in one stage by mixing the four concatenated bit string pads into one bin which contains the fixed set of Turing-machine-simulation tiles of [20]. The bit string pads contain spacing between the exposed binary glues, while the simulation tile types of [20] expect adjacent glues. This is resolved by modifying the Turing-machine-simulation tile set to include generic tiles for transferring information across spacing, similar to the tiles of the same purpose discussed in the proof of Theorem 11. We need at most 1 such tile for each tile in the (constant-sized) Turing-machine-simulation tile set, for a constant increase in tile complexity. The stage complexity is $4 \times \mathcal{O}(\frac{K(S)-t'\log t'-t'b'}{b'^2} + \frac{\log\log b'}{\log t'}) + \mathcal{O}(\frac{\log(2K(S)\log K(S)+2)-t'\log t'-t'b'}{b'^2} + \frac{\log\log b'}{\log t'}) = \mathcal{O}(\frac{K(S)-t\log t-tb}{b^2} + \frac{\log\log b}{\log t})$. ◀

The following theorem follows from the information-theoretic bound of Lemma 1.

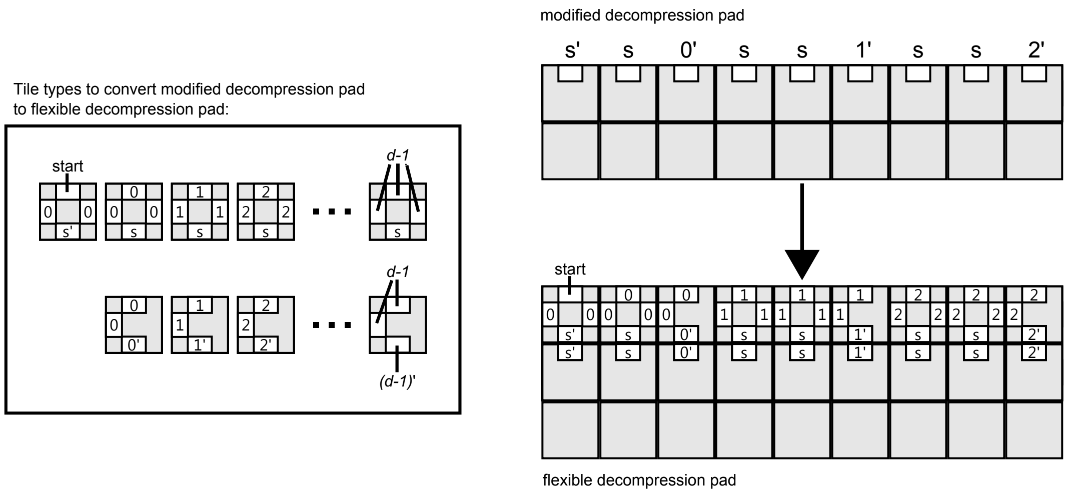

Tile types to convert modified decompression pad
to flexible decompression pad:
start
d-1
(d-1)'
modified decompression pad
s' s 0' s s 1' s s 2'
start
flexible decompression pad

Figure 8 The templates to convert a modified decompression pad to a flexible decompression pad using $2d + 1$ tile types, where integer $d \geq 3$, on the left. Using these additional tile types, a modified decompression pad is converted into a flexible decompression pad. A *modified decompression pad* has a westmost northmost glue of s' and every non-s glue on the north surface is a special *prime* version distinct from other similar glue types. On the left, a width-2 modified decompression pad representing the string 012 in base-8 is converted to a width-3 length-9 flexible decompression pad.

▶ **Theorem 14.** *For any $b, t \in \mathbb{N}$ and shape S with Kolmogorov complexity $K(S)$, any staged self-assembly system which uses at most b bins and t tile types that uniquely assembles S must use $\Omega(\frac{K(S)-t\log t-tb}{b^2})$ stages.*

7 Flexible Glues

Here, an alternate model permitting non-diagonal glue functions, also called *flexible glues* is considered. By modifying Step 2 of the bit string pad construction of Section 4 to encode $\Theta(t^2)$ bits rather than $\Theta(t\log(t))$ bits in t tile types, similarly tight results are obtained for the same problems in this more powerful model. The technique uses a modified decompression bad, similar to the technique introduced in [6].

▶ **Definition 15** (flexible decompression pad)**.** A width-k length-r^2 flexible decompression pad is a $k \times r^2$ rectangular assembly with r^2 north glue types from the set $\{\text{start}, 0, 1, \ldots, r\}$ exposed on the north face of the rectangle. The westmost glue is “start”, the following $r - 1$ glues have type “0”, followed by r glues of type “1”, r glues of type “2”, and so on. The exposed south, east, and west tile edges have glues g_S, g_E, and g_W, respectively.

In order to build the flexible decompression pad, a modified decompression pad representing a number $C = c_0c_1 \ldots c_{d-1}$ in base 2^d is needed.

▶ **Lemma 16.** *Given an integer $d \geq 3$, there exists a staged assembly system with 1 bin, $8d - 1$ tile types, and 1 stage that assembles a width-3 length-d^2 flexible decompression pad.*

Proof. Start with the construction of Lemma 7 that yields a a width-2 length-d^2 decompression pad encoding C. Modify the tile types of this construction such that the leftmost northmost glue is s' and every non-s glue on the north surface is a special *prime* version, to differentiate between other similar glue types. Then add $2d + 1$ tiles that modify the north surface decompression pad to yield width-3 flexible decompression pad, as seen in Figure 8.

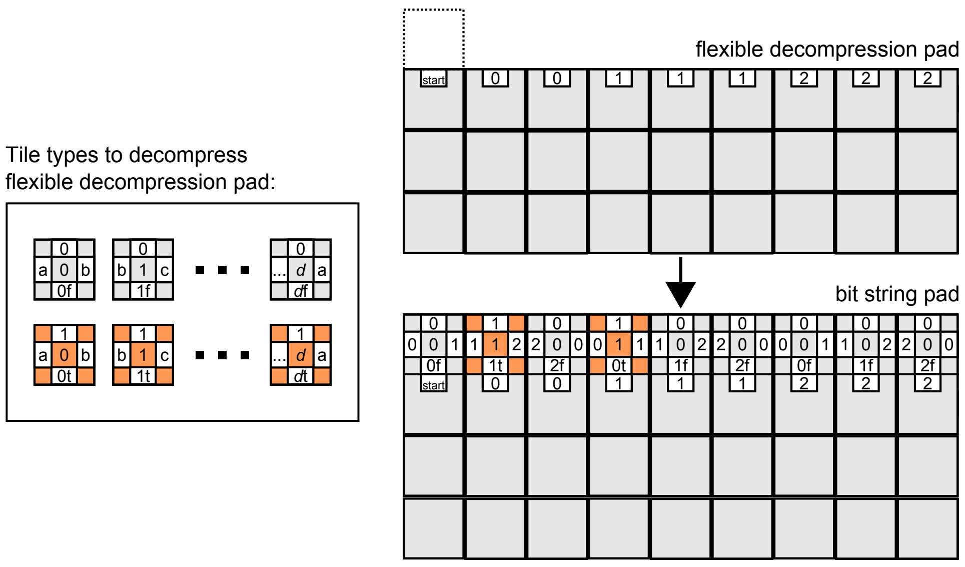


Figure 9 On the left, the templates for the decompression tiles needed to decompress a flexible decompression pad for any given $d \geq 3$. In the top right, an example of a length-9 flexible decompression pad. In the bottom right, the decompression tiles interact with the flexible decompression pad and glue function to assemble a bit string pad from a flexible decompression pad, representing the bit string “010100000”. The flexible glues form a bond of strength 2 between the glue pair (‘start’, ‘0f’), strength 1 between glues pairs (‘0’, ‘0’), (‘1’, ‘1’), (‘2’, ‘2’), (‘0’, ‘1t’), (‘0’, ‘2f’), (‘1’, ‘0t’), (‘1’, ‘1f’), (‘1’, ‘2f’), (‘2’, ‘0f’), (‘2’, ‘1f’), and (‘2’, ‘2f’), and strength 0 between all other glue pairs.

This step requires $5d + \log 2^d - 2$ tile types to build a modified decompression pad and $2d+1$ tiles to convert this modified decompression pad that into a flexible decompression pad. Thus $2d + 1 + 5d + \log 2^d - 2 \leq 8d - 1$ tile types are used in total. ◀

▶ **Lemma 17.** *Given integers $d \geq 3$ and any length d^2 bit string R, there exists a 1 stage, 1 bin, staged assembly system with flexible glues that assembles a width-4 gap-0 d^2-bit string pad representing R, using at most $10d - 1$ tile types.*

Proof. Consider a width-3, length d^2 flexible decompression pad. The idea is to use $2d$ tile types and flexible glues to build a width-4 gap-0 d^2-bit string pad from the flexible decompression pad (see Figure 9). Consider a sequence of d bitstrings $D = D_0, D_1, \ldots, D_{d-1}$ with each $D_i = s_0 s_1 s_2 \ldots s_{d-1}$ such that the in-order concatenation of all bitstrings in D equals R. Let $D_{i,j}$ denote the j^{th} bit of the i^{th} bit string of D.

The goal is to construct a glue function such that it specifies the tiles that can attached to the top of the flexible decompression pad to be the concatenation of the bitstrings in D. Note that the tiles that have a “0” or “1” glue as those with labels that end in “f” or “t”, respectively. Let $str(g_1, g_2)$ denote the strength between glues g_1 and g_2. Set the tile that attaches to the “start” glue to be one that exposes “0” or a “1” by setting $str(start, 0f) = 2$ or $str(start, 0t) = 2$, respectively. For all $D_{i,j}$, we set $str(i, jf) = 0$ if and only if $D_{i,j} = 0$ and $str(i, jt) = 1$, otherwise. In addition, we set $str(a, a) = 1$, $str(b, b) = 1$, and so on.

With this, we build a width-4 gap-0 d^2-bit string pad from the flexible decompression pad. An example of this can be seen in Figure 9. Also, $8d - 1$ tile types are used to build a width-3, length d^2 flexible decompression pad. An additional $2d$ tile types are needed to decompress, using flexible glues, into a width-4 d^2-bit string pad. So the total number of tile types used is $10d - 1$. ◀

▶ **Lemma 18.** *Given $t \in \mathbb{N}$, there exists some constant $c \in \mathbb{N}$, such that for all cases where $t \geq c$, there exists a staged self-assembly system with t tiles which assembles any width-4 $\Theta(t^2)$-bit string pad using 1 stage, 1 bin, and flexible glues.*

Proof. Given t tile types, consider how many bits can be produced using Lemma 17. Let $d = \lfloor \frac{t+1}{10} \rfloor$. Invoke Lemma 17 to build a width-4 d^2-bit string pad with flexible glues using $10d - 1$ tiles. The number of bits produced is $y = d^2 = (\lfloor \frac{t+1}{10} \rfloor)^2 = \Theta(t^2)$. Then by Lemma 17, any width-4 $(\lfloor \frac{t+1}{10} \rfloor)^2$-bit string pad can be build in the flexible glue model using at most t tiles, 1 stage, and 1 bin. The smallest choice of d requires $d = \lfloor \frac{t+1}{10} \rfloor \geq 3$, implying $t \geq 29$. For all cases where $t \geq c$ we have a constant, $c = 29$, where this lemma holds true. ◀

The improvements to Lemmas 17 and 18 allow for a larger bit string pad to be built in Step 2 when compared to standard glues, reducing stage complexity to $\mathcal{O}(\frac{\log\log b}{\log t} + \frac{x-tb-t^2}{b^2})$:

▶ **Lemma 19.** *Given $t, b \in \mathbb{N}$ and a bit string r where $x = |r|$. Then, there exist some constants $c, d \in \mathbb{N}$, such that for all cases where $t > c$ and $b > d$, there is a staged self-assembly system using flexible glues with b bins and t tiles which assembles an x-bit string pad representing r with width 9 and gap $\Theta(\log b)$ using $\mathcal{O}(\frac{x-tb-t^2}{b^2} + \frac{\log\log b}{\log t})$ stages.*

Nearly tight upper and lower bounds for square and general shape construction in the flexible glue model are obtained by replacing the bit string construction of Lemma 3 with Lemma 19, and applying the flexible glue lower bound of Lemma 1:

▶ **Theorem 20.** *For any $b, t, n \in \mathbb{N}$ and constants c_b, c_t such that $b \geq c_b$ and $t \geq c_t$, there exists a staged self-assembly system using flexible glues with b bins and t tile types that uniquely produces an $n \times n$ square using $\mathcal{O}(\frac{\log n - t^2 - tb}{b^2} + \frac{\log\log b}{\log t})$ stages.*

▶ **Theorem 21.** *For any $b, t \in \mathbb{N}$ and almost all $n \in \mathbb{N}$, any staged self-assembly system with flexible glues which uses at most b bins and t tile types that uniquely assembles an $n \times n$ square must use $\Omega(\frac{\log n - t^2 - tb}{b^2})$ stages.*

▶ **Theorem 22.** *For any shape S and $b, t \in \mathbb{N}$ and constants c_b, c_t such that $b \geq c_b$ and $t \geq c_t$, there exists a staged self-assembly system using flexible glues with b bins and t tile types which uniquely produces S at some scale factor using $\mathcal{O}(\frac{K(S) - t^2 - tb}{b^2} + \frac{\log\log b}{\log t})$ stages.*

▶ **Theorem 23.** *For any $b, t \in \mathbb{N}$ and shape S with Kolmogorov complexity $K(S)$, any staged self-assembly system with flexible glues which uses at most b bins and t tile types that uniquely assembles S must use $\Omega(\frac{K(S) - t^2 - tb}{b^2})$ stages.*

8 Conclusion

In this work, we achieved nearly optimal staged assembly of two classic benchmark shape classes. These constructions generalize the known upper bounds of [2, 6, 8, 20] to arbitrary choices of tile type and bin counts, as well as to the flexible glue model. The natural problem left open is the elimination of the additive $\mathcal{O}(\frac{\log\log b}{\log t})$ gap between the upper and lower bounds induced by the wings subconstruction of Section 4.1. Although this subconstruction is the cause of an additive gap in an otherwise optimal result, it is a useful approach for general assembly labeling and coordinated attachment and is likely useful in other staged constructions. The constant width of our bit string pads can also potentially be exploited for efficient construction of shapes with geometric bottlenecks, e.g., thin rectangles.

References

1 Zachary Abel, Nadia Benbernou, Mirela Damian, Erik Demaine, Martin Demaine, Robin Flatland, Scott Kominers, and Robert Schweller. Shape replication through self-assembly and RNAse enzymes. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2010.

2 Leonard Adleman, Qi Cheng, Ashish Goel, and Ming-Deh Huang. Running time and program size for self-assembled squares. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 740–748, 2001.

3 Bahar Behsaz, Ján Maňuch, and Ladislav Stacho. Turing universality of step-wise and stage assembly at temperature 1. In *DNA Computing and Molecular Programming (DNA)*, volume 7433 of *LNCS*, pages 1–11. Springer, 2012.

4 Sarah Cannon, Erik D. Demaine, Martin L. Demaine, Sarah Eisenstat, Matthew J. Patitz, Robert Schweller, Scott M. Summers, and Andrew Winslow. Two hands are better than one (up to constant factors): Self-assembly in the 2HAM vs. aTAM. In *Proceedings of 30th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 20 of *LIPIcs*, pages 172–184. Schloss Dagstuhl, 2013.

5 Ho-Lin Chen and David Doty. Parallelism and time in hierarchical self-assembly. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1163–1182, 2012.

6 Qi Cheng, Gagan Aggarwal, Michael H. Goldwasser, Ming-Yang Kao, Robert T. Schweller, and Pablo Moisset de Espanés. Complexities for generalized models of self-assembly. *SIAM Journal on Computing*, 34:1493–1515, 2005.

7 E. D. Demaine, M. J. Patitz, T. A. Rogers, R. T. Schweller, and D. Woods. The two-handed tile assembly model is not intrinsically universal. In *Automata, Languages and Programming (ICALP)*, volume 7965 of *LNCS*, pages 400–412. Springer, 2013.

8 Erik D. Demaine, Martin L. Demaine, Sándor P. Fekete, Mashhood Ishaque, Eynat Rafalin, Robert T. Schweller, and Diane L. Souvaine. Staged self-assembly: nanomanufacture of arbitrary shapes with $O(1)$ glues. *Natural Computing*, 7(3):347–370, 2008.

9 Erik D. Demaine, Sarah Eisenstat, Mashhood Ishaque, and Andrew Winslow. One-dimensional staged self-assembly. *Natural Computing*, 12(2):247–258, 2013.

10 Erik D. Demaine, Sándor P. Fekete, Christian Scheffer, and Arne Schmidt. New geometric algorithms for fully connected staged self-assembly. In *DNA Computing and Molecular Programming (DNA)*, volume 9211 of *LNCS*, pages 104–116. Springer, 2015.

11 Erik D. Demaine, Matthew J. Patitz, Robert T. Schweller, and Scott M. Summers. Self-assembly of arbitrary shapes using RNAse enzymes: Meeting the Kolmogorov bound with small scale factor (extended abstract). In *Proceedings of the 28th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 9 of *LIPIcs*, pages 201–212. Schloss Dagstuhl, 2011.

12 David Doty. Producibility in hierarchical self-assembly. In *Proceedings of Unconventional Computation and Natural Computation (UCNC) 2014*, pages 142–154, 2014.

13 Constantine Evans. *Crystals that Count! Physical Principles and Experimental Investigations of DNA Tile Self-Assembly.* PhD thesis, Caltech, 2014.

14 David Furcy, Samuel Micka, and Scott M. Summers. Optimal program-size complexity for self-assembly at temperature 1 in 3D. In *DNA Computing and Molecular Programming (DNA)*, volume 9211 of *LNCS*, pages 71–86. Springer, 2015.

15 Yonggang Ke, Luvena L. Ong, William M. Shih, and Peng Yin. Three-dimensional structures self-assembled from dna bricks. *Science*, 338(6111):1177–1183, 2012.

16 Thomas H. Labean, Sung Ha Park, Sang Jung Ahn, and John H. Reif. Stepwise DNA self-assembly of fixed-size nanostructures. In *Foundations of Nanoscience, Self-assembled Architectures, and Devices*, pages 179–181, 2005.

17 Ján Maňuch, Ladislav Stacho, and Christine Stoll. Step-wise tile assembly with a constant number of tile types. *Natural Computing*, 11(3):535–550, 2012.

18 Matthew J. Patitz and Scott M. Summers. Identifying shapes using self-assembly. *Algorithmica*, 64:481–510, 2012.

19 Paul W. K. Rothemund and Erik Winfree. The program-size complexity of self-assembled squares (extended abstract). In *Proceedings of the 32nd ACM Symposium on Theory of Computing (STOC)*, pages 459–468, 2000.

20 David Soloveichik and Erik Winfree. Complexity of self-assembled shapes. *SIAM Journal on Computing*, 36(6):1544–1569, 2007.

21 Erik Winfree. *Algorithmic Self-Assembly of DNA*. PhD thesis, Caltech, 1998.

22 Andrew Winslow. Staged self-assembly and polyomino context-free grammars. *Natural Computing*, 14(2):293–302, 2015.

Homotopy Measures for Representative Trajectories

Erin Chambers[*1], **Irina Kostitsyna**[†2], **Maarten Löffler**[‡3], **and Frank Staals**[§4]

1 **Dept. of Math and Computer Science, Saint Louis University, Saint Louis, USA**
`echambe5@slu.edu`
2 **Dept. of Mathematics and Computer Science, TU Eindhoven, Eindhoven, The Netherlands**
`i.kostitsyna@tue.nl`
3 **Dept. of Computing and Information Sciences, Utrecht University, The Netherlands**
`m.loffler@uu.nl`
4 **MADALGO, Aarhus University, Aarhus, Denmark**
`f.staals@cs.au.dk`

Abstract

An important task in trajectory analysis is defining a meaningful representative for a cluster of similar trajectories. Formally defining and computing such a representative r is a challenging problem. We propose and discuss two new definitions, both of which use only the geometry of the input trajectories. The definitions are based on the homotopy area as a measure of similarity between two curves, which is a minimum area swept by all possible deformations of one curve into the other. In the first definition we wish to minimize the maximum homotopy area between r and any input trajectory, whereas in the second definition we wish to minimize the sum of the homotopy areas between r and the input trajectories. For both definitions computing an optimal representative is NP-hard. However, for the case of minimizing the sum of the homotopy areas, an optimal representative can be found efficiently in a natural class of restricted inputs, namely, when the arrangement of trajectories forms a directed acyclic graph.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

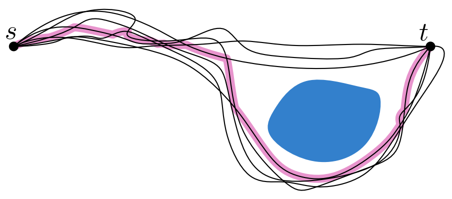
Keywords and phrases trajectory analysis, representative trajectory, homotopy area

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.27

1 Introduction

Motivated by GIS applications, the question of extracting a meaningful representative trajectory from a collection of similar trajectories has recently received considerable attention in the computational geometry community [3, 11, 12, 16, 19, 1, 10, 2]. In many trajectory analysis applications, only the locations (and not the corresponding time stamps) in the

[*] E.C. is supported by the National Science Foundation under grant CCF-1054779 and IIS-1319573.
[†] I.K. is (partially) supported by the Netherlands Organisation for Scientific Research (NWO) under grant 639.023.208 and 639.021.123, respectively.
[‡] M.L. is (partially) supported by the Netherlands Organisation for Scientific Research (NWO) under grant 639.023.208 and 639.021.123, respectively.
[§] F.S. is supported by the Danish National Research Foundation under grant nr. DNRF84.



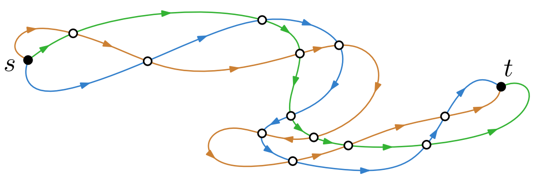


Figure 1 Left: Curves representing hiking trajectories between two points, and a possible representative. A pointwise average trajectory would go through the lake. Right: The trajectory graph G. In this example, G is acyclic.

trajectories are relevant. Consider, for example, the case in the input trajectories originate from hikers that walked a similar trail, but possibly on different days. In such a setting the trajectories are just curves in the plane, and hence we wish to find a representative curve that captures important features shared by most of the input curves. It has been argued before that it is desirable that the representative uses only pieces of the input trajectories, so that it avoids obstacles in the underlying space [4]. See for example Fig. 1 (left). Hence, we will restrict our attention to representatives that consist of pieces of the input trajectories, and that ignore any temporal information available.

Buchin et al. investigate whether a reasonable notion of a median exists in such a setting that depends only on the intersections in a set of trajectories [4]. Their *simple median* is essentially not using the geometry. They also present a second definition, that incorporates a notion of the topology of the underlying space, by placing obstacles in large open regions and restricting the class of trajectories to the same *homotopy type*, that is, they require the representative trajectory to "wind around" the obstacles in the same way as the majority of the input trajectories. For example, in Fig. 1 (left), a user could for example mark the lake as an obstacle. Buchin et al. conclude that while computation of the median is possible to some extent, some notion of geometry and topology seems necessary to handle practical situations.

In this paper, we include some geometric and topological information in the selection of a representative trajectory (curve), namely, the area of the faces in the arrangement of trajectories. As a measure of similarity between two curves, we use the *homotopy area* from Chambers and Wang [7], which is the minimum area swept by a deformation of one non-self-intersecting curve into the other. More formally, if $H : [0,1] \times [0,1] \to \mathbb{R}^2$ is a continuous deformation of curve μ into τ, the homotopy area of H is

$$A(H) = \int_{s\in[0,1]} \int_{t\in[0,1]} \left| \frac{\mathrm{d}H}{\mathrm{d}s} \times \frac{\mathrm{d}H}{\mathrm{d}t} \right| \mathrm{d}s\, \mathrm{d}t.$$

The infimum $\mathrm{HA}(\mu, \tau)$ of A over all continuous deformations between μ and τ is the homotopy area. The notion of homotopy area seems particularly attractive in our setting as it implicitly penalizes a representative trajectory for deviating from the bulk of the trajectories without making it necessary to artificially place obstacles in the ambient space, which was the solution used in prior work [4]. Homotopy area is defined only on curves which are non-self-intersecting, so we must also place this constraint on each of our input trajectories.

Problem Statement. We are given a set of *trajectories* $\mathcal{T} = \{\tau_1, .., \tau_n\}$, which are piecewise curves, each piece of low algebraic degree, in the plane. We wish to compute a single trajectory μ^* that best represents all trajectories in $\mathcal{T}$. As we will use homotopy area to measure the quality of μ^* we require that each individual trajectory τ_i is *simple*, that is, it has no self-intersections (otherwise homotopy area is not well defined). Consider the arrangement of the trajectories $\mathcal{T}$ in $\mathbb{R}^2$, and orient each edge so that its direction corresponds to that of the trajectory defining it. We refer to this arrangement as the *trajectory graph* G. See Fig. 1.

Initially, we will assume that all trajectories start and end at the same points, say s and t respectively, and that s and t lie in the outer face of the arrangement of the trajectories. We will (partially) lift these restrictions in Sections 4.2 and 4.3.

For the output trajectory μ^* we require that it is a simple path in the trajectory graph; this means that it can consist only of segments of the input trajectories, that it is simple, and that it uses each segment in the same direction as used in the input trajectory.

Among all possible output trajectories (simple paths in G), we wish to construct one that represents $\mathcal{T}$ best. We measure this by the distance between the (candidate) median μ and the trajectories in $\mathcal{T}$. Let $\text{HA}(\mu, \tau_i)$ be the minimum homotopy area between μ and a trajectory $\tau_i \in \mathcal{T}$. We consider two variants: minimizing the maximum distance $\text{HA}_{\text{max}}(\mu, \mathcal{T}) = \max_{\tau_i \in \mathcal{T}} \text{HA}(\mu, \tau_i)$ between μ and the trajectories in $\mathcal{T}$, and the sum of the distances $\text{HA}_{\text{sum}}(\mu, \mathcal{T}) = \sum_{\tau_i \in \mathcal{T}} \text{HA}(\mu, \tau_i)$ between μ and the trajectories in $\mathcal{T}$. If $\mathcal{T}$ is clear from the context we will write $\text{HA}_{\text{max}}(\mu) = \text{HA}_{\text{max}}(\mu, \mathcal{T})$ and $\text{HA}_{\text{sum}}(\mu) = \text{HA}_{\text{sum}}(\mu, \mathcal{T})$.

Results. We show that the first variant considered, minimizing the maximum distance, is NP-hard, even if we have only two trajectories, both of which are x-monotone (Section 2). In general, minimizing the sum of the distances, HA_{sum}, is also NP-hard, as we show in Section 3. However, the second hardness reduction is more involved and critically relies on cycles in the trajectory graph. If the trajectory graph is a directed acyclic graph (DAG), then we can compute a representative minimizing HA_{sum} efficiently, as we show in Section 4. Quite surprisingly, our results show that when the graph is a DAG and all trajectories share a start and end point on the outer face, the simple median from Buchin et al. [4] that does not incorporate areas in any way, remains the optimal choice for minimizing HA_{sum}. Hence, even though the best running time to compute homotopy area between two curves is $O(n + I^2 \log n)$ time, where n is the complexity of the input curves and I is the number of intersections between the two curves [7], we are able to calculate a median trajectory under homotopy area much more quickly using the simple median algorithm [4]. We also show that our approach generalizes to the case when the start and end points of the trajectories are in different, arbitrary faces of the DAG, although the simple median is no longer the curve minimizing HA_{sum}. Instead, a simple median must be computed between lifts of the trajectories in a particular covering space of the plane. Omitted proofs are in the full version.

2 Minimizing the Maximum Distance HA_{max} is NP-hard

In this section we consider computing a representative that minimizes the maximum distance to all other trajectories. Unfortunately, this problem is NP-hard, even for the case of a constant number of x-monotone input curves.

▶ **Theorem 1.** *Given a set of trajectories $\mathcal{T}$, computing a median μ that minimizes HA_{max} is NP-hard, even if $\mathcal{T}$ contains only two trajectories, both of which are x-monotone.*

Proof. We reduce from the Partition problem, which, given a set $A = \{a_1, .., a_n\}$ of positive integers, asks for a partition of A into sets A_1 and A_2 such that $\sum_{a_i \in A_1} a_i = \sum_{a_i \in A_2} a_i = \sum_{a_i \in A} a_i/2$. Given the set A, we construct two x-monotone trajectories τ_1 and τ_2 such that the faces between successive intersections have area equal to some $a_i \in A$. See Fig. 2.

Any candidate trajectory μ corresponds to a partition of A into A_1 and A_2: $a_i \in A_1$ if and only if μ uses the piece of τ_1 that bounds the face corresponding to a_i. It

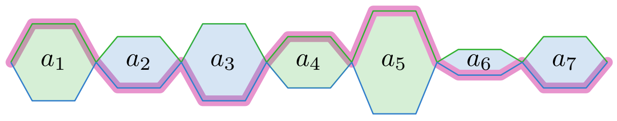


Figure 2 An illustration of the NP-hardness reduction from Partition. The purple curve represents the partition $B = \{a_2, a_3, a_6, a_7\}$ and $G = \{a_1, a_4, a_5\}$.

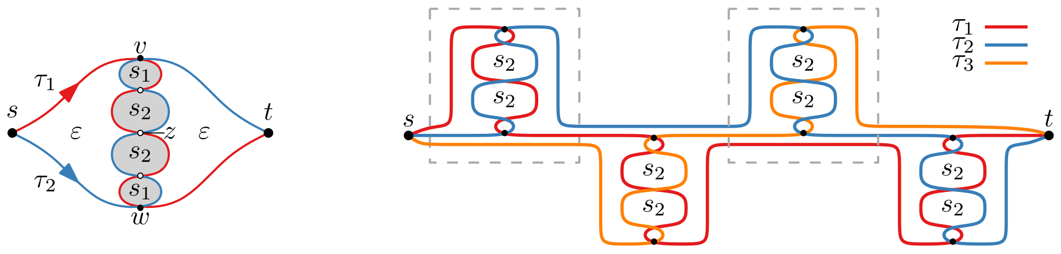


Figure 3 Left: The *braid* construction—a basic building block for hardness proof gadgets. Optimal representative trajectory does not switch at intersections. $\mathrm{HA}_{\mathrm{sum}} = 4s_1 + 2\varepsilon$. Right: Four building blocks joint together. Optimal representative trajectory follows the red or the blue trajectory all the way from s to t and does not switch at intersections. $\mathrm{HA}_{\mathrm{sum}} = 5s_2 + O(s_1)$.

follows that the homotopy area between μ and τ_j is exactly $\sum_{a_i \in A_j} a_i$. Thus $\mathrm{HA}_{\max}(\mu) = \max\left\{\sum_{a_i \in A_1} a_i, \sum_{a_i \in A_2} a_i\right\}$. Let μ^* be a trajectory minimizing $\mathrm{HA}_{\max}$. We have that $\mathrm{HA}_{\max}(\mu^*) = \sum_{a_i \in A} a_i/2$ if and only if A can be partitioned such that $\sum_{a_i \in A_1} a_i = \sum_{a_i \in A_2} a_i$. Therefore minimizing $\mathrm{HA}_{\max}$ is (weakly) NP-hard. ◀

3 Minimizing the Sum of Distances $\mathrm{HA}_{\mathrm{sum}}$ is NP-hard

In this section we show that minimizing the total sum of the distances from the representative trajectory to all the trajectories in $\mathcal{T}$ is NP-hard in general.

Before we describe the gadgets for variables and clauses, consider the two trajectories τ_1 and τ_2 in Fig. 3 (left). Let $\varepsilon \ll s_1 \ll s_2$ be the areas swept by the deformation of τ_1 into τ_2 between the intersection points, and let μ^* be a representative trajectory that minimizes $\mathrm{HA}_{\mathrm{sum}}$. We will call this construction a *braid* of τ_1 and τ_2. We will show that in a braid $\mu^* = \tau_1$ or $\mu^* = \tau_2$, i.e., μ^* does not switch to another trajectory at any intersection point.

▶ **Lemma 2.** *If the areas of the faces of the arrangement of $\mathcal{T} = \{\tau_1, \tau_2\}$, for a braid construction of two trajectories τ_1 and τ_2 (depicted in Fig. 3 (left)), satisfy $\varepsilon \ll s_1 \ll s_2$, then the optimal representative trajectory $\mu^* = \tau_1$ or $\mu^* = \tau_2$.*

The braid construction is a crossing gadget, it allows two trajectories to cross while enforcing that μ^* does not switch to another trajectory at intersections. We will use it as a basic building element in the hardness proof gadgets.

Now consider an arrangement of three trajectories in Fig. 3 (right). There are four braids of pairs of trajectories used in this arrangement. The red trajectory, τ_1, and the blue trajectory, τ_2, are rotationally symmetrical. Let μ^* be a representative trajectory that minimizes $\mathrm{HA}_{\mathrm{sum}}$.

▶ **Lemma 3.** *For the arrangement of three trajectories $\mathcal{T} = \{\tau_1, \tau_2, \tau_3\}$ depicted in Fig. 3 (right) the optimal representative trajectory $\mu^* = \tau_1$ or $\mu^* = \tau_2$.*

▶ **Theorem 4.** *Minimizing HA_{sum} is NP-hard.*

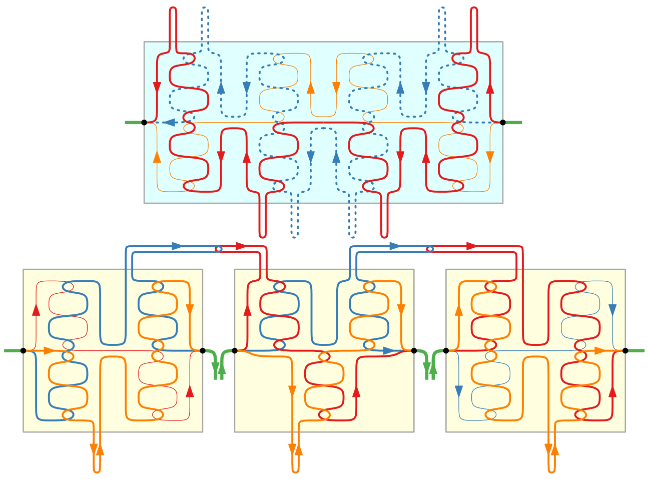


Figure 4 Left: An example of a variable gadget consisting of two building blocks. Variable gadgets are traversed by trajectories from right to left. Right: The clause gadget is traversed from left to right.

Proof. We prove that it is NP-hard to minimize HA_{sum} by a reduction from planar 3-SAT [17]: given an instance of a planar 3-SAT formula Φ with n variables and m clauses, and a rectilinear embedding[1] of its graph [15], we construct a set of three trajectories $\mathcal{T}$ such that minimizing HA_{sum} for $\mathcal{T}$ is equivalent to answering the question if Φ is satisfiable.

Variable gadget. The variable gadget (refer to Fig. 4 (left)) consists of a series of building blocks from Fig. 3 (right) with red and blue trajectories having two thin extensions (such that the area covered by them is $O(\varepsilon)$) that will serve as connectors to clauses. Up until entering the variable gadget all three trajectories follow the same path (shown in green in the figure), and they diverge after entering into the gadget. Selecting the red or the blue trajectory for the optimal representative trajectory μ^* to follow at this moment corresponds to setting the variable to true or false. As the variable gadget consists of building blocks that prevent μ^* from switching the color, the next color change can only occur once μ^* exits the variable gadget. One block of the variable gadget contributes $5s_2 + O(s_1)$ area to the total homotopy area between μ^* and the three trajectories.

Clause gadget. The clause gadget (shown in Fig. 4 (right)) consists of three blocks that will be connected to the corresponding variable gadgets: the leftmost and the rightmost blocks are the same as in Fig. 3 (right) (up to change of colors), and the middle block is a similar construction but consists of only three braids from Fig. 3 (left). The green trajectories in-between the blocks represent all three trajectories (red, blue, and yellow) following the same path that connect the current clause to the other clauses in hierarchical order (for more details refer to the next paragraph). The first block allows μ^* to follow the blue or the orange trajectory. In the second block all three trajectories will contribute the same amount to the area measure, thus any of the three trajectories can be chosen by μ^*. The third block allows μ^* to follow the red or the orange trajectory. Moreover, μ^* cannot choose

[1] Recall that in a rectilinear embedding of a graph of a 3-SAT formula, the variable-vertices are placed on a horizontal line, and the clause-vertices are placed above and below the horizontal line and connected to the corresponding variable-vertices with axis-aligned L-shape connectors.

the blue trajectory in the first block and the red trajectory in the second block at the same time, as this would cause a self-intersection. Similarly, μ^* cannot choose the blue trajectory in the second block and the red trajectory in the third block at the same time. Thus, μ^* has to choose the orange trajectory in at least one of the three blocks. Choosing the orange trajectory corresponds to satisfying the clause with the value of the corresponding variable. A clause gadget contributes $5s_2 + O(s_1) + 4s_2 + O(s_1) + 5s_2 + O(s_1) = 14s_2 + O(s_1)$ area to the total homotopy area between μ^* and the three trajectories.

Fig. 5 (left) shows an example of a clause $(\neg x \vee y \vee z)$ connected to the three corresponding variables. It is depicting the case when the clause is satisfied by setting the value of y to true.

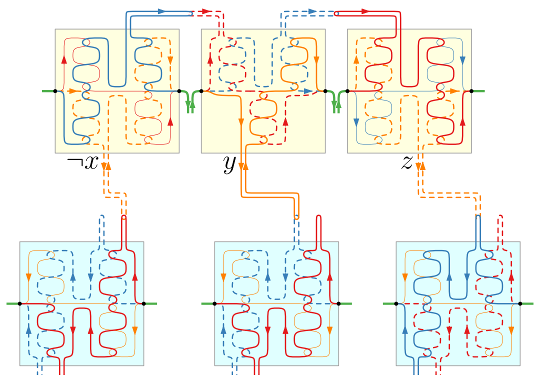
Putting all the building blocks together. Given the rectilinear embedding of the graph of the planar 3-SAT formula, we construct the gadgets for the variables and the clauses. The embedding provides a hierarchy of the clauses that leads to a natural order in which the clauses can be traversed (refer to Fig. 5 (right)). The outermost clause gets traversed the first; the clauses that lie between the first two legs of the outermost clause get traversed after the first block and before the second block of that clause; analogously, the clauses that lie between the second and the third legs of the outermost clause get traversed after the second block and before the third block of that clause; sibling clauses that lie in the same level get traversed one after another. Thus, the three trajectories will start at the top left of the embedding, traversing all the clauses that lie above the horizontal line containing the variables, then they traverse all the variables in order of appearance on the horizontal line, and then they traverse the clauses that lie below the variable line. If formula Φ is satisfiable, the total homotopy area of μ^* is

$$\mathrm{HA}_{\mathrm{sum}}(\mu^*, \mathcal{T}) = (\sum_{1}^{n} 5k_i + 14m)s_2 + (\sum_{1}^{n} 5k_i + 14m)O(s_1) \leq 29ms_2 + 29mO(s_1)\,,$$

where k_i is the number of blocks in the variable x_i's gadget, and since some of the blocks can be connected to multiple clauses, $\sum_1^n k_i \leq 3m$. Let $s_2 = 1$, and $s_1 = o(\frac{1}{m})$. If μ^* switches a trajectory at any intersection point inside of any gadget, the total area added as a penalty to $\mathrm{HA}_{\mathrm{sum}}$ shall be $\gg 29m$. This can be easily achieved by increasing the space between the gadgets. Therefore, deciding if there exists a μ^* such that the total homotopy area $\mathrm{HA}_{\mathrm{sum}}(\mu^*, \mathcal{T})$ is not greater than $29m + o(1)$, is equivalent to deciding if Φ is satisfiable. The size of the construction is polynomial in size of the 3-SAT instance, therefore, it is NP-hard to find a representative trajectory that minimizes $\mathrm{HA}_{\mathrm{sum}}$. ◀

4 Minimizing the Sum of Distances $\mathrm{HA}_{\mathrm{sum}}$ when G is a DAG

We now describe how to compute a representative trajectory that minimizes $\mathrm{HA}_{\mathrm{sum}}$ for a set of trajectories $\mathcal{T}$ whose trajectory graph is acyclic. For simplicity of presentation, we assume that n is odd. All our proofs can be extended to the case when n is even. As a warmup, we consider the case in which the trajectories in $\mathcal{T}$ are x-monotone. Next, we expand to the case when s and t lie on the boundary of the outer face of G but the trajectories are no longer required to be x-monotone. Finally, we consider the most general case, when s and t lie in the interior faces of G.



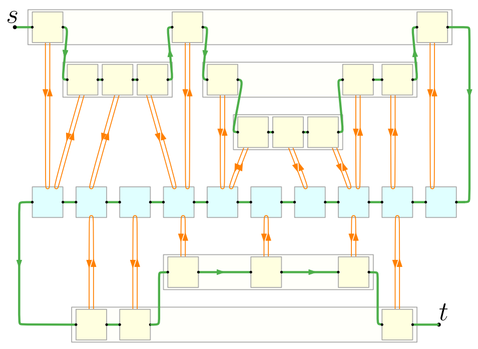


Figure 5 Left: An example of clause $(\neg x \vee y \vee z)$ connected to the three corresponding variables. Here, $x =$ true, $y =$ true and $z =$ false. Right: An order of traversal of the clause and variable gadgets is induced by the embedding of the planar graph of Φ.

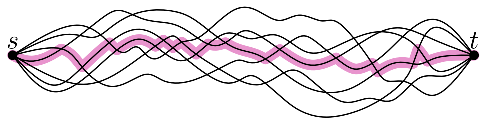


Figure 6 The simple median for a set of x-monotone trajectories.

4.1 Minimizing $\mathsf{HA}_{\mathsf{sum}}$ for x-Monotone Trajectories

In this section we will show that for x-monotone trajectories, the *simple median*, as defined by Buchin et al. [4], minimizes the sum of the homotopy areas $\mathrm{HA}_{\mathrm{sum}}$. At the starting point s, the simple median starts at the $\lceil n/2 \rceil^{\text{th}}$ curve (ranking the trajectories by their y-coordinate just after s). It switches to the other trajectory at every intersection point it encounters, thus staying on the $\lceil n/2 \rceil^{\text{th}}$ trajectory. So, for x-monotone trajectories the simple median corresponds to the $\lceil n/2 \rceil$-level in G. See Fig. 6.

To show that the simple median μ^* minimizes $\mathrm{HA}_{\mathrm{sum}}$ we now write $\mathrm{HA}_{\mathrm{sum}}(\mu)$ as an integral $\int f(x)\,\mathrm{d}x$. The value $f(x)$ represents the sum of the lengths of a set of intervals along a vertical line with abscissa x. All intervals share a common endpoint $\mu(x)$. The total length of these intervals is minimal when μ has the same number of trajectories above and below it, that is, when it coincides with the simple median.

▶ **Lemma 5.** *The simple median μ^* minimizes* $F(\mu) = \displaystyle\int_x \sum_{\tau_i \in \mathcal{T}} |\mu(x) - \tau_i(x)|\,\mathrm{d}x$.

Proof. Let $y_1,..,y_n$ denote the (y-coordinates of the) intersection points of the trajectories with a vertical line ℓ with abscissa x. Any valid representative trajectory uses one of the points y_i, i.e., $\mu(x) \in \{y_1,..,y_n\}$. Note, that the median point $y_{\lceil n/2 \rceil}$ minimizes $f(y) = \sum_{i=1}^n |y - y_i|$. The simple median μ^* is on the $\lceil n/2 \rceil^{\text{th}}$ trajectory at any coordinate x. Therefore, μ^* minimizes $\int_x f(y)\,\mathrm{d}x = \int_x \sum_{i=1}^n |\mu(x) - \tau_i(x)|$. ◀

▶ Remark. When n is even, there are two points $y_{\frac{n}{2}}$ and $y_{\frac{n}{2}+1}$ that minimize $f(y) = \sum_{i=1}^n |y - y_i|$. Therefore, any trajectory switching between the levels $\frac{n}{2}$ and $(\frac{n}{2}+1)$ will minimize $F(\mu) = \int_x \sum_{\tau_i \in \mathcal{T}} |\mu(x) - \tau_i(x)|\,\mathrm{d}x$.

Given a point p let $\omega(p, \delta)$ denote the winding number of p with respect to an oriented closed curve δ. We say that δ is *atomic* if $\omega(p, \delta)$ is either all non-negative, or all non-positive,

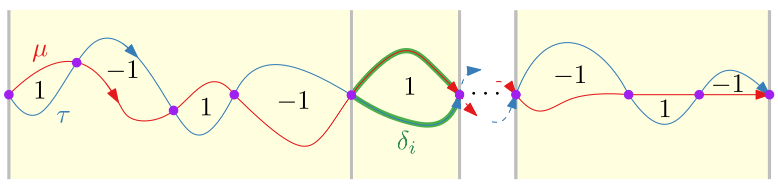


Figure 7 There is a subsequence $p_1,..,p_\ell$ of intersection vertices (purple) that partition μ and τ (in red and blue, respectively) into subcurves $\delta_1,..,\delta_k$, such that all faces in $\delta_i = \mathit{loop}(\mu_i, \tau_i)$ (green) have either winding number one or minus one.

for all points $p \in \mathbb{R}^2$. Furthermore, let $W(\delta) = \int_{p\in\mathbb{R}^2} \omega(p,\delta)\,\mathrm{d}p$ denote the *total winding number* of curve δ.

Let μ and τ be two curves from s to t, let $\delta = \mathit{loop}(\mu,\tau)$ denote the closed curve obtained by concatenating μ and the reverse of τ, and let $s = p_1,..,p_\ell = t$ denote the intersection points between μ and τ, ordered along μ. Chambers and Wang [7] show that there is a (not necessarily contiguous) subsequence of the intersection points $\{p_i\}$ that decompose δ into a set of atomic closed curves $\Delta(\mu,\tau) = \delta_1,..,\delta_k$, such that the minimum homotopy area $\mathrm{HA}(\mu,\tau) = \sum_{i=1}^{k} |W(\delta_i)|$. See Fig. 7 for an illustration.

▶ **Observation 6.** *If μ and τ are x-monotone curves, the atomic curves in $\Delta(\mu,\tau)$ are pairwise disjoint (except for the subsequence of the intersection points $\{p_i\}$).*

▶ **Theorem 7.** *Let $\mathcal{T}$ be a set of x-monotone trajectories. The simple median μ^* minimizes HA_{sum}.*

Proof. We will show that $\mathrm{HA}_{\mathrm{sum}}(\mu) = F(\mu)$. The theorem then follows from Lemma 5. Using the result of Chambers and Wang [7] we can then rewrite $\mathrm{HA}_{\mathrm{sum}}(\mu)$ as

$$\begin{aligned}
\mathrm{HA}_{\mathrm{sum}}(\mu) &= \sum_{\tau_i\in\mathcal{T}} \mathrm{HA}(\mu,\tau_i) = \sum_{\tau_i\in\mathcal{T}} \sum_{\delta\in\Delta(\mu,\tau_i)} |W(\delta)| = \\
&= \sum_{\tau_i\in\mathcal{T}} \sum_{\delta\in\Delta(\mu,\tau_i)} \left| \int_{p\in\mathbb{R}^2} \omega(p,\delta)\,\mathrm{d}p \right| = \\
&= \sum_{\tau_i\in\mathcal{T}} \sum_{\delta\in\Delta(\mu,\tau_i)} \int_{x\in\mathbb{R}} \int_{y\in\mathbb{R}} |\omega((x,y),\delta)|\,\mathrm{d}y\,\mathrm{d}x = \\
&= \int_{x\in\mathbb{R}} \sum_{\tau_i\in\mathcal{T}} \sum_{\delta\in\Delta(\mu,\tau_i)} \int_{y\in\mathbb{R}} |\omega((x,y),\delta)|\,\mathrm{d}y\,\mathrm{d}x\,.
\end{aligned}$$

A vertical line ℓ_x with x-coordinate x intersects (the faces of) G in a set of intervals $\mathcal{I}(x) = I_1,..,I_n$. So, for any curve δ that uses only edges of G, all points (values) in an interval I_i have the same winding number $\omega(I_i,\delta)$. So,

$$\begin{aligned}
\mathrm{HA}_{\mathrm{sum}}(\mu) &= \int_{x\in\mathbb{R}} \sum_{\tau_i\in\mathcal{T}} \sum_{\delta\in\Delta(\mu,\tau_i)} \sum_{I\in\mathcal{I}(x)} \int_{y\in I} |\omega((x,y),\delta)|\,\mathrm{d}y\,\mathrm{d}x = \\
&= \int_{x\in\mathbb{R}} \sum_{\tau_i\in\mathcal{T}} \sum_{\delta\in\Delta(\mu,\tau_i)} \sum_{I\in\mathcal{I}(x)} |\omega(I,\delta)| \cdot |I|\,\mathrm{d}x\,.
\end{aligned}$$

Since the trajectories are x-monotone, so is μ. The curves $\delta \in \Delta(\mu,\tau_i)$ are built by concatenating a piece of μ and a reversed piece of τ_i. Hence, any vertical line ℓ_x intersects δ in exactly two points: $\mu(x)$ and $\tau_i(x)$. Therefore, any point p on ℓ_x that lies in between these points has winding number one or minus one with respect to δ. Any point outside the interval defined by $\mu(x)$ and $\tau_i(x)$ has winding number zero. Thus, we get

$$\mathrm{HA}_{\mathrm{sum}}(\mu) = \int_{x\in\mathbb{R}} \sum_{\tau_i\in\mathcal{T}} \sum_{\delta\in\Delta(\mu,\tau_i)} |\mu(x) - \tau_i(x)|\,\mathrm{d}x\,.$$

Since μ and all τ_i are x-monotone, Observation 6 gives us that all curves in $\Delta(\mu,\tau_i)$ are pairwise disjoint. This gives us

$$\mathrm{HA}_{\mathrm{sum}}(\mu) = \int_{x\in\mathbb{R}} \sum_{\tau_i\in\mathcal{T}} |\mu(x) - \tau_i(x)|\,\mathrm{d}x = F(\mu)\,. \qquad \blacktriangleleft$$

4.2 Extending to Acyclic G with s and t on the Outer Face

The proof from the previous section consists of two steps: (i) show that the simple median minimizes the function F, which represents the sum of interval lengths along a sweep-line; and (ii) show that minimizing the sum of interval lengths along this sweep-line is equivalent to minimizing $\mathrm{HA}_{\mathrm{sum}}$. The two key ideas to extend the algorithm to the case in which the trajectory graph is a DAG that has s and t on the outer face (but is otherwise unconstrained), are that (a) we can generalize (i) to minimizing curve-intervals lengths along a sweep-curve, and, (b) a suitable sweep-curve exists for which minimizing the sum of curve-intervals lengths is again the same as minimizing $\mathrm{HA}_{\mathrm{sum}}$.

We say that a curve is *conforming* to trajectories $\mathcal{T}$ if and only if it is simple and intersects all trajectories of $\mathcal{T}$ exactly once. Let $\gamma : \mathbb{R}\times\mathbb{R} \to \mathbb{R}^2$ be a continuous map such that for any $u \in [0,1]$, $\gamma(u) = \bigcup_{z\in\mathbb{R}} \gamma(u,z)$ is an (open) conforming curve that separates s and t, such that for any u, $\gamma(u,-\infty)_y = -\infty$ and $\gamma(u,+\infty)_y = +\infty$, and for any $u \notin [0,1]$, $\gamma(u) = \bigcup_{z\in\mathbb{R}} \gamma(u,z)$ is an open curve that does not intersect $\mathcal{T}$. We say that γ is a (conforming) sweep-curve. Assume, without loss of generality, that s lies to the left of $\gamma(u)$ and t to the right of $\gamma(u)$ for all $u \in (0,1)$.

Let $c(\gamma,u,i)$ denote the i^{th} intersection point of $\gamma(u)$ with a trajectory in $\mathcal{T}$, and let μ_γ be the curve that for any value u corresponds to the $\lceil n/2\rceil^{\mathrm{th}}$ intersection point on $\gamma(u)$, i.e. $\mu_\gamma(u) = c(\gamma,u,\lceil n/2\rceil)$. Note that μ_γ is simply connected.

▶ **Lemma 8.** *Let φ_0 and φ_1 be conforming curves. Furthermore, assume that the only point from $\bigcup\mathcal{T}$ to the left of φ_1 is s. There is a conforming sweep-curve γ that deforms $\gamma(0) = \varphi_0$ into $\gamma(1) = \varphi_1$.*

Proof. Let $k(\varphi)$ denote the number of vertices of the trajectory graph G that lie to the left the conforming curve φ. We have $k(\varphi_1) = 1$, and $k(\varphi_0) = m+1$, for some $m \in \mathbb{N}$. We now prove by induction on m that we can continuously deform φ_0 into φ_1 while remaining conforming. The lemma then follows.

The base case $m = 0$ is trivial, because two conforming curves without vertices of G between them must intersect exactly the same edges in exactly the same order. Hence, such curves are actually combinatorially equivalent.

For the induction step, let $V = v_0, v_1, .., v_z$ denote the vertices of G in topological order, let L be the set of vertices left of φ_0, and let v_ℓ be the last vertex (with respect to order V) in L. Since φ_0 is conforming, it separates s from t. It follows that $v_\ell \neq t = v_z$, and thus $\ell < z$. Since $v = v_\ell$ is the last vertex in V that lies left of φ_0, and $\ell < z$, both its outgoing edges cross φ_0. Furthermore, the area enclosed by these edges and φ_0 is empty of other vertices: the trajectories that visit such a vertex would have to cross φ_0 twice, or they would have to intersect the outgoing edges of v (see Fig. 8). Since such an intersection point would

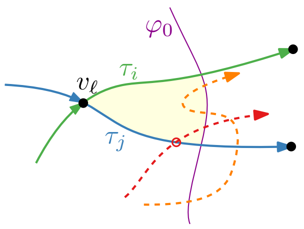


Figure 8 The region enclosed by the outgoing edges of $v_\ell = v$ and φ_0 (yellow) must be empty.

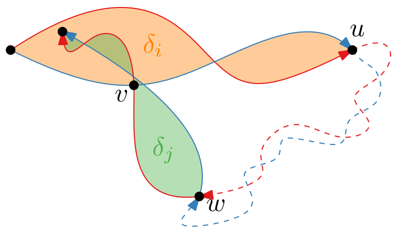


Figure 9 If δ_i and δ_j intersect in v then u, w, and v form a cycle.

have been a vertex in G both these cases cannot occur. Let τ_i and τ_j be the trajectories on the outgoing edges of v. Since φ_0 is conforming, it intersects τ_i and τ_j at most once, namely on the outgoing edges of v. Therefore, φ_0 does not intersect the incoming edges of v. It follows that we can continuously deform φ_0 into a conforming curve φ'_0 that (a) intersects the trajectories in the same order as φ_0, with the exception of τ_i and τ_j; they are swapped, and (b) has the set of vertices $L \setminus \{v\}$ to its left, by sweeping over vertex v, and while remaining conforming at any time. Since the number of vertices to the left of φ'_0 is only $m-1$, the induction hypothesis gives us that there is a continuous deformation from φ'_0 into φ_1. This completes the proof. ◀

▶ **Lemma 9.** *Let φ_0 and φ_1 be conforming curves. There is a conforming sweep-curve γ that deforms $\gamma(0) = \varphi_0$ into $\gamma(1) = \varphi_1$.*

▶ **Lemma 10.** *Let γ_1 and γ_2 be two conforming sweep-curves, with $\gamma_1(0) = \gamma_2(0)$ and $\gamma_1(1) = \gamma_2(1)$, and let $\mu_1 = \mu_{\gamma_1}(u)$ and $\mu_2 = \mu_{\gamma_2}(u)$ be their corresponding median curves for $u \in [0,1]$. We have that $\mu_1 = \mu_2$.*

Proof. Let $u_1,..,u_k$ and $v_1,..,v_\ell$ be the vertices of μ_1 and μ_2, respectively. Since, $\gamma_1(0) = \gamma_2(0)$, the order in which $\gamma_1(0)$ and $\gamma_2(0)$ intersect the trajectories is the same. It follows that $\mu_1(0) = c(\gamma_0, 0, \lceil n/2 \rceil) = c(\gamma_1, 0, \lceil n/2 \rceil) = \mu_2(0)$, and thus $u_1 = v_1$.

Assume by contradiction that i is the index at which μ_1 and μ_2 diverge for the first time. So, μ_1 and μ_2 both arrive at $v = u_i = v_i$ on the same incoming edge, and leave on different outgoing edges of v. Clearly, μ_j, $j \in [1,2]$, changes only if γ_j sweeps over a vertex of G. However, since γ_j is conforming, the number of curves intersected by γ_j before v does not change when it sweeps over a vertex $w \neq v$. This means that μ_1 and μ_2 also use the same outgoing edge of $v = v_i = u_i$. This contradicts the fact that i is the first index on which μ_1 and μ_2 diverge. ◀

Recall that μ_γ is the curve that for any value u corresponds to the $\lceil n/2 \rceil^{\text{th}}$ intersection point on $\gamma(u)$. Lemma 10 then implies:

▶ **Corollary 11.** *There is a unique curve μ^* connecting s to t, such that for any conforming sweep-curve γ, we have that $\mu_\gamma \subseteq \mu^*$.*

A conforming sweep-curve γ is *complete* if and only if the only point from $\bigcup \mathcal{T}$ left of $\gamma(0)$ is s, and the only point from $\bigcup \mathcal{T}$ right of $\gamma(1)$ is t. We then have:

▶ **Lemma 12.** *Let γ be a conforming sweep-curve that is complete, as defined above. The simple median μ^* minimizes* $F_\gamma(\mu) = \displaystyle\int_{u\in[0,1]} \sum_{\tau_i \in \mathcal{T}} \int_{z \in [z_{\mu(u)}, z_{\tau_i}(u)]} |J(u,z)| \,\mathrm{d}z\,\mathrm{d}u,$

where $J(u,z)$ is the Jacobian determinant of γ, and $z_{\mu(u)}$ and $z_{\tau_i}(u)$ are the z-coordinates of points $\mu(u)$ and $\tau_i(u)$ respectively.

Proof. The inner integral $\int_{z\in[z_{\mu(u)},z_{\tau_i}(u)]}|J(u,z)|\,\mathrm{d}z$ represents the length of the curve $\gamma(u)$ between the two intersection points with curves μ and τ_i. Analogous to Lemma 5 we note that $\mu_\gamma(u)$ minimizes $f(\mu)=\sum_{i=1}^{n} len_\gamma(\mu(u)-\tau_i(u))$, and therefore μ_γ minimizes $F_\gamma(\mu)$. The lemma follows from the fact that $\mu^*=\mu_\gamma$. ◀

As in Theorem 7 we now rewrite $\mathrm{HA}_{\mathrm{sum}}(\mu)$ as an integral over u. However, instead of directly mapping u to a vertical line we map it to a conforming curve. The resulting mapping is a conforming sweep-curve. Thus, we prove:

▶ **Lemma 13.** *For any pair of simple paths A and B in G from s to t, the atomic curves in $\Delta(A,B)$ are disjoint.*

Proof. Assume, by contradiction, that $\delta_i\in\Delta(A,B)$ and $\delta_j\in\Delta(A,B)$, with $i<j$, are not disjoint. Then there is an intersection vertex v between δ_i and δ_j. Let $\prec$ denote the topological order of the vertices in G, let u be the ending vertex of δ_i and let w be the starting vertex of δ_j (see Fig. 9). Since $i<j$ we have that $u\prec w$, and since v lies on δ_i we have that $v\prec u$. However, v lies also on δ_j, so we have $w\prec v$, and thus $v\prec u\prec w\prec v$. Contradiction. ◀

▶ **Theorem 14.** *Let $\mathcal{T}$ be a set of trajectories for which G is acyclic, and s and t are on the outer face of G. The simple median μ^* minimizes HA_{sum}.*

Proof. It is easy to see that there is a conforming curve φ_0 which, from the points in $\bigcup\mathcal{T}$, has just s to its left. Similarly, there exists a conforming curve φ_1 that has only t to its right. Therefore, by Lemma 9 there is a complete conforming sweep-curve γ. Lemma 12 then gives us that the simple median μ^* minimizes $F_\gamma(m)$. We now show that $\mathrm{HA}_{\mathrm{sum}}(\mu)=F_\gamma(m)$. Using the result of Chambers and Wang [7] we again rewrite $\mathrm{HA}_{\mathrm{sum}}(\mu)$ as

$$\begin{aligned}\mathrm{HA}_{\mathrm{sum}}(\mu)&=\sum_{\tau_i\in\mathcal{T}}\mathrm{HA}(\mu,\tau_i)=\sum_{\tau_i\in\mathcal{T}}\sum_{\delta\in\Delta(\mu,\tau_i)}|W(\delta)|=\\&=\sum_{\tau_i\in\mathcal{T}}\sum_{\delta\in\Delta(\mu,\tau_i)}\iint_{(x,y)\in\mathbb{R}^2}|\omega(p(x,y),\delta)|\,\mathrm{d}x\,\mathrm{d}y=\\&=\sum_{\tau_i\in\mathcal{T}}\sum_{\delta\in\Delta(\mu,\tau_i)}\int_{u\in\mathbb{R}}\int_{z\in\mathbb{R}}|\omega(\gamma(u,z),\delta)|\,|J(u,z)|\,\mathrm{d}z\,\mathrm{d}u\,.\end{aligned}$$

Since for all $u\in[0,1]$, $\gamma(u)$ intersects every atomic closed curve $\delta\in\Delta(\mu,\tau_i)$ in exactly two points, and due to Lemma 13 the curves in $\Delta(\mu,\tau_i)$ are pairwise disjoint, we get that

$$\sum_{\delta\in\Delta(\mu,\tau_i)}\int_{z\in\mathbb{R}}|\omega(\gamma(u,z),\delta)|\,|J(u,z)|\,\mathrm{d}z=\int_{z\in[z_{\mu(u)},z_{\tau_i}(u)]}|J(u,z)|\,\mathrm{d}z\,,$$

where $z_{\mu(u)}$ and $z_{\tau_i}(u)$ are z-coordinates of the intersection points of $\gamma(u)$ with μ and τ_i respectively. And since $\gamma(u)$ does not intersect any trajectory in $\mathcal{T}$ for $u\notin[0,1]$,

$$\begin{aligned}\mathrm{HA}_{\mathrm{sum}}(\mu)&=\sum_{\tau_i\in\mathcal{T}}\int_{u\in\mathbb{R}}\int_{z\in[z_{\mu(u)},z_{\tau_i}(u)]}|J(u,z)|\,\mathrm{d}z\,\mathrm{d}u=\\&=\int_{u\in[0,1]}\mathrm{d}u\cdot\sum_{\tau_i\in\mathcal{T}}\int_{z\in[z_{\mu(u)},z_{\tau_i}(u)]}|J(u,z)|\,\mathrm{d}z=F_\gamma(\mu)\,.\end{aligned}$$ ◀

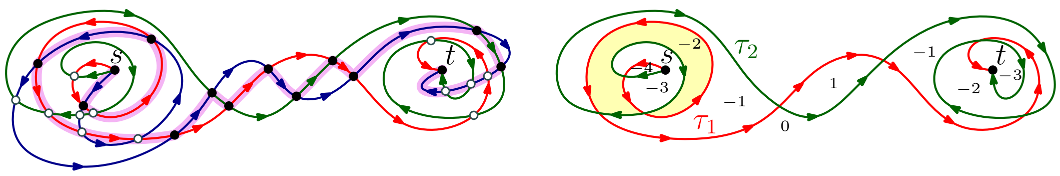


Figure 10 Left: A set of three trajectories with s and t lying on the boundary of an interior face. The optimal representative (light-purple) does not switch at every intersection. Right: The winding numbers for $loop(\tau_1, \tau_2)$. The highlighted face is swept twice by a minimal homotopy.

▶ **Observation 15.** *Note that the simple median μ^* minimizes F_γ, and thus HA_{sum}, among all curves from s to t, even ones that are not necessarily restricted to consist of pieces of the input trajectories.*

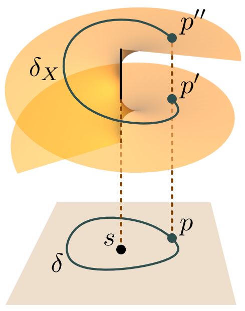
▶ Remark. When n is odd there is a unique curve that minimizes F_γ, and it is simple median μ^*. When n is even there can be multiple curves, not necessarily restricted to consist of pieces of the input trajectories, that all minimize F_γ. These curves are all bounded by the $\frac{n}{2}$- and $(\frac{n}{2}+1)$-levels of the trajectories.

4.3 Extending to Acyclic G with Unrestricted s and t

In this section we extend our approach to compute an optimal representative trajectory when s and t can be anywhere in the DAG. However, unlike in the previous two sections, we can no longer start at the median trajectory from s and switch at every intersection point we encounter. Fig. 10 (left) shows an example of a set of trajectories in which any curve that always switches is not optimal, no matter where we start. The main reason why our argument breaks here is that the winding numbers between the individual pairs of curves are no longer just in the range $[-1, 1]$. Hence, an optimal homotopy may have to sweep over a face more than once.

Instead, we will lift the trajectories into a space X that we will construct from the covering spaces of $\mathbb{R}^2 \setminus s$ and $\mathbb{R}^2 \setminus t$; we refer the reader to a standard topology text for detailed definitions of covering spaces [14, 18]. The key in this setting is that we will be able to lift the trajectories into X in such a way that the trajectory graph will form a DAG with s and t on the outer face and the pairwise homotopy areas between lifted trajectories will be the same as the homotopy area in the plane. We then compute an optimal representative for the lifted trajectories, using our simple median algorithm, and show that its corresponding projection is an optimal representative for the original trajectories.

The space X. Intuitively, we start with a covering space of the space formed from the plane by cutting out small disks around s and t, where each of the boundaries is collapsed to a single point, and then obtain space X by adding the points s and t back. This means that X will cover $\mathbb{R}^2 \setminus \{s, t\}$ with infinitely many "layers" forming Riemann-like spirals around points s and t. Consider a simple cycle δ in $\mathbb{R}^2$ that goes through some point p and that encloses s or t. A walk along δ starting at p and ending at p in $\mathbb{R}^2$ corresponds to a curve in X that starts at the copy of p in some layer i and ends at the copy of p in layer $i+1$ or $i-1$. See Fig. 11 for an illustration. We formalize this more carefully (including the metric on the space X, which will be necessary in order to argue about the homotopy area) using a particular conforming curve, as follows:



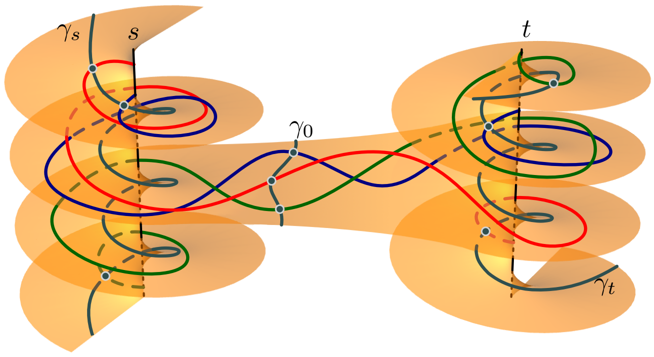


Figure 11 Lifting cycle δ from $\mathbb{R}^2$ into X.

Figure 12 Input trajectories lifted into the space X.

▶ **Lemma 16.** *Given a set of simple trajectories $\mathcal{T}$ that start at s and end in t, whose arrangement forms a directed acyclic graph G, there exists a conforming curve γ with endpoints at infinity that separates s and t.*

Proof. The graph G defines a partial order on the intersection points of the trajectories in $\mathcal{T}$. Consider a conforming 0-length cycle δ enclosing s. Let δ grow by sweeping over the vertices of G according to their partial order. As in Lemma 8 we can do this while maintaining conformity with respect to $\mathcal{T}$. Once γ crosses some intersection point that lies on the outer face of G, we can cut it at any point in the outer face and pull the endpoints toward infinity. The resulting open curve γ is conforming and separates s from t. ◀

Let $\gamma \subset \mathbb{R}^2$ be a conforming curve that separates s from t and has its end-points at infinity, and let p_0 be an arbitrary point on γ. Note that by Lemma 16 such a curve exists. Define space X_s corresponding to $\mathbb{R}^2\backslash\{s\}$ in terms of polar coordinates, taking s to be the origin: Let r be the "radius" parameter, and let θ be the angular parameter, such that point $(0, \|sp_0\|) \in X_s$ corresponds to point $p_0 \in \mathbb{R}^2\backslash\{s\}$, and a positive θ corresponds to a clockwise turn. We then have $X_s = \{(\theta, r) \mid (\theta, r \in \mathbb{R}\backslash\{s\}) \wedge r > 0\}$. Note that in the definition of X_s we explicitly do *not* limit the range of θ to $[0, 2\pi)$ (which would give us exactly $\mathbb{R}^2$, parameterized around s). Instead, our space X_s allows us to "wind around" s an arbitrary number of times. Analogously, define X_t. Note, that X_s and X_t are the universal covers of $\mathbb{R}^2\backslash\{s\}$ and $\mathbb{R}^2\backslash\{t\}$, respectively.

We partition X_s into *layers* L_i, with $i \in \mathbb{Z}$. A point $(\theta, r) \in X_s$ lies in layer L_i, if and only if $\lfloor \theta/2\pi \rfloor = i$. We define layers analogously for X_t.

Recall that γ separates s and t in $\mathbb{R}^2$, and hence we can consider a copy of γ in each of X_s and X_t. We cut each space along the copy of γ, and glue the part of X_s containing s and the part of X_t containing t together along each's copy of γ. Furthermore, we again add the points s and t, and connect them to all the layers of X_s and X_t. Note, that we add the points s and t only so that all trajectories again start at s and end at t rather than arbitrarily close to s and t. Let X to be the resulting space.

Lifting trajectories. Next, we describe the image of trajectories $\mathcal{T}$ in X. (See Fig. 12.) For each trajectory $\tau_i \in \mathcal{T}$, we construct its corresponding trajectory τ_i' in X by starting at the image of the intersection point of τ_i with γ, and moving along τ_i while continuously mapping the points to τ_i'. We call this process *lifting* trajectory τ_i to space X. Let $\mathcal{T}' = \{\tau_i' \mid \tau_i \in \mathcal{T}\}$ denote the set of resulting trajectories, and let G' be the corresponding trajectory graph.

We note that all trajectories cross the conforming curve γ and hence are fixed on a common reference, although not at a common lift of a base point as is more commonly seen in topology. However, we obtain that each trajectory lifts to a unique curve in X. In addition, since we have local homeomorphisms which lift everywhere (except right at s and t), we can also lift the definition of a winding number for any point p inside $loop(\tau_i', \tau_j')$ for any pair of lifted trajectories τ_i' and τ_j'. This leads to the following observation:

▶ **Observation 17.** *The points s and t lie on the outer face of G'. Thus, for any simple paths A and B in G' from s to t, and for any point $p \in X$, we have that $|\omega(p, loop(A, B))| \leq 1$.*

▶ **Lemma 18.** *For any two curves ϕ_1 and ϕ_2 in $\mathbb{R}^2$ that connect s to t. We have $HA(\phi_1, \phi_2) = HA(\phi_1', \phi_2')$, where ϕ_1' and ϕ_2' are the corresponding curves lifted into space X.*

Proof. It is easy to see that $\mathrm{HA}(\phi_1, \phi_2) \leq \mathrm{HA}(\phi_1', \phi_2')$: the covering map f_X is continuous, so a minimum homotopy between ϕ_1' and ϕ_2' defines a homotopy between ϕ_1 and ϕ_2 of cost $\mathrm{HA}(\phi_1', \phi_2')$. Since $\mathrm{HA}(\phi_1, \phi_2)$ is a minimum homotopy we have that $\mathrm{HA}(\phi_1, \phi_2) \leq \mathrm{HA}(\phi_1', \phi_2')$.

Next, we show that $\mathrm{HA}(\phi_1, \phi_2) \geq \mathrm{HA}(\phi_1', \phi_2')$. The lemma then follows. Let $\delta = loop(\phi_1, \phi_2)$ and $\delta' = loop(\phi_1', \phi_2')$. Now assume, by contradiction, that $\mathrm{HA}(\phi_1, \phi_2) > \mathrm{HA}(\phi_1', \phi_2')$. It follows that there is a point $p \in \mathbb{R}^2$, with $\omega(p, \delta) = k$ that is swept by a minimum homotopy H between ϕ_1' and ϕ_2' more than k times. Furthermore, assume without loss of generality that p lies left of γ, and thus the copies of p lie in X_s. Since all winding numbers in X are in the range $[-1, 1]$ (Observation 17) that means there must be more than k copies of point p swept by H. It follows that there is a point q' in layer L_ℓ or layer $L_{-\ell}$, with $\ell > k$, that lies on δ', and has a larger r-coordinate than p' (otherwise we would not sweep over p'). Furthermore, note that δ' intersects γ (as the curves ϕ_1' and ϕ_2' connect s to t), and thus contains a point on in layer L_0. Now consider traversing δ', starting from point q'. It follows that the total turning angle is at least $2\ell\pi$ (since we must visit layer L_0). This means that the total turning angle of curve δ with respect to $f_X(p)$ is also at least $\ell 2\pi$. Therefore $\omega(p, \delta) \geq \ell > k$. Contradiction. ◀

▶ **Corollary 19.** *Let μ be a representative for the set of trajectories $\mathcal{T}$, and let μ' be its corresponding representative for $\mathcal{T}'$. We have that $HA_{sum}(\mu, \mathcal{T}) = HA_{sum}(\mu', \mathcal{T}')$.*

Any representative trajectory μ in X corresponds to some representative trajectory μ' in $\mathbb{R}^2$. However, not every representative trajectory in $\mathbb{R}^2$ has a corresponding representative trajectory in X. The difference between the two cases comes from the fact, that some of the intersection points between trajectories $\mathcal{T}$ that existed in $\mathbb{R}^2$ no longer exist once the trajectories are lifted to X. We call the intersection points of $\mathcal{T}$ that remain in X *legal*, and the ones that disappear *illegal*. Following the projection to $\mathbb{R}^2$ of the median trajectory μ_*' corresponds to switching the trajectory at every legal intersection point. Next we will prove, that this projection of μ_*' to $\mathbb{R}^2$ gives an optimal representative trajectory for trajectories $\mathcal{T}$.

▶ **Theorem 20.** *Let $\mathcal{T}$ be a set of trajectories for which G is acyclic, let μ_*' be the simple median on the lifted trajectories in X. The representative curve $\mu^* = f_X(\mu_*')$ corresponding to μ_*' minimizes HA_{sum} with respect to $\mathcal{T}$.*

Proof. Suppose there exists some representative trajectory μ with the total homotopy area $\mathrm{HA}_{\mathrm{sum}}(\mu, \mathcal{T}) < \mathrm{HA}_{\mathrm{sum}}(\mu^*, \mathcal{T})$. Let curve μ' be μ lifted to X. If μ uses only legal intersections in G, then μ' is a candidate representative for the set of trajectories $\mathcal{T}'$. By Corollary 19 we then have that $\mathrm{HA}_{\mathrm{sum}}(\mu, \mathcal{T}) = \mathrm{HA}_{\mathrm{sum}}(\mu', \mathcal{T}') < \mathrm{HA}_{\mathrm{sum}}(\mu_*', \mathcal{T}') = \mathrm{HA}_{\mathrm{sum}}(\mu^*, \mathcal{T})$. Contradiction. If μ uses also illegal intersections, then μ' does not consist of pieces of the trajectories $\mathcal{T}'$.

Theorem 14 and Observation 15 then implies that $\mathrm{HA}_{\mathrm{sum}}(\mu', \mathcal{T}') \geq \mathrm{HA}_{\mathrm{sum}}(\mu'_*, \mathcal{T}')$. Applying Corollary 19 on both sides then gives us $\mathrm{HA}_{\mathrm{sum}}(\mu, \mathcal{T}) \geq \mathrm{HA}_{\mathrm{sum}}(\mu^*, \mathcal{T})$. Contradiction. ◀

5 Computing a Representative Trajectory

From Theorem 14 it immediately follows that if s and t lie on the outer face of G, we can compute a representative μ^* trajectory that minimizes $\mathrm{HA}_{\mathrm{sum}}$ using the algorithm of Buchin et al. [4]. Thus, we can compute μ^* in $O((N+k)\alpha(N)\log(N))$ time, where N is the total complexity of the input trajectories, and k is the output complexity. For an arbitrary DAG G, we have $k = O(N^2)$. If the trajectories are x-monotone, the simple median corresponds to the $\lceil n/2 \rceil$-level in an arrangement of n curves, and thus bounds on the complexity of the $\lceil n/2 \rceil$-level also bound the complexity of μ^*. In case our trajectories (curves) are all polylines with at most m vertices each, we have $k = O(mn^{4/3} \log^{1/3-\varepsilon} n)$, for some arbitrarily small constant $\varepsilon > 0$ [9]. Similarly, we can derive the results for more general types of curves.

Unrestricted s and t. When s and t are not restricted to lie on the outer face we first construct a conforming curve γ that separates s from t and intersects the outer face. This allows us to find the $\lceil n/2 \rceil^{\mathrm{th}}$ intersection p of γ with the trajectories $\mathcal{T}$, which is guaranteed to lie on the representative μ'_* that minimizes the homotopy area of the lifted trajectories $\mathcal{T}'$. We can now construct G' from G by walking along the trajectories, starting from their intersection points with γ. Similarly, we can trace μ'_* trough G', starting from p. The representative μ'_* then also gives us an optimal representative μ^* (Theorem 20). All that remains is to describe how to construct γ. We do this using the same procedure as used in the proof of Lemma 16: we explicitly construct G, sort the vertices in topological order, and add the vertices in this order to some set L. Once L contains a vertex v on the outer face of G, we can construct γ, starting on the outgoing edge of v incident to the outer face, and walking through G, while keeping exactly the set of vertices L to our left. It is easy to see that computing μ^* using this algorithm takes $O(|G|) = O(N^2)$ time. We summarize our results in the following theorem.

▶ **Theorem 21.** *Let $\mathcal{T}$ be a set of trajectories that all start in s and end in t, and whose trajectory graph G is acyclic. If s and t lie on the outer face, a representative trajectory μ^* that minimizes HA_{sum} can be computed in $O((N+k)\alpha(N)\log(N))$ time, where N is the total complexity of the trajectories in $\mathcal{T}$, and k is the complexity of the resulting trajectory. If s and t can be anywhere in G, μ^* can be computed in $O(|G|) = O(N^2)$ time.*

6 Future Work

We have shown that computing a representative that minimizes $\mathrm{HA}_{\mathrm{sum}}$ is NP-hard when the trajectory graph G may be an arbitrary graph, and we have presented an efficient algorithm for when G is a DAG. Hence, our results cover all cases. However, clearly there are situations in which the trajectories are similar, but for which the trajectory graph is not a DAG. Hence, we would like a more fine grained classification which kind of graphs allow us to find a representative efficiently.

We expect that we can extend our approach from Section 4.3 to cases in which the trajectories are have a similar "shape" but their trajectory graph contains cycles. In particular, we again lift the trajectories into a space, or *corridor*, that captures the global shape of the trajectories, and in which the trajectory graph forms a DAG. We then compute a concrete curve representing the trajectories in this space. The conceptual existence of a

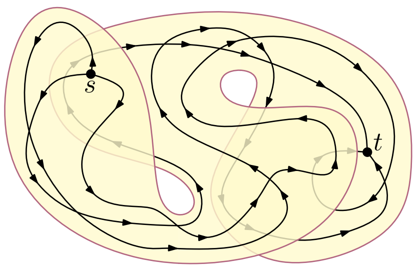


Figure 13 Even when trajectories do not form a DAG, the trajectories may lie in a "corridor".

corridor is justified by the assumption that the input trajectories are similar. See Fig. 13 for an illustration.

While we focused on using the homotopy area to measure distance between the trajectories, there are other alternative measures that balance topology and geometry. Homotopy width (or homotopic Fréchet distance) [8] and homotopy height [5, 13] are obvious options, as is homology area [6], although it is unclear if any of these are tractable or useful in practice.

References

1 Pankaj K. Agarwal, Mark de Berg, Jie Gao, Leonidas J. Guibas, and Sariel Har-Peled. Staying in the middle: Exact and approximate medians in $\mathbb{R}^1$ and $\mathbb{R}^2$ for moving points. In *CCCG*, pages 43–46, 2005.

2 Riddhipratim Basu, BhaswarB. Bhattacharya, and Tanmoy Talukdar. The projection median of a set of points in $\mathbb{R}^d$. *Discrete & Computational Geometry*, 47(2):329–346, 2012.

3 Kevin Buchin, Maike Buchin, Joachim Gudmundsson, Maarten Löffler, and Jun Luo. Detecting commuting patterns by clustering subtrajectories. *IJCGA*, 21(03):253–282, 2011. doi:10.1142/S0218195911003652.

4 Kevin Buchin, Maike Buchin, Marc Kreveld, Maarten Löffler, RodrigoI. Silveira, Carola Wenk, and Lionov Wiratma. Median trajectories. *Algorithmica*, 66(3):595–614, 2013. doi:10.1007/s00453-012-9654-2.

5 Erin W. Chambers and David Letscher. On the height of a homotopy. In *CCCG*, pages 103–106, 2009.

6 Erin Wolf Chambers and Mikael Vejdemo-Johansson. Computing minimum area homologies. *Computer Graphics Forum*, 2014. doi:10.1111/cgf.12514.

7 Erin Wolf Chambers and Yusu Wang. Measuring similarity between curves on 2-manifolds via homotopy area. In *Proc. 29th Ann. Symp. on CG*, pages 425–434. ACM, 2013. doi:10.1145/2462356.2462375.

8 Erin Wolf Chambers, Éric Colin de Verdière, Jeff Erickson, Sylvain Lazard, Francis Lazarus, and Shripad Thite. Homotopic fréchet distance between curves or, walking your dog in the woods in polynomial time. *CG*, 43(3):295–311, 2010. doi:10.1016/j.comgeo.2009.02.008.

9 Timothy M Chan. On levels in arrangements of curves, iii: further improvements. In *Proc. of the 24th annual symposium on Computational geometry*, pages 85–93. ACM, 2008.

10 Stephane Durocher and David Kirkpatrick. The projection median of a set of points. *CG*, 42(5):364–375, 2009.

11 S. Gaffney, A. Robertson, P. Smyth, S. Camargo, and M. Ghil. Probabilistic clustering of extratropical cyclones using regression mixture models. *Climate Dynamics*, 29(4):423–440, 2007.

12 S. Gaffney and P. Smyth. Trajectory clustering with mixtures of regression models. In *Proc. 5th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, pages 63–72, 1999.

13 Sariel Har-Peled, Amir Nayyeri, Mohammad Salavatipour, and Anastasios Sidiropoulos. How to walk your dog in the mountains with no magic leash. In *Proc. 28th Ann. Symp. on CG*, pages 121–130. ACM, 2012. `doi:10.1145/2261250.2261269`.

14 Allen Hatcher. *Algebraic Topology*. Cambridge University Press, 2001. URL: `http://www.math.cornell.edu/~hatcher/`.

15 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`.

16 J.G. Lee, J. Han, and K.Y. Whang. Trajectory clustering: a partition-and-group framework. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 593–604, 2007.

17 David Lichtenstein. Planar formulae and their uses. *SIAM J. Comput.*, 11(2):329–343, 1982. `doi:10.1137/0211025`.

18 James R. Munkres. *Topology*. Prentice-Hall, 2nd edition, 2000.

19 M. Vlachos, D. Gunopulos, and G. Kollios. Discovering similar multidimensional trajectories. In *Proc. 18th Int. Conf. Data Engin.*, pages 673–684, 2002.

Optimal Reachability and a Space-Time Tradeoff for Distance Queries in Constant-Treewidth Graphs[*]

Krishnendu Chatterjee[1], Rasmus Ibsen-Jensen[2], and Andreas Pavlogiannis[3]

1 IST Austria, Klosterneuburg, Austria
`kchatterjee@ist.ac.at`
1 IST Austria, Klosterneuburg, Austria
`ribsen@ist.ac.at`
1 IST Austria, Klosterneuburg, Austria
`pavlogiannis@ist.ac.at`

Abstract

We consider data-structures for answering reachability and distance queries on constant-treewidth graphs with n nodes, on the standard RAM computational model with wordsize $W = \Theta(\log n)$. Our first contribution is a data-structure that after $O(n)$ preprocessing time, allows (1) pair reachability queries in $O(1)$ time; and (2) single-source reachability queries in $O(\frac{n}{\log n})$ time. This is (asymptotically) *optimal* and is *faster than DFS/BFS* when answering more than a constant number of single-source queries. The data-structure uses at all times $O(n)$ space. Our second contribution is a space-time tradeoff data-structure for distance queries. For any $\epsilon \in [\frac{1}{2}, 1]$, we provide a data-structure with polynomial preprocessing time that allows pair queries in $O(n^{1-\epsilon} \cdot \alpha(n))$ time, where α is the inverse of the Ackermann function, and at all times uses $O(n^{\epsilon})$ space. The input graph G is not considered in the space complexity.

1998 ACM Subject Classification G.2.2 Graph Theory: Graph algorithms

Keywords and phrases Graph algorithms; Constant-treewidth graphs; Reachability queries; Distance queries

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.28

1 Introduction

In this work we consider two of the most classic graph algorithmic problems, namely the reachability and distance problems, on low-treewidth graphs. We consider the case where the input is a graph G with n nodes and a tree-decomposition Tree(G) of G with $b = O(n)$ bags and width t. The computational model is the standard RAM with wordsize $W = \Theta(\log n)$.

Low-treewidth graphs. A very well-known concept in graph theory is the notion of *treewidth* of a graph, which is a measure of how similar a graph is to a tree (a graph has treewidth 1 precisely if it is a tree) [30]. The treewidth of a graph is defined based on a *tree decomposition* of the graph [24], see Section 2 for a formal definition. Beyond the mathematical elegance of the treewidth property for graphs, there are many classes of graphs

* The research was partly supported by Austrian Science Fund (FWF) Grant No P23499-N23, FWF NFN Grant No S11407-N23 (RiSE/SHiNE) and ERC Start grant (279307: Graph Games).

which arise in practice and have low (even constant) treewidth. An important example is that the control flow graph for goto-free programs for many programming languages are of constant treewidth [32]. Also many chemical compounds have treewidth 3 [34]. For many other applications see the surveys [11, 10]. Given a tree decomposition of a graph with low treewidth t, many problems on the graph become complexity-wise easier (i.e., many NP-complete problems for arbitrary graphs can be solved in time polynomial in the size of the graph, but exponential in t, given a tree decomposition [3, 7, 8]). Even for problems that can be solved in polynomial time, faster algorithms can be obtained for low-treewidth graphs, for example, for the distance (or the shortest path) problem [16]. The constant treewidth of control flow graphs has also been shown to lead to faster algorithms for interprocedural analysis [14], quantitative verification [15], and analysis of concurrent programs [13].

Reachability/distance problems. The *pair* reachability (resp., distance) problem is one of the most classic graph algorithmic problems that, given a pair of nodes u, v, asks to compute if there is a path from u to v (resp., the weight of the shortest path from u to v). The *single-source* variant problem given a node u asks to solve the pair problem u, v for every node v. Finally, the *all pairs* variant asks to solve the pair problem for each pair u, v. While there exist many classic algorithms for the distance problem, such as A^*-algorithm (pair) [26], Dijkstra's algorithm (single-source) [19], Bellman-Ford algorithm (single-source) [5, 23, 28], Floyd-Warshall algorithm (all pairs) [22, 33, 31], and Johnson's algorithm (all pairs) [27] and others for various special cases, there exist in essence only two different algorithmic ideas for reachability: Fast matrix multiplication (all pairs) [21] and DFS/BFS (single-source) [18].

Previous results. The algorithmic question of the distance (pair, single-source, all pairs) problem for low-treewidth graphs has been considered extensively in the literature, and many data-structures have been presented [2, 16, 29, 1, 4, 17]. The previous results are incomparable, in the sense that the best data-structure depends on the treewidth and the number of queries. The pair query reachability for low-treewidth graphs has been considered in [35]. Despite many results for constant (or low) treewidth graphs, none of them improves the complexity for the basic single-source reachability problem, i.e., the bound for DFS/BFS has not been improved in any of the previous works.

Our results. Our algorithms take as input a graph G with n nodes. Our main contributions are as follows (summarized in Table 1 and Table 2):

1. Our first contribution is a data-structure that supports reachability queries in G. The computational complexity we achieve is as follows: (i) $O(n \cdot t^2)$ preprocessing (construction) time; (ii) $O(n \cdot t)$ space; (iii) $O(\lceil t/\log n \rceil)$ pair-query time; and (iv) $O(n \cdot t/\log n)$ time for single-source queries. Note that for constant-treewidth graphs, the data-structure is *optimal* in the sense that it only uses linear preprocessing time, and supports answering queries in the size of the output (the output for single-source queries requires one bit per node, and thus has size $\Theta(n/W) = \Theta(n/\log n)$). Moreover, also for constant-treewidth graphs, the data-structure answers single-source queries faster than DFS/BFS, after linear preprocessing time (which is asymptotically the same as for DFS/BFS). Thus there exists a constant c_0 such that the total of the preprocessing and querying time of the data-structure is smaller than that of DFS/BFS for answering at least c_0 single-source queries.
2. Second, we present a space-time tradeoff data-structure that supports distance pair queries in G and given a number $\epsilon \in [\frac{1}{2}, 1]$. The weights of G come from the set of

Table 1 Data-structures for pair and single-source reachability queries, on a directed graph G with n nodes, m edges, and a treewidth t. The model of computation is the standard RAM model with wordsize $W = \Theta(\log n)$. Space usage refers to the total space used during the preprocessing and query phase. Rows 1 and 2 are previous results, and row i is the result of this paper.

Row	Preprocessing time	Space usage	Pair query time	Single-source query time	From
1	$O(n \cdot \log n)$	$O(n \cdot \log n)$	$O(\log n)$	$O(n \cdot \log n)$ a)	[35] b)
2	–	$O(\lceil n/\log n \rceil)$	$O(m)$	$O(m)$	DFS/BFS [18]
i	$O(n \cdot t^2)$	$O(n \cdot t)$	$O(\lceil \frac{t}{\log n} \rceil)$	$O(\frac{n \cdot t}{\log n})$	Theorem 6

a) Obtained by multiplying the time for a pair query by n.
b) The result is only stated for constant treewidth.

Table 2 Data-structures for pair and single-source distance queries, on a weighted directed graph G with n nodes, m edges, and a tree decomposition of width $O(1)$ and height h. The number ϵ can be any fixed number in $[\frac{1}{2}, 1]$ and $\alpha(n)$ is the inverse Ackermann function. Space usage refers to the total space used during the preprocessing and query phase. When measuring space complexity, we do not count the input size. Rows 1-6 are previous results, and row i is the result of this paper.

Row	Preprocessing time	Space usage	Pair query time	Single-source query time	From
1	$O(n^2)$	$O(n^2)$	$O(1)$	$O(n)$	[29] a)
2	$O(n)$	$O(n)$	$O(\alpha(n))$	$O(n)$	[16]
3	$O(n \cdot \log h)$	$O(n)$	$O(\log \log n)$	$O(n \cdot \log \log n)$ b)	[2]
4	$O(n \cdot \log^2 n)$	$O(n \cdot \log n)$	$O(\log n)$	$O(n \cdot \log n)$ b)	[1]
5	$O(n \cdot \log n)$	$O(n \cdot \log n)$	$O(\log^2 n)$	$O(n \cdot \log^2 n)$ b)	[4, 17]
6	Not given	$O(n^\epsilon \cdot \log^2 n)$ c)	$O(n^{1-\epsilon} \cdot \log n)$	– d)	[2] e)
i	polynomial	$O(n^\epsilon)$	$O(n^{1-\epsilon} \cdot \alpha(n))$	– d)	Theorem 13

a) This data-structure solves the all pairs problem in the given time and space bounds.
b) Obtained by multiplying the time for a pair query by n.
c) This is the space usage after preprocessing.
d) Not given/supported since the size of the output is larger than the data-structure.
e) Note that [2] does not explicitly state the tradeoff given (they only state linear space), but it follows from their technique by picking other values for their variable k. Also, note that [2] requires a tree-decomposition to be part of the input, whereas our data-structure only requires that the graph G is part of the input.

integers $\mathbb{Z}$, but we do not allow negative cycles. For constant-treewidth graphs, our data-structure requires (i) polynomial preprocessing time; (ii) $O(n^\epsilon)$ working space; and (iii) $O(n^{1-\epsilon} \cdot \alpha(n))$ time for pair queries.

The graph G is considered part of the input, and is not counted towards the space complexity.

Technical contributions. Our results rely on three key technical contributions:

1. For pair reachability queries, the key idea is to store reachability information from each node to $O(\log n)$ other nodes. For single-source queries, for some nodes this reachability information might be of size $\Theta(n)$, but on average remains $O(\log n)$. Our data-structure computes reachability information in such a way that allows for compact representation and fast retrieval using word tricks, which for constant-treewidth graphs leads to asymptotically optimal preprocessing and query (both pair and single-source) bounds. The idea of storing $O(\log n)$ information per node has appeared before ([35, 16]) however those algorithms follow different approaches, where word tricks do not seem to be applicable (at least not without significantly modifying the algorithms).
2. For distance queries, we devise a procedure for shrinking a tree-decomposition of size $O(n)$ to one of size $O(n^{1-\epsilon})$, by partitioning the tree-decomposition to components of

sufficient size. A key property of this partitioning is that each component has only a constant number of neighbor components. We show how this shrank tree-decomposition can be preprocessed for answering pair distance queries in the stated bounds.

2 Preliminaries

Graphs. We consider weighted directed graphs $G = (V, E, \mathsf{wt})$ where V is a set of n nodes, $E \subseteq V \times V$ is an edge relation of m edges, and $\mathsf{wt} : E \to \mathbb{Z}$ is a weight function where $\mathbb{Z}$ is the set of integers. In the sequel we write graphs for directed graphs, and explicitly mention if the graph is undirected. Given a set $X \subseteq V$, we denote by $G[X]$ the subgraph $(X, E \cap (X \times X))$ of G induced by the set of nodes X. A path $P : u \rightsquigarrow v$ is a sequence of nodes $(x_1, \dots, x_k)$ such that $u = x_1$, $v = x_k$, and for all $1 \leq i \leq k-1$ we have $(x_i, x_{i+1}) \in E$. The path P is *simple* if every node appears at most once in P. The length of P is $k-1$, and a single node is by itself a 0-length path. We denote by $E^* \subseteq V \times V$ the transitive closure of E, i.e., $(u, v) \in E^*$ iff there exists a path $P : u \rightsquigarrow v$. Given a path P, a node u, and a set of nodes A, we use the set notation $u \in P$ to denote that u appears in P, and $A \cap P$ to refer to the set of nodes that appear in both P and A. The weight function is extended to paths, and the weight of a path $P = (x_1, \dots, x_k)$ is $\mathsf{wt}(P) = \sum_{i=1}^{k-1} \mathsf{wt}(x_i, x_{i+1})$ if $k > 1$, else $\mathsf{wt}(P) = 0$. For $u, v \in V$, the distance from u to v is defined as $d(u, v) = \min_{P : u \rightsquigarrow v} \mathsf{wt}(P)$, where P ranges over simple paths in G (and $d(u, v) = \infty$ if no such path exists). We consider that G does not have negative cycles.

Trees. A (rooted) tree $T = (V_T, E_T)$ is an undirected graph with a distinguished node r which is the root such that there is a unique simple path $P_u^v : u \rightsquigarrow v$ for each pair of nodes u, v. The *size* of T is $|V_T|$. Given a tree T with root r, the *level* $\mathsf{Lv}(u)$ of a node u is the length of the simple path P_u^r from u to the root r, and every node in P_u^r is an *ancestor* of u. If v is an ancestor of u, then u is a *descendant* of v. Note that a node u is both an ancestor and descendant of itself. For a pair of nodes $u, v \in V_T$, the *lowest common ancestor (LCA)* of u and v is the common ancestor of u and v with the largest level. The *parent* u of v is the unique ancestor of v in level $\mathsf{Lv}(v) - 1$, and v is a *child* of u. A *leaf* of T is a node with no children. For a node $u \in V_T$, we denote by $T(u)$ the subtree of T rooted in u (i.e., the tree consisting of all descendants of u). The tree T is *binary* if every node has at most two children. The *height* of T is $\max_u \mathsf{Lv}(u)$ (i.e., it is the length of the longest path P_u^r), and T is *balanced* if its height is $O(\log |V_T|)$. Given a tree T, a *connected component* $C \subseteq V_T$ of T is a set of nodes of T such that for every pair of nodes $u, v \in C$, the unique simple path P_u^v in T visits only nodes in C.

Tree decompositions. Given a graph G, a tree-decomposition $\mathrm{Tree}(G) = (V_T, E_T)$ is a tree with the following properties.

T1: $V_T = \{B_1, \dots, B_b :$ for all $1 \leq i \leq b.\ B_i \subseteq V\}$ and $\bigcup_{B_i \in V_T} B_i = V$.

T2: For all $(u, v) \in E$ there exists $B_i \in V_T$ such that $u, v \in B_i$.

T3: For all B_i, B_j and any bag B_k that appears in the simple path $B_i \rightsquigarrow B_j$ in $\mathrm{Tree}(G)$, we have $B_i \cap B_j \subseteq B_k$.

The sets B_i which are nodes in V_T are called *bags*. The *width* of a tree-decomposition $\mathrm{Tree}(G)$ is the size of the largest bag minus 1, and the *treewidth* of G is the width of a minimum-width tree decomposition of G. Let G be a graph, $T = \mathrm{Tree}(G)$, and B_0 be the root of T. For $u \in V$, we say that a bag B is the *root bag* of u if B is the bag with the smallest level among all bags that contain u. By definition, for every node u there exists a unique bag which is the root of u. We often write B_u for the root bag of u, i.e., $B_u = \arg\min_{B_i \in V_T\colon u \in B_i} \mathsf{Lv}(B_i)$, and denote by $\mathsf{Lv}(u) = \mathsf{Lv}(B_u)$. A bag B is said to *introduce* a node $u \in B$ if either B is a leaf, or u does not appear in any child of B. In this work we consider only *binary* tree decompositions (if not, a tree decomposition can be made binary by a standard process that increases its size by a constant factor while keeping the width the same). The following lemma states a well-known "separator property" of tree decompositions.

▶ **Lemma 1.** *Consider a graph $G = (V, E)$, a binary tree-decomposition $T = \mathrm{Tree}(G)$, and a bag B of T. Let $(C_i)_{1\le i\le 3}$ be the components of T created by removing B from T, and let V_i be the set of nodes that appear in bags of component C_i. For every $i \neq j$, nodes $u \in V_i$, $v \in V_j$ and path $P : u \rightsquigarrow v$, we have that $P \cap B \neq \emptyset$ (i.e., all paths between u and v go through some node in B).*

Using Lemma 1, we prove the following stronger version of the separator property, which will be useful throughout the paper.

▶ **Lemma 2.** *Consider a graph $G = (V, E)$ and a tree-decomposition $\mathrm{Tree}(G)$. Let $u, v \in V$, and consider two distinct bags B_1 and B_j such that $u \in B_1$ and $v \in B_j$. Let $P' : B_1, B_2, \dots, B_j$ be the unique simple path in T from B_1 to B_j. For each $i \in \{2, \dots, j\}$ and for each path $P : u \rightsquigarrow v$, there exists a node $x_i \in (B_{i-1} \cap B_i \cap P)$.*

Proof. Let $T = \mathrm{Tree}(G)$. Fix a number $i \in \{2, \dots, j\}$. We argue that for each path $P : u \rightsquigarrow v$, there exists a node $x_i \in (B_{i-1} \cap B_i \cap P)$. We construct a tree T', which is similar to T except that instead of having an edge between bag B_{i-1} and bag B_i, there is a new bag B, that contains the nodes in $B_{i-1} \cap B_i$, and there is an edge between B_{i-1} and B and one between B and B_i. It is easy to see that T' satisfies the properties T1-T3 of a tree-decomposition of G. By Lemma 1, each bag B' in the unique path $P'' : B_1, \dots, B_{i-1}, B, B_i, \dots, B_j$ in T' separates u from v in G. Hence, each path $u \rightsquigarrow v$ must go through some node in B, and the result follows. ◀

The following lemma states that for nodes that appear in bags B, B' of the tree-decomposition $T = \mathrm{Tree}(G)$, their distance can be written as a sum of distances $d(x_i, x_{i+1})$ between pairs of nodes (x_i, x_{i+1}) that appear in bags B_i that constitute the unique $B \rightsquigarrow B'$ path in T.

▶ **Lemma 3.** *Consider a weighted graph $G = (V, E, \mathsf{wt})$ and a tree-decomposition $\mathrm{Tree}(G)$. Let $u, v \in V$, and $P' : B_1, B_2, \dots, B_j$ be a simple path in T such that $u \in B_1$ and $v \in B_j$. Let $A = \{u\} \times \left(\prod_{1<i\le j} (B_{i-1} \cap B_i)\right) \times \{v\}$. Then $d(u, v) = \min_{(x_1,\dots,x_{j+1})\in A} \sum_{i=1}^{j} d(x_i, x_{i+1})$.*

Proof. Consider a witness path $P : u \rightsquigarrow v$ such that $\mathsf{wt}(P) = d(u, v)$. By Lemma 2, there exists some node $x_i \in (B_{i-1} \cap B_i \cap P)$, for each $i \in \{1, \dots, j\}$. It easily follows that $d(u, v) = \sum_{i=1}^{j} d(x_i, x_{i+1})$ with $x_1, \dots x_{j+1} \in A$. ◀

Small tree decompositions. A tree-decomposition $T = \mathrm{Tree}(G) = (V_T, E_T)$ is called *small* if $|V_T| = O(\frac{n}{t})$.

▶ **Lemma 4.** *Given a tree decomposition $\mathrm{Tree}(G)$ of G of width $O(t)$ and $O(n)$ bags, a small, binary tree decomposition $\mathrm{Tree}'(G)$ of width $O(t)$ can be constructed in $O(n \cdot t)$ time. Moreover, if $\mathrm{Tree}(G)$ is balanced, then so is $\mathrm{Tree}'(G)$.*

Proof. Let $k = O(t)$ be the width of $\mathrm{Tree}(G)$. The construction is achieved using the following steps.

1. Following the steps of [9, Lemma 2.4], we turn $\mathrm{Tree}(G)$ to a *smooth* tree-decomposition $T_1 = (V_1, E_1)$, which has the properties that (i) for every bag $B \in V_1$ we have $|B| = k+1$, and (ii) for every pair of bags $(B_1, B_2) \in E_1$ we have $|B_1 \cap B_2| = k$. The process of [9, Lemma 2.4] can be performed $O(n \cdot t)$ time and increases the height by at most a factor 2, hence if $\mathrm{Tree}(G)$ is balanced, T_1 is also balanced, and by [9, Lemma 2.5], we have $|V_1| = O(n)$.
2. We turn T_1 to a binary tree-decomposition $T_2 = (V_2, E_2)$, by a standard tree-binarization process [16, Fact 3], which increases the size and the height of T_2 by at most a factor 2.
3. We construct a tree-decomposition $T_3 = (V_3, E_3)$ by partitioning T_2 to disjoint connected components of size between $\frac{k}{2}$ and k each (the last component might have size less than $\frac{k}{2}$) and contracting each such component to a single bag in T_3. Since T_2 is smooth, the number of nodes in the union of the bags of each component is at most $2 \cdot k$. Hence the width of T_3 is $O(k)$. The partitioning is done as follows. We traverse T_2 bottom-up and group bags into components in a greedy way. In particular, given that the traversal is on a current bag B, we keep track of the number of bags i_B below B (not including B) that have not been grouped to a component yet. The first time we find $i_B \geq t$, let B' be the child of B with the largest number $i_{B'}$ among the children of B. We group B' and its ungrouped descendants into a new component C, and continue with the traversal. Observe that the size of C is $\frac{k}{2} \leq |C| < k$.
4. Finally, we construct $\mathrm{Tree}'(G)$ by turning T_3 to a binary tree-decomposition as in Step 2.

Note that all steps above require $O(n \cdot t)$ time. The desired result follows. ◀

▶ **Lemma 5** ([16]). *Given a weighted graph $G = (V, E, \mathsf{wt})$ of treewidth t and a tree-decomposition $T = (V_T, E_T)$ of G of width $O(t)$, we can compute for all bags $B \in V_T$ a* local distance map $\mathsf{LD}_B : B \times B \to \mathbb{Z}$ *with $\mathsf{LD}_B(u, v) = d(u, v)$ in total time $O(|V_T| \cdot t^3)$ and space $O(|V_T| \cdot t^2)$.*

Model and word tricks. We consider the standard RAM model with word size $W = \Theta(\log n)$, where $\mathsf{poly}(n)$ is the size of the input. Our reachability algorithm (in Section 3) uses so called "word tricks" heavily. We use constant-time LCA queries which also use word tricks [25, 6].

3 Optimal Reachability for Low-Treewidth Graphs

In this section we present algorithms for building and querying a data-structure $\mathsf{Reachability}$, which handles single-source and pair reachability queries over an input a graph G of n nodes and treewidth t. In particular, we establish the following.

▶ **Theorem 6.** *Given a graph G of n nodes and treewidth t, let $\mathcal{T}(G)$ be the time and $\mathcal{S}(G)$ be the space required for constructing a balanced tree-decomposition $\mathrm{Tree}(G)$ of $O(n)$ bags and width $O(t)$ on the standard RAM with wordsize $W = \Theta(\log n)$. The data-structure* $\mathsf{Reachability}$ *correctly answers reachability queries and requires*
1. *$O(\mathcal{T}(G) + n \cdot t^2)$ preprocessing time;*
2. *$O(\mathcal{S}(G) + n \cdot t)$ preprocessing space;*
3. *$O\left(\left\lceil \frac{t}{\log n} \right\rceil\right)$ pair query time; and*
4. *$O\left(\frac{n \cdot t}{\log n}\right)$ single-source query time.*

For constant-treewidth graphs we have that $\mathcal{T}(G) = O(n)$ and $\mathcal{S}(G) = O(n)$ ([12, Lemma 2]), and thus along with Theorem 6 we obtain the following corollary.

▶ **Corollary 7.** *Given a graph G of n nodes and constant treewidth, the data-structure* Reachability *requires $O(n)$ preprocessing time and space, and correctly answers (i) pair reachability queries in $O(1)$ time, and (ii) single-source reachability queries in $O\left(\frac{n}{\log n}\right)$ time.*

Intuition. Informally, the preprocessing consists of first obtaining a small, balanced and binary tree-decomposition T of G, and computing the local reachability information in each bag B (i.e., the pairs $(u, v) \in E^*$ with $u, v \in B$) using Lemma 5. Then, the whole of preprocessing is done on T, by constructing two types of sets, which are represented as bit sequences and packed into words of length $W = \Theta(\log n)$. Initially, every node u receives an *index* i_u, such that for every bag B, the indices of nodes whose root bag is in $T(B)$ form a contiguous interval. Additionally, for every appearance of node u in a bag B, the node u receives a *local index* l_u^B in B. For brevity, a sequence $(A^0, A^1, \dots A^k)$ will be denoted by $(A^i)_{0 \leq i \leq k}$. When k is implied, we simply write $(A^i)_i$. The following two types of sets are constructed.

1. Sets that store information about subtrees. Specifically, for every node u, the set F_u stores the relative indices of nodes v that can be reached from u, and whose root bag is in $T(B_u)$. These sets are used to answer single-source queries.
2. Sets that store information about ancestors. Specifically, for every node u, two sequences of sets are stored $(\mathsf{F}_u^i)_{0 \leq i \leq \mathsf{Lv}(u)}$, $(\mathsf{T}_u^i)_{0 \leq i \leq \mathsf{Lv}(u)}$, such that F_u^i (resp., T_u^i) contains the local indices of nodes v in the ancestor bag B_u^i of B_u at level i, such that $(u, v) \in E^*$ (resp., $(v, u) \in E^*$). These sets are used to answer pair queries.

The sets of the first type are constructed by a bottom-up pass, whereas the sets of the second type are constructed by a top-down pass. Both passes are based on the separator property of tree decompositions (recall Lemma1 and Lemma 2), which informally states that reachability properties between nodes in distant bags will be captured transitively, through nodes in intermediate bags.

Reachability Preprocessing. We now give a formal description of the preprocessing of Reachability that takes as input a graph G of n nodes and treewidth t, and a balanced tree-decomposition $T = \text{Tree}(G)$ of width $O(t)$. After the preprocessing, Reachability supports single-source and pair reachability queries. We say that we "insert" set A to set A' meaning that we replace A' with $A \cup A'$. Sets are represented as bit sequences where 1 denotes membership in the set, and the operation of inserting a set A "at the i-th position" of a set A' is performed by taking the bit-wise logical OR between A and the segment $[i, i + |A|]$ of A'. The preprocessing consists of the following steps.

1. Turn T to a small, balanced binary tree-decomposition of G of width $O(t)$, using Lemma 4.
2. Preprocess T to answer LCA queries in $O(1)$ time [25].
3. Compute the local distance map $\mathsf{LD}_B : B \times B \to \mathbb{Z}$ for every bag B w.r.t reachability, i.e., for any bag B and nodes $u, v \in B$, we have $\mathsf{LD}_B(u, v) = 1$ iff $(u, v) \in E^*$.
4. Apply a pre-order traversal on T, and assign an incremental index i_u to each node u at the time the root bag B of u is visited. If there are multiple nodes u for which B is the root bag, assign the indices to those nodes in some arbitrary order. Additionally, store the number s_u of nodes whose root bag is in $T(B)$ and have index at least i_u. Finally, for each bag B and $u \in B$, assign a unique local index l_u^B to u, and store in B the number of nodes (with multiplicities) a_B contained in all ancestors of B, and the number b_B of nodes in B.
5. For every node u, initialize a bit set F_u of length s_u, pack it into words, and set the first bit to 1.

6. Traverse T bottom-up, and for every bag B execute the following step. For every pair of nodes $u, v \in B$ such that B is the root bag of v and $i_u < i_v$ and $\mathsf{LD}_B(u,v) = 1$, insert F_v to the segment $[i_v - i_u, i_v - i_u + s_v]$ of F_u (the nodes reachable from v now become reachable from u, through v).
7. For every node u initialize two sequences of bit sets $(\mathsf{T}_u^i)_{0 \leq i \leq \mathsf{Lv}(u)}$, $(\mathsf{F}_u^i)_{0 \leq i \leq \mathsf{Lv}(u)}$, and pack them into consecutive words. Each set T_u^i and F_u^i has size $b_{B_u^i}$, where B_u^i is the ancestor of B_u at level i.
8. Traverse T top-down, and for B the bag currently visited, for every node $x \in B$, maintain two sequences of bit sets $(\overline{\mathsf{T}}_x^i)_{0 \leq i \leq \mathsf{Lv}(B)}$ and $(\overline{\mathsf{F}}_x^i)_{0 \leq i \leq \mathsf{Lv}(B)}$. Each set $\overline{\mathsf{T}}_x^i$ and $\overline{\mathsf{F}}_x^i$ has size b_{B^i}, where B^i is the ancestor of B at level i. Initially, B is the root of T (hence $\mathsf{Lv}(B) = 0$), and set the position l_w^B of $\overline{\mathsf{F}}_x^0$ (resp., $\overline{\mathsf{T}}_x^0$) to 1 for every node w such that $\mathsf{LD}_B(x,w) = 1$ (resp., $\mathsf{LD}_B(w,x) = 1$). For each other bag B encountered in the traversal, do as follows. Let $S = B \cap B'$, where B' is the parent of B in T, and let x range over S.
   a. For each node x, create a set $\overline{\mathsf{T}}_x$ (resp., $\overline{\mathsf{F}}_x$) of 0s of length b_B, and for every $w \in B$ such that $\mathsf{LD}_B(x,w) = 1$ (resp., $\mathsf{LD}_B(w,x) = 1$), set the l_w^B-th bit of $\overline{\mathsf{F}}_x$ (resp., $\overline{\mathsf{T}}_x$) to 1. Append the set $\overline{\mathsf{T}}_x$ (resp., $\overline{\mathsf{F}}_x$) to $(\overline{\mathsf{T}}_x^i)_i$ (resp., $(\overline{\mathsf{F}}_x^i)_i$). Now each set sequence $(\overline{\mathsf{T}}_x^i)_i$ and $(\overline{\mathsf{F}}_x^i)_i$ has size $a_B + b_B$.
   b. For each $u \in B$ whose root bag is B, initialize set sequences $(\overline{\mathsf{F}}_u^i)_i$ and $(\overline{\mathsf{T}}_u^i)_i$ with 0s of length $a_B + b_B$ each, and set the bit at position l_u^B of $\overline{\mathsf{F}}_u^{\mathsf{Lv}(B)}$ and $\overline{\mathsf{T}}_u^{\mathsf{Lv}(B)}$ to 1. For every $w \in B$ with $\mathsf{LD}_B(u,w) = 1$ (resp., $\mathsf{LD}_B(w,u) = 1$), insert $(\overline{\mathsf{F}}_w^i)_i$ to $(\overline{\mathsf{F}}_u^i)_i$ (resp., $(\overline{\mathsf{T}}_w^i)_i$ to $(\overline{\mathsf{T}}_u^i)_i$). Finally, set $(\mathsf{F}_u^i)_i$ equal to $(\overline{\mathsf{F}}_u^i)_i$ (resp., $(\mathsf{T}_u^i)_i$ equal to $(\overline{\mathsf{T}}_u^i)_i$).

Figure 1 illustrates the constructed sets on a small example.

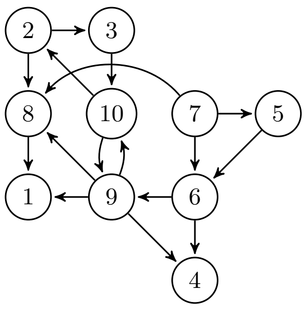
It is fairly straightforward that at the end of the preprocessing, the i-th position of each set F_u is 1 only if $(u,v) \in E^*$, where v is such that $i_v - i_u = i$. The following lemma states the opposite direction, namely that each such i-th position will be 1, as long as the path $P : u \rightsquigarrow v$ only visits nodes with certain indices.

▶ **Lemma 8.** *At the end of preprocessing, for every pair of nodes u and v with $i_u \leq i_v \leq i_u + s_u$, if there exists a path $P : u \rightsquigarrow v$ such that for every $w \in P$, we have $i_u \leq i_w \leq i_u + s_u$, then the $(i_v - i_u)$-th bit of F_u is 1.*

Proof. We prove inductively the following claim. For every ancestor B of B_v, if there exists $w \in B$ and a path $P_1 : w \rightsquigarrow v$, then exists $x \in B \cap P_1$ such that $i_x \leq i_v \leq i_x + s_x$ and the $i_v - i_x$-th bit of F_x is 1. The proof is by induction on the length of the simple path $P_2 : B \rightsquigarrow B_v$.

1. If $|P_2| = 0$, the statement is true by taking $x = v$, since the 0-th bit of F_v is 1.
2. If $|P_2| > 0$, examine the child B' of B in P_2. By Lemma 2, there exists $x \in B \cap B' \cap P$, and let $P_3 : x \rightsquigarrow v$. By the induction hypothesis there exists some $y \in B' \cap P_3$ with $i_y \leq i_v \leq i_y + s_y$ and the $i_v - i_y$-th bit of F_y is 1. If $y \in B$, we take $x = y$. Otherwise, B' is the root bag of y, and by the local distance computation of Lemma 5, it is $\mathsf{LD}_{B'}(x,y) = 1$. By the choice of x, y we have that B_x is an ancestor of B_y. Thus, by construction we have $i_x < i_y$ and $s_x \geq s_y + i_y - i_x$, and hence $i_x \leq i_v \leq i_x + s_x$. Then in step 5, F_y is inserted in position $i_y - i_x$ of F_x, thus the bit at position $i_y - i_x + i_v - i_y = i_v - i_x$ of F_x will be 1, and we are done.

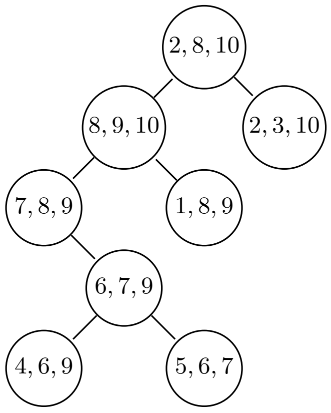
When B_u is examined, by the above claim there exists $x \in P$ such that $i_x \leq i_v$ and the $i_v - i_x$-th bit of F_x is 1. If $x = u$ we are done. Otherwise, by the choice of P, we have $i_u < i_x$, which can only happen if B_u is also the root bag of x. Then in step 5, F_x is inserted



(a)

u	i_u	Bit-set F_u									
		0	1	2	3	4	5	6	7	8	9
2	0	1	1	1	1	0	0	1	0	1	1
8	1		1	0	0	0	0	0	0	0	1
10	2			1	1	0	0	1	0	1	1
9	3				1	0	0	1	0	1	
7	4					1	1	1	1		
6	5						1	1	0		
4	6							1			
5	7								1		
1	8									1	
3	9										1

(b)



	$i=0$			$i=1$			$i=2$			$i=3$		
v	2	8	10	8	9	10	7	8	9	6	7	9
$l_v^{B_6^i}$	0	1	2	0	1	2	0	1	2	0	1	2
$(\mathsf{F}_6^i)_i$	1	1	1	1	1	1	0	1	1	1	0	1
$(\mathsf{T}_6^i)_i$	0	0	0	0	0	0	1	0	0	1	1	0

(c) **(d)**

Figure 1 a, c: A graph G and a tree-decomposition Tree(G). b: The sets F_u constructed from step 5 to answer single-source queries. The j-th bit of a set F_u is 1 iff $(u, v) \in E^*$, where v is such that $i_v - i_u = j$. d: The set sequences $(\mathsf{F}_u^i)_i$ and $(\mathsf{T}_u^i)_i$ constructed from step 6 to answer pair queries, for $u = 6$. For every $i \in \{0, 1, 2, 3\}$ and ancestor B_6^i of B_6 at level i, every node $v \in B_u^i$ is assigned a local index $l_v^{B_6^i}$. The j-th bit of set F_6^i (resp. T_6^i) is 1 iff $(6, v) \in E^*$ (resp. $(v, 6) \in E^*$), where v is such that $l_v^{B_6^i} = j$.

in position $i_x - i_u$ of F_u, and hence the bit at position $i_x - i_u + i_v - i_x = i_v - i_u$ of F_x will be 1, as desired. ◀

Similarly, given a node u and an ancestor bag B_u^i of B_u at level i, the j-th position of the set F_u^i (resp., T_u^i) is 1 only if $(u, v) \in E^*$ (resp., $(v, u) \in E^*$), where $v \in B_u^i$ is such that $l_v^{B_u^i} = j$. The following lemma states that the inverse is also true.

▶ **Lemma 9.** *At the end of preprocessing, for every node u, for every $v \in B_u^i$ where B_u^i is the ancestor of B_u at level i, we have that if $(u, v) \in E^*$ (resp., $(v, u) \in E^*$), then the $l_v^{B_u^i}$-th bit of F_u^i (resp., T_u^i) is 1 .*

▶ **Lemma 10.** *Given a graph G with n nodes and treewidth t, let $\mathcal{T}(G)$ be the time and $\mathcal{S}(G)$ be the space required for constructing a balanced tree-decomposition of G with $O(n)$ bags and width $O(t)$. The preprocessing phase of* Reachability *on G requires $O(\mathcal{T}(G) + n \cdot t^2)$ time and $O(\mathcal{S}(G) + n \cdot t)$ space.*

Proof. First, we construct a balanced tree-decomposition $T = \text{Tree}(G)$ of G in $\mathcal{T}(G)$ time and $\mathcal{S}(G)$ space. We establish the complexity of each preprocessing step separately.

1. Using Lemma 4, this step requires $O(n \cdot t)$ time. From this point on, T consists of $b = O(\frac{n}{t})$ bags, has height $h = O(\log n)$, and width $t' = O(t)$.

2. By a standard construction for balanced trees, preprocessing T to answer LCA queries in $O(1)$ time requires $O(b) = O(\frac{n}{t})$ time.

3. By Lemma 5, this step requires $O(b \cdot t'^3) = O(\frac{n}{t} \cdot t^3) = O(n \cdot t^2)$ time and $O(b \cdot t'^2) = O(\frac{n}{t} \cdot t^2) = O(n \cdot t)$ space.

4. Every bag B is visited once, and each operation on B takes constant time. We make $O(t')$ such operations in B, hence this step requires $O(b \cdot t') = O(n)$ time in total.

5–6. The space required in this step is the space for storing all the sets F_u of size s_u each, packed into words of length W:

$$\sum_{u \in V} \left\lceil \frac{s_u}{W} \right\rceil = \sum_{i=0}^{h} \sum_{u:\mathsf{Lv}(u)=i} \left\lceil \frac{s_u}{W} \right\rceil \leq \sum_{i=0}^{h} \sum_{u:\mathsf{Lv}(u)=i} \left(\frac{s_u}{W} + 1 \right)$$
$$= \frac{1}{W} \cdot \sum_{i=0}^{h} \sum_{u:\mathsf{Lv}(u)=i} s_u + \sum_{i=0}^{h} \sum_{u:\mathsf{Lv}(u)=i} 1 \leq \frac{1}{W} \cdot \sum_{i=0}^{h} n \cdot (t'+1) + n = O(n \cdot t)$$

since $h = O(\log n)$, $t' = O(t)$ and $W = \Theta(\log n)$. Note that we have $\sum_{u:\mathsf{Lv}(u)=i} s_u \leq n \cdot (t'+1)$ because $|\bigcup_u \mathsf{F}_u| \leq n$ (as there are n nodes) and every element of $\bigcup_u \mathsf{F}_u$ belongs to at most $t'+1$ such sets F_u (i.e., for those u that share the same root bag at level i). The time required in this step is $O(n \cdot t)$ in total for iterating over all pairs of nodes (u, v) in each bag B such that B is the root bag of either u or v, and $O(n \cdot t^2)$ for the set operations, by amortizing $O(t)$ operations per word used.

7. The time and space required for storing each sequence of the sets $(\mathsf{F}_u^i)_{0 \leq i \leq \mathsf{Lv}(u)}$ and $(\mathsf{T}_u^i)_{0 \leq i \leq \mathsf{Lv}(u)}$ is:

$$\sum_{u \in V} 2 \cdot \left\lceil \frac{a_{B_u} + b_{B_u}}{W} \right\rceil \leq 2 \cdot n \cdot \left\lceil \frac{(t'+1) \cdot h}{W} \right\rceil = O(n \cdot t)$$

since $a_{B_u} + b_{B_u} \leq (t'+1) \cdot h$, $h = O(\log n)$ and $W = \Theta(\log n)$.

8. The space required is the space for storing the set sequences $(\overline{\mathsf{T}}_v^i)_i$ and $(\overline{\mathsf{F}}_v^i)_i$, which is $O(t^2)$ by a similar argument as in the previous item. The time required is $O(t)$ for initializing every new set sequence $(\overline{\mathsf{T}}_u^i)_i$ and $(\overline{\mathsf{F}}_u^i)_i$ and this will happen once for each node u at its root bag B_u, hence the total time is $O(n \cdot t)$. ◀

Reachability Querying. We now turn our attention to the querying phase.

Pair query. Given a pair query (u, v), find the LCA B of bags B_u and B_v. Obtain the sets $\mathsf{F}_u^{\mathsf{Lv}(B)}$ and $\mathsf{T}_v^{\mathsf{Lv}(B)}$ of size b_B. Each set starts in bit position a_B of the corresponding sequence $(\mathsf{F}_u^i)_i$ and $(\mathsf{T}_v^i)_i$. Return True iff the logical-AND of $\mathsf{F}_u^{\mathsf{Lv}(B)}$ and $\mathsf{T}_v^{\mathsf{Lv}(B)}$ contains an entry which is 1.

Single-source query. Given a single-source query u, create a bit set A of size n, initially all 0s. For every node $x \in B_u$ with $i_x \leq i_u$, if the $l_x^{B_u}$-th bit of $\mathsf{F}_u^{\mathsf{Lv}(u)}$ is 1, insert F_x to the segment $[i_x, i_x + s_x]$ of A. Then traverse the path from B_u to the root of T, and let B_u^i be the ancestor of B_u at level $i < \mathsf{Lv}(B_u)$. For every node $x \in B_u^i$, if the $l_x^{B_u^i}$-th bit of F_u^i is 1, set the i_x-th bit of A to 1. Additionally, if B_u^i has two children, let B be the child of B_u^i that is not ancestor of B_u, and $j_{\min}$ and $j_{\max}$ the smallest and largest indices, respectively, of nodes whose root bag is in $T(B)$. Insert the segment $[j_{\min} - i_x, j_{\max} - i_x]$ of F_x to the segment $[j_{\min}, j_{\max}]$ of A. Report that the nodes v reached from u are those v for which the i_v-th bit of A is 1.

The following lemma establishes the correctness and complexity of the query phase.

▶ **Lemma 11.** *After the preprocessing phase of* Reachability*, pair and single-source reachability queries are answered correctly in* $O\left(\left\lceil\frac{t}{\log n}\right\rceil\right)$ *and* $O\left(\frac{n\cdot t}{\log n}\right)$ *time respectively.*

Proof. Let $t' = O(t)$ be the width of the small tree-decomposition constructed in Step 1. The correctness of the pair query comes immediately from Lemma 9 and Lemma 1, which implies that every path $u \rightsquigarrow v$ must go through the LCA of B_u and B_v. The time complexity follows from the $O\left(\left\lceil\frac{t}{W}\right\rceil\right)$ word operations on the sets $\mathsf{F}_u^{\mathsf{Lv}(B)}$ and $\mathsf{T}_v^{\mathsf{Lv}(B)}$ of size $O(t)$ each.

Now consider the single-source query from a node u and let v be any node such that there is a path $P : u \rightsquigarrow v$. Let B be the LCA of B_u, B_v, and by Lemma 1, there is a node $y \in B \cap P$. Let x be the last such node in P, and let $P' : x \rightsquigarrow v$ be the suffix of P from x. It follows that P' is a path such that for every $w \in P'$ we have $i_x \leq i_w \leq i_x + s_x$.

1. If B_v is an ancestor of B_u, then necessarily $x = v$, and by Lemma 9, the l_v^B-th bit of $\mathsf{F}_u^{\mathsf{Lv}(B)}$ is 1. Then the algorithm sets the i_v-th bit of A to 1.
2. Else, B_x is an ancestor of B_v (recall that a bag is an ancestor of itself), and by Lemma 8, the $(i_v - i_x)$-th bit of F_x is 1.
   a. If B is B_u, the algorithm will insert F_x to the segment $[i_x, i_x + s_x]$ of A, thus the $i_x + i_v - i_x = i_v$-th bit of A is set to 1.
   b. If B is not B_u, it can be seen that $j_{\min} \leq i_v \leq j_{\max}$, where $j_{\min}$ and $j_{\max}$ are the smallest and largest indices of nodes whose root bag is in $T(B')$, with B' the child of B that is not ancestor of B_u. Since the $(i_v - i_x)$-th bit of F_x is 1, the $(i_v - j_{\min})$-th bit of the $[j_{\min}, j_{\max}]$ segment of F_x is 1, thus the $j_{\min} + i_v - j_{\min} = i_v$-th bit of A is set to 1.

Regarding the time complexity, the algorithm performs $O(h \cdot t') = O(h \cdot t)$ set insertions to A. For every position j of A, the number of such set insertions that overlap on j is at most $t' + 1$ (once for every node in the LCA of B_u and B_v, where v is such that $i_v = j$). Hence if H_i is the size of the i-th insertion in A, we have $\sum_i H_i \leq n \cdot (t' + 1)$. Since the insertions are word operations, the total time spent for the single source query is

$$\sum_{i=0}^{h} \left\lceil \frac{H_i}{W} \right\rceil \leq h + \sum_{i=0}^{h} \frac{H_i}{W} \leq h + \frac{n \cdot (t'+1)}{W} = O\left(\frac{n \cdot t}{\log n}\right)$$

since $h = O(\log n)$, $t' = O(t)$ and $W = \Theta(\log n)$. ◀

4 Space vs Query Time Tradeoff for Sub-linear Space

In this section we present the data-structure LowSpDis, for low-space distance queries. Our results make use of the following lemma.

▶ **Lemma 12** ([16]). *Consider a weighted graph* $G = (V, E, \mathsf{wt})$ *of* n *nodes and constant-treewidth, and a tree-decomposition* T *of* G *of* $O(n)$ *nodes and constant width be given. There exists a data-structure* DistanceLP *that answers distance queries on* G *and requires*
1. $O(n)$ *preprocessing time and space; and*
2. $O(\alpha(n))$ *pair query time.*

Throughout this section we fix a constant $\epsilon \in [\frac{1}{2}, 1]$. The main idea is to partition the initial tree-decomposition T to sufficiently large components, and discard all bags that don't appear in the boundary of their component. We use Lemma 12 to preprocess $\overline{T}$ and the induced graph. Answering a pair query (u, v) is performed similarly as in Lemma 12, but requires additional time for processing the components in which u and v appear (since they have not been preprocessed). The challenge comes in performing these computations within the targeted space and time bounds. We establish the following theorem.

▶ **Theorem 13.** *Let (1) a constant $\epsilon \in [\frac{1}{2}, 1]$; and (2) a weighted graph $G = (V, E, \mathsf{wt})$ with n nodes and of constant treewidth, be given. The data structure* LowSpDis *correctly answers pair distance queries on G and requires*
1. *Polynomial in n preprocessing time;*
2. *$O(n^{\epsilon})$ working space; and*
3. *$O(n^{1-\epsilon} \cdot \alpha(n))$ pair query time.*

▶ Remark. The data-structure LowSpDis accesses the graph in the input space, i.e., the graph and is not counted for the working space bound of LowSpDis.

Informal description. Here we outline the key steps required for LowSpDis to achieve the bounds stated in Theorem 13. The preprocessing consists of the following conceptual steps.
1. A binary tree-decomposition $T = \text{Tree}(G)$ of $O(n)$ bags is constructed in polynomial time and logarithmic space, using [20]. Hence, LowSpDis does not store T explicitly, but uses the logspace construction of [20] to traverse T and access its bags.
2. A tree-partitioning algorithm LowSpTreePart is used to partition T into $O(n^{1-\epsilon})$ components C of size $O(n^{\epsilon})$ each. A key point in this construction is that every such component C contains a constant number of bags on its boundary.
3. Given a list of components $\mathcal{C} = (C_1, \dots, C_\ell)$ constructed in the previous step, a tree of bags called *summary tree* $\overline{T}$ is constructed. The summary tree occurs by contracting every component C_i of T to a single bag $\mathcal{B}_i$. Moreover, $\mathcal{B}_i$ contains precisely the nodes that appear in the bags of the boundary of C_i. Since there are $O(1)$ such bags for every component, each $\mathcal{B}_i$ has constant size. The key point in this step is that $\overline{T}$ is a tree-decomposition of G restricted on the nodes that appear in bags of $\overline{T}$. Moreover, $\overline{T}$ has size $O(n^{1-\epsilon})$ instead of $O(n)$, which is the size of the initial tree-decomposition T.
4. Since $\overline{T}$ is a tree-decomposition, Lemma 12 applies to preprocess $\overline{T}$ in the stated bounds.
5. An algorithm LowSpLD is used to compute the distance $d(u, v)$ between any pair of nodes u, v that appear together in some boundary bag of a component C_i. This is achieved by traversing T in a particular way, and applying a standard, linear-space computation on each component C_i separately. Since $|C_i| = O(n^{\epsilon})$, this requires $O(n^{\epsilon})$ space. Since the boundary bags of C_i are constantly many, the algorithm only needs to store constant-size information per component, and thus $O(n^{1-\epsilon}) = O(n^{\epsilon})$ information in total.
6. Finally, given a node u, it is crucial to obtain the set V_u of nodes that u can reach going through nodes v that appear in bags of $\overline{T}$. Moreover, this set needs to be obtained in linear time in the size of the component, i.e., $O(n^{1-\epsilon})$. This is achieved by a graph traversal on G starting from u, in combination with perfect hashing for testing in $O(1)$ time whether a node v appears in bags of $\overline{T}$.

A query u, v is answered by LowSpDis using the following conceptual steps.
1. First, the algorithm retrieves the sets V_u and V_v. If $v \in V_u$, then the distance $d(u, v)$ is retrieved by constructing a tree-decomposition T_u of $G[V_u]$, and using standard methods for solving the problem in T_u, in $O(n^{\epsilon})$ time. Similarly if $u \in V_v$.
2. If $v \notin V_u$ and $u \notin V_v$, then the algorithm again constructs the tree-decompositions T_u and T_v of $G[V_u]$ and $G[V_v]$ respectively. The algorithm retrieves two bags $\mathcal{B}_u$ and $\mathcal{B}_v$ of $\overline{T}$ with $\mathcal{B}_u \subseteq V_u$ and $\mathcal{B}_v \subseteq V_v$, and uses the standard methods of the previous item to obtain the distances $d(u, x)$ and $d(y, v)$, for every node $x \in \mathcal{B}_u$ and $\mathcal{B}_v$. Additionally, the

algorithm uses Lemma 12 to obtain the distance $d(x, y)$ between every such pair x, y. Finally, the algorithm returns the value $\min_{x\in\mathcal{B}_u, y\in\mathcal{B}_v}(d(u, x) + d(x, y) + d(y, v))$.

In the remaining of this section we describe in detail the above phases of LowSpDis.

Tree partitioning: The algorithm LowSpTreePart. We first describe algorithm LowSpTreePart, which operates on a binary tree-decomposition $T = (V_T, E_T)$ of $O(n)$ bags. Given a constant ϵ, LowSpTreePart splits T to $O(n^{1-\epsilon})$ connected components $C \subseteq V_T$ of size $|C| = O(n^\epsilon)$. Each component C is implicitly represented as a list of bags $C(B_1, \dots, B_k)$, which mark the boundaries of C in T. The *root* of $C(B_1, \dots, B_k)$ is $B = \arg\min_{B_i} \mathsf{Lv}(B_i)$, i.e., the smallest-level bag among all B_i. We will consider w.l.o.g. that B_1 is always the root bag of component $C(B_1, \dots, B_k)$. A bag B' belongs to C iff the $\mathsf{Lv}(B') \geq \mathsf{Lv}(B_1)$ and the unique simple path $B \rightsquigarrow B_1$ in T does not contain any of the B_i as intermediate bags.

The algorithm traverses T in post-order, and maintains a two variables $x, y \in \mathbb{N}$, that represent the size of the current component C and the number of components that appear directly below C. As the algorithm backtracks to a bag B, it updates $x = x_1 + x_2 + 1$ and $y = y_1 + y_2$, where x_i, y_i is the pair corresponding to the child B'_i of B (recall that T is binary), or sets $x = x_1 + 1$ and $y = y_1$ if B has only one child B'_1. If $x \geq n^\epsilon$ or $y \geq 3$, the algorithm creates a new component $C(B_1, \dots, B_k)$, where B_1 is the current bag B, and $B_2, \dots, B_k$ are parents of roots of components that have been constructed already (or leaves of T). Finally, the algorithm sets $x = 0$ and $y = 1$, and proceeds to the parent of B.

▶ **Lemma 14.** LowSpTreePart *constructs* $O(n^{1-\epsilon})$ *components. For every constructed component* $C(B_1, \dots, B_k)$ *we have* $|C| \leq 2 \cdot n^\epsilon - 1$ *and* $k \leq 5$.

Proof. If $|C| > 2 \cdot n^\epsilon - 1$, then, before backtracking to B_1, the algorithm examined a child B of B_1 with value $x \geq j$, and thus would have grouped B and B_1 in different components. It is easy to see that every root of a component appears in the same component with its children, a contradiction. A similar argument holds for showing that $k \leq 5$. We now argue that LowSpTreePart constructs $O(n^{1-\epsilon})$ components. We say that the algorithm "performs a type A cut" and "performs a type B cut" when it constructs a component based on the criterion $x \geq j$ and $y \geq 3$ respectively. Let X and Y be the number of type A and type B cuts. Every type A cut constructs a component of size at least j, hence $X = O(n^{1-\epsilon})$. Additionally, we have $Y \leq X$, hence $X + Y = O(n^{1-\epsilon})$, as desired. To see that $Y \leq X$, let Z be a counter that counts the sum of the y values that LowSpTreePart maintains at any point in the traversal. Observe that a type A cut increases Z by at most one, and a type B cut decreases Z by at least one. Since Z is always non-negative, we have that there is at least one type A cut for each type B cut, thus $Y \leq X$. The desired result follows. ◀

We denote by $\mathsf{Root}(C)$ the root bag of a component C. Given two components C_1, C_2 constructed by LowSpTreePart, we say that C_1 is the *parent* of C_2 if $\mathsf{Root}(C_1)$ is the lowest ancestor of $\mathsf{Root}(C_2)$ among all bags that appear as roots in some component. In such case, C_2 is a *child* of C_1. Given a component C that is the parent of components $C_1, \dots, C_i$, we let $\mathsf{Merge}(C) = C \cup \bigcup_j C_j$.

The summary tree construction SummaryTree. Let $\mathcal{C} = (C_1, \dots, C_\ell) = \mathsf{LowSpTreePart}(T)$ be the list of components that LowSpTreePart returns, where each component is implicitly represented by the bags of its boundary, i.e., $C_i = C_i(B_1^i, \dots, B_{k_i}^i)$. We construct a *summary tree* of bags $\overline{T} = \mathsf{SummaryTree}(\mathcal{C}) = (\overline{V}, \overline{E})$ as follows.

1. $\overline{V}$ consists of bags $\mathcal{B}_i$ for $1 \leq i \leq \ell$, where $\mathcal{B}_i = B_1^i \cup \cdots \cup B_{k_i}^i$, i.e., $\mathcal{B}_i$ is the union of all bags in the boundary of C_i.
2. We have $(\mathcal{B}_i, \mathcal{B}_j) \in \overline{E}$ if C_i is a parent of C_j.

The following lemma follows easily from Lemma 14 and the above construction.

▶ **Lemma 15.** *Let $V_S = \bigcup_{\mathcal{B}_i \in \overline{V}} \mathcal{B}_i$ be the set of nodes of G that appear in bags of the summary tree $\overline{T}$. Then $\overline{T}$ is a tree-decomposition of the graph $G[V_S]$ induced by V_S. $\overline{T}$ has $O(n^{1-\epsilon})$ bags and constant width.*

Local distance computation in low space LowSpLD. Let $\mathcal{C} = (C_1, \dots, C_\ell) = \mathsf{LowSpTreePart}(T)$ be the list of components constructed by LowSpTreePart. We describe algorithm LowSpLD, which computes the distance $d(u, v)$ between any pair of nodes u, v that appear in the root bag $\mathsf{Root}(C_i)$ of some component C_i. Let $T_i = \mathrm{Tree}(G)[\mathsf{Merge}(C_i)]$ be the subtree of $\mathrm{Tree}(G)$ restricted in the bags of component C_i and its children components, and $V_i = \bigcup_{B \in \mathsf{Merge}(C_i)} B$ the set of nodes that appear in bags of $\mathsf{Merge}(C_i)$. It is easy to verify that T_i is a subtree of T, and thus a tree decomposition of the graph $G[V_i] = (V_i, E_i)$ induced by V_i. The algorithm LowSpLD operates as follows. For every component C, it maintains a local distance map $\mathsf{LD}_{\mathsf{Root}(C)} : \mathsf{Root}(C) \times \mathsf{Root}(C) \to \mathbb{Z}$. Initially, $\mathsf{LD}_{\mathsf{Root}(C)}(u, v) = \mathsf{wt}(u, v)$ for every component C and pair of nodes $u, v \in \mathsf{Root}(C)$. Then, LowSpLD performs the following two passes.

1. Traverse $\overline{T}$ bottom-up, and for every encountered bag $\mathcal{B}$ that corresponds to component C, let $C_1, \dots, C_k$ be the children components of C. Obtain the tree-decomposition T_i, and construct a weight function $\mathsf{wt}_i : E_i \to \mathbb{Z}$ defined as follows:

$$\mathsf{wt}_i(u, v) = \begin{cases} \mathsf{LD}_{\mathsf{Root}(C)}(u, v) & \text{if } u, v \in \mathsf{Root}(C) \\ \mathsf{LD}_{\mathsf{Root}(C_i)}(u, v) & \text{if } u, v \in \mathsf{Root}(C_i) \text{ for some } 1 \leq i \leq k \\ \mathsf{wt}(u, v) & \text{otherwise} \end{cases}$$

   and execute the local distance computation of Lemma 5 Afterwards, update $\mathsf{LD}_{\mathsf{Root}(C)}$ and $\mathsf{LD}_{\mathsf{Root}(C_i)}$ for all $1 \leq i \leq k$ with the newly discovered distances.
2. Traverse $\overline{T}$ top-down, and for every encountered bag $\mathcal{B}$ execute the steps of Step 1.

▶ **Lemma 16.** *At the end of* LowSpLD*, for every component C and nodes $u, v \in \mathsf{Root}(C)$ we have $\mathsf{LD}_{\mathsf{Root}(C)}(u, v) = d(u, v)$. Moreover,* LowSpLD *operates in $O(n^\epsilon)$ space and polynomial time.*

Proof. The correctness of LowSpLD follows straightforwardly from Lemma 5 and Lemma 3. Since T has constant width, the size of each local distance map $\mathsf{LD}_{\mathsf{Root}(C)}$ has constant size. Hence the space used by the algorithm is asymptotically the space required for storing $\overline{T}$, plus the space for constructing each tree-decomposition T_i. By Lemma 15 the former requires $O(n^{1-\epsilon})$ space, while by Lemma 14 the latter $O(n^\epsilon)$ space. Since $\epsilon \geq \frac{1}{2}$, we conclude that the space usage is $O(n^\epsilon)$. The polynomial time bound follows from the space bound. ◀

Fast component retrieval GetCompNodes. Given a node u of G, we are interested in retrieving the set V_u of nodes that u can reach in G without going through nodes v that appear in bags of $\overline{T}$. The desired set V_u can be obtained in $O(n^\epsilon)$ time by a performing any standard graph traversal on G starting from u, and making sure that the traversal never expands a node v that appears in the bags of $\overline{T}$. This can be done if testing whether v appears in any of the bags of $\overline{T}$ can be performed in constant time. Let $V_S = \bigcup_{\mathcal{B}_i \in \overline{V}} \mathcal{B}_i$ be the set of all such nodes, and $k = |V_S| = O(n^{1-\epsilon})$. We cannot store V_S as a standard bit-set which allows $O(1)$ membership testing, as this would require linear space (i.e., beyond our space bound $O(n^\epsilon)$). The problem can be solved using standard techniques from perfect hashing to store the set V_S. In the query phase, given a node u, GetCompNodes detects that $u \in V_S$ by testing whether u equals its entry in the hash table.

LowSpDis Preprocessing. We now describe the preprocessing phase of LowSpDis. The input is a weighted graph $G = (V, E, \mathsf{wt})$ of constant treewidth, and a constant $\epsilon \in [\frac{1}{2}, 1]$.

1. Construct a binary tree-decomposition $T = \text{Tree}(G)$ in logspace [20].
2. Use LowSpTreePart to construct a list of components $\mathcal{C} = (C_1, \dots, C_\ell) = \mathsf{LowSpTreePart}(T)$, with $\ell = n^{1-\epsilon}$ (i.e., LowSpTreePart is executed with $j = n^\epsilon$).
3. Construct the local distance maps $\mathsf{LD}_{\mathsf{Root}(C)}$ using LowSpLD.
4. Construct the summary tree $\overline{T} = \mathsf{SummaryTree}(\mathcal{C}) = (\overline{V}, \overline{E})$. For every component C_i that corresponds to $\mathcal{B}_i$ in $\overline{T}$, find a node $z \not\in \mathcal{B}_i$ that appears in bags of C_i, and associate z with $\mathcal{B}_i$.
5. Use Lemma 12 to build a data-structure DistanceLP on $G[V_S]$ and $\overline{T}$.
6. Let $V_S = \bigcup_{\mathcal{B}_i \in \overline{V}} \mathcal{B}_i$ be the set of nodes of G that appear in bags of the summary tree $\overline{T}$. Construct the data-structure GetCompNodes on V_S.

LowSpDis Querying. We now turn our attention to the query phase of LowSpDis.

1. Use the data-structure GetCompNodes to construct the sets V_u and V_v.
2. Construct the tree-decompositions T_u and T_v of the graphs $G[V_u]$ and $G[V_v]$ induced by V_u and V_v. This is done using some standard linear-time algorithm, e.g. [12, Lemma 2]. If $u \in V_v$, insert u to every bag of T_v, and use Lemma 5 to obtain the distance $d(u, v)$. Similarly if $v \in V_u$.
3. If $u \not\in V_v$ and $v \not\in V_u$ let $\mathcal{B}_u$ be the unique bag of $\overline{T}$ with that is associated with a node $z_u \in V_u$, and $\mathcal{B}_v$ the unique bag of $\overline{T}$ that is associated with a node $z_v \in V_v$. Insert every node of $\mathcal{B}_u$ in every bag of T_u, and every node of $\mathcal{B}_v$ in every bag of T_u, and use Lemma 5 to obtain the distances $d(u, x)$ and $d(y, v)$ for every node $x \in \mathcal{B}_u$ and $y \in \mathcal{B}_v$. Return the value $\min_{x \in \mathcal{B}_u, y \in \mathcal{B}_v}(d(u, x) + d(x, y) + d(y, v))$ where for every pair x, y the distance $d(x, y)$ is obtained by querying DistanceLP.

Proof of Theorem 13. It is clear from Lemma 12, Lemma 14, Lemma 15 and Lemma 16 that the preprocessing of LowSpDis requires polynomial time and $O(n^\epsilon)$ space, where $\epsilon \geq \frac{1}{2}$. In the query phase, LowSpDis uses $O(n^\epsilon)$ time and space for extracting the sets V_u and V_v, since each has size $O(n^\epsilon)$. Using a linear time and space algorithm for constructing the tree-decompositions T_u and T_v, this step also requires $O(n^{1-\epsilon})$ time and space. If $u \in V_v$ or $v \in V_u$, applying Lemma 5 on T_u and T_v is also done in $O(n^{1-\epsilon})$ time and space.

If $u \not\in V_v$ and $v \not\in V_u$, note that by Lemma 15 $\mathcal{B}_u$ and $\mathcal{B}_v$ have constant size, hence after inserting every node of $\mathcal{B}_u$ to every bag of T_u and every node of $\mathcal{B}_v$ to every bag of T_v, T_u and T_v still have constant width. Hence all distances $d(u, x)$ and $d(v, y)$ can be obtained using Lemma 5 in $O(n^{1-\epsilon})$ time and space. Finally, DistanceLP will be queried for the distances $d(x, y)$ of a constant number of pairs x, y, and by Lemma 12, all such queries can be served in $O(n^{1-\epsilon} \cdot \alpha(n))$ time. ◀

References

1 Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In *SIGMOD'13*, SIGMOD'13, pages 349–360, 2013.

2 Takuya Akiba, Christian Sommer, and Ken-ichi Kawarabayashi. Shortest-Path Queries for Complex Networks: Exploiting Low Tree-width Outside the Core. In *EDBT*, pages 144–155, 2012.

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 Reinhard Bauer, Tobias Columbus, Ignaz Rutter, and Dorothea Wagner. Search-space size in contraction hierarchies. In *ICALP 13*, pages 93–104, 2013.

5 R. Bellman. On a Routing Problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

6 Michael A. Bender and Martín Farach-Colton. The LCA problem revisited. In *LATIN 2000: Theoretical Informatics*. Springer Berlin Heidelberg, 2000.

7 M.W Bern, E.L Lawler, and A.L Wong. Linear-time computation of optimal subgraphs of decomposable graphs. *Journal of Algorithms*, 8(2):216–235, 1987.

8 H. L. Bodlaender. Dynamic programming on graphs with bounded treewidth. In *ICALP*, volume LNCS 317, pages 105–118. Springer, 1988.

9 H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6), December 1996.

10 H. L. Bodlaender. Discovering treewidth. In *SOFSEM'05*, volume LNCS 3381, pages 1–16. Springer, 2005.

11 Hans L. Bodlaender. A tourist guide through treewidth. *Acta Cybern.*, 11(1-2):1–21, 1993.

12 HansL. Bodlaender and Torben Hagerup. Parallel algorithms with optimal speedup for bounded treewidth. *SIAM Journal on Computing*, 27:1725–1746, 1995.

13 Krishnendu Chatterjee, Amir Kafshdar Goharshady, Rasmus Ibsen-Jensen, and Andreas Pavlogiannis. Algorithms for algebraic path properties in concurrent systems of constant treewidth components. In *POPL*, pages 733–747, 2016.

14 Krishnendu Chatterjee, Rasmus Ibsen-Jensen, Prateesh Goyal, and Andreas Pavlogiannis. Faster algorithms for algebraic path properties in recursive state machines with constant treewidth. In *POPL*, 2015.

15 Krishnendu Chatterjee, Rasmus Ibsen-Jensen, and Andreas Pavlogiannis. Faster algorithms for quantitative verification in constant treewidth graphs. In *CAV*, 2015.

16 Shiva Chaudhuri and Christos D. Zaroliagis. Shortest Paths in Digraphs of Small Treewidth. Part I: Sequential Algorithms. *Algorithmica*, 27:212–226, 1995.

17 Tobias Columbus. Search space size in contraction hierarchies. Master's thesis, Karlsruhe Institute of Technology, 2012.

18 T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction To Algorithms*. MIT Press, 2001.

19 Edsger. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

20 M. Elberfeld, A. Jakoby, and T. Tantau. Logspace versions of the theorems of Bodlaender and Courcelle. In *FOCS*, 2010.

21 Michael J. Fischer and Albert R. Meyer. Boolean Matrix Multiplication and Transitive Closure. In *SWAT (FOCS)*, pages 129–131. IEEE Computer Society, 1971.

22 Robert W. Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, 1962.

23 Lester R. Ford. Network Flow Theory. Report P-923, The Rand Corporation, 1956.

24 Rudolf Halin. S-functions for graphs. *Journal of Geometry*, 8(1-2):171–186, 1976.

25 D. Harel and R. Tarjan. Fast Algorithms for Finding Nearest Common Ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.

26 Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. on Systems Science and Cybernetics*, 4(2):100–107, 1968.

27 Donald B. Johnson. Efficient Algorithms for Shortest Paths in Sparse Networks. *J. ACM*, 24(1):1–13, January 1977.

28 Edward F. Moore. The shortest path through a maze. In *Proceedings of the International Symposium on the Theory of Switching, and Annals of the Computation Laboratory of Harvard University*, pages 285–292. Harvard University Press, 1959.

29 Leon R. Planken, Mathijs M. de Weerdt, and Roman P.J. van der Krogt. Computing all-pairs shortest paths by leveraging low treewidth. In *ICAPS-11*, pages 170–177. AAAI Press, 2011.

30 Neil Robertson and P.D Seymour. Graph minors. III. planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49–64, 1984.

31 B. Roy. Transitivité et connexité. *C. R. Acad. Sci. Paris*, 249:216–218, 1959.

32 Mikkel Thorup. All Structured Programs Have Small Tree Width and Good Register Allocation. *Information and Computation*, 142(2):159–181, 1998.

33 Stephen Warshall. A Theorem on Boolean Matrices. *J. ACM*, 9(1):11–12, January 1962.

34 Atsuko Yamaguchi, Kiyoko F. Aoki, and Hiroshi Mamitsuka. Graph complexity of chemical compounds in biological pathways. *Genome Informatics*, 14:376–377, 2003.

35 Yosuke Yano, Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. Fast and scalable reachability queries on graphs by pruned labeling with landmarks and paths. In *CIKM'13*, pages 1601–1606, 2013.

An ILP-based Proof System for the Crossing Number Problem*

Markus Chimani[1] and Tilo Wiedera[2]

1 Theoretical Computer Science, Osnabrück University, Germany
markus.chimani@uni-osnabrueck.de

2 Theoretical Computer Science, Osnabrück University, Germany
tilo.wiedera@uni-osnabrueck.de

Abstract

Formally, approaches based on mathematical programming are able to find provably optimal solutions. However, the demands on a verifiable formal proof are typically much higher than the guarantees we can sensibly attribute to implementations of mathematical programs. We consider this in the context of the *crossing number problem*, one of the most prominent problems in topological graph theory. The problem asks for the minimum number of edge crossings in any drawing of a given graph. Graph-theoretic proofs for this problem are known to be notoriously hard to obtain. At the same time, proofs even for very specific graphs are often of interest in crossing number research, as they can, e.g., form the basis for inductive proofs.

We propose a system to automatically generate a formal proof based on an ILP computation. Such a proof is (relatively) easily verifiable, and does not require the understanding of any complex ILP codes. As such, we hope our proof system may serve as a showcase for the necessary steps and central design goals of how to establish formal proof systems based on mathematical programming formulations.

1998 ACM Subject Classification G.1.6 [Optimization] Integer Programming, G.2.2 Graph Theory, G.4 [Mathematical Software] Verification

Keywords and phrases automatic formal proof, crossing number, integer linear programming

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.29

1 Introduction

A typical corner stone of (integral) mathematical programming formulations for combinatorial optimization problems is that solving the formulation constitutes a *proof* that the obtained solution is in fact optimal. While this is true in theory, it is not clear, per se, that this actually transfers into practice, as many factors may influence or invalidate the program's outcome: the probably most prominent ones are hidden bugs in the software or numerical instabilities. E.g. [1] discusses the problems and challenges of proving the correct computation of an optimal TSP tour for *one* specific instance; we are interested in a system to deduce proofs automatically, without any human interaction.

Hence, while there exist many successful formulations, e.g. as ILPs, to many important graph-theoretic (optimization) problems, successful computations are generally not considered to be proofs accepted by the graph theory community.

We aim at bridging this gap for the well-known *crossing number problem*, to be defined below, which is arguably one of the most prominent and notorious problems in topological

* This research has been supported by the German Research Foundation (DFG) project CH 897/2-1.

graph theory. We propose a system to extract a *simply verifiable* proof from a successful ILP computation, which can be accepted by graph theorists: it is shielded against ill-effects based on the software realization of the mathematical model. To understand the proof, the graph theorist does *not* have to have a deeper understanding of mathematical programming nor the required implementations; she only has to understand the mathematical model (and possibly a simple proof verification program, designed to be readable and checkable by non-experts).

When describing the proof system, we will also pinpoint the generally necessary differences in the formulation-, algorithm-, and software-design between the typical goal of obtaining a strong and fast solver and the goal of obtaining a system for easily understandable proofs. Crossing number formulations are in particular interesting in that respect, as their implementations need to combine a diverse set of different tools (branch-and-cut-and-prize with exact and heuristic constraint separation, column generation with non-standard bounding schemes, intricate heuristics for primal bounds, etc.) to obtain a system that can solve realistically-sized instances. This inevitably leads to a software too complex to directly check against all possible bugs. As such, even though we focus on the crossing number problem, our system may serve as a showcase of how to obtain trustworthy mathematical programming based proof systems for graph-theoretic problems whose formulations do not allow easily checkable implementations.

We will start with describing the crossing number problem, the reason why we are interested in formal proofs for the crossing number of specific graphs, and the currently available methods to obtain solutions. In Section 3 we will discuss the central design goals of our proof system, as well as their realization. This includes a new column generation scheme, balancing simplicity and effectiveness. A brief experimental study in Section 4 shows the applicability of our approach.

2 Crossing Number Problem

The crossing number $cr(G)$ of a graph G is the minimum number of pairwise edge crossings in any drawing of G in the plane. The problem garnered a lot of contributions since its first mention over 70 years ago; see [30, 29] for an extensive bibliography and survey. Beside its own inherent appeal, crossing numbers also occur, e.g., in conjectures relating $cr(G)$ to graph colorings and knot theory.

Nonetheless, some of the most natural questions are still open, most importantly the crossing number of nearly all classical graph classes like complete graphs (known only for K_n with $n \leq 12$ [28]), complete bipartite graphs, etc. The quest for crossing number proofs of particular families is a lively research field in graph theory, see, e.g., [3, 16, 21, 22, 26], and even partial results (like proving that K_{13} cannot have crossing number 217 [25]) are publishable in renowned journals. Several such proofs start out with a set of base cases for which the crossing number has to be proven in excruciating detail by hand, before employing an inductive proof to consider a full graph class. Obtaining sound, automatic, and standardized proofs for such base cases is one of the goals of our proof system.

Deciding the crossing number of an arbitrary graph G is known to be NP-complete [17]; this holds even for restricted graph classes like cubic graphs [18] or graphs that become planar when removing a single edge [6]. While there are several known practically strong heuristics and (partial) approximation results, we do, e.g., not even know if the problem allows a constant approximation ratio; we only know that the problem does not allow a PTAS [5]. The problem is known to be fixed-parameter tractable with parameter $cr(G)$ [20]. However, the algorithm's runtime is doubly-exponentially dependent on the parameter and it is, as already mentioned in [20], not feasible in practice.

The only known practical method to obtain the crossing number of a given graph is based in integer linear programs [4, 11, 12], based on two different modeling ideas to be described below. However, the models contain both too many variables and too many constraints to be solved directly via off-the-shelf techniques, and require a lot of (bug-prone) implementation effort. Even if implemented correctly, solving such ILPs can be error-prone due to numerical instabilities. This constitutes a problem for graph theorists, interested in utilizing the computed crossing number in a proof.

Basics

Kuratowski's famous theorem [24] states that a graph G is planar if and only if it does not contain a subdivision of a complete graph on five vertices (K_5) or a complete bipartite graph on three vertices per partition set ($K_{3,3}$) as a subgraph. We will call these subgraphs *Kuratowski subdivisions* of G. The paths in the subdivision that resemble a single edge of the K_5 or $K_{3,3}$ are referred to as *Kuratowski paths* in the following. We use this theorem to argue that every such subdivision in given G is required to be drawn with at least one crossing.

When considering the crossing number of a graph, it is well-known that it suffices to consider *good* drawings: no edge crosses itself; adjacent edges do not cross; each pair of edges crosses at most once; and no three edges cross in a common point. Considering such an optimal drawing of G, we can obtain a *planarization* of G, which is the graph arising from G when replacing each crossing with a new *dummy* vertex of degree 4. We may speak of a *partial planarization* if we substitute only certain crossings via new vertices such that the obtained graph possibly remains non-planar.

2.1 Known ILP Models

All known ILP models have a common core idea; they differ in how to handle the arising *realizability problem*, described below. We will only describe the formulation on a level necessary to comprehend the proof system (and the design decisions that lead to it). For a more detailed and formal description see the individual publications [4, 11, 12] or the full compilation in [7].

Let $G = (V, E)$ be the given simple and undirected graph for which to compute $cr(G)$, and let $\mathcal{CP} := \big\{\{e, f\} \subseteq E : e \cap f = \varnothing\big\}$ be the set of edge pairs that potentially cross in a good drawing of G. Consider a binary variable x_c for each $c \in \mathcal{CP}$ that is 1 if and only if the edge pair crosses. This gives the objective function

$$\min \sum_{\{e,f\} \in \mathcal{CP}} w(e) \cdot w(f) \cdot x_{\{e,f\}} \tag{1}$$

where $w(e)$ denotes the (integral) weight of edge e. For usual (i.e., unweighted) graphs we have $w(e) = 1$ for all $e \in E$. In our implementation we first preprocess G to obtain a smaller but integrally-weighted graph with the same crossing number [9].

To guarantee feasible solutions we may be tempted to introduce the following *Kuratowski constraints*. Let $K \subseteq E$ be an edge subset forming a Kuratowski subdivision; we want to ensure that each such K gives rise to at least one crossing. We say that a crossing $c \in \binom{K}{2} \cap \mathcal{CP}$ is *planarizing* if the graph obtained from K by realizing c via a dummy vertex is planar. Clearly, a crossing is planarizing if and only if the crossing edges do *not* belong to adjacent (or identical) Kuratowski paths. Let $\mathcal{CP}(K)$ be the set of planarizing crossings for

$K \subseteq E$.

$$\sum_{c \in \mathcal{CP}(K)} x_c \geq 1 \quad \forall \text{ Kuratowski subdivisions } K \text{ in } G.$$

While these constraints form facets of the crossing number polytope [8], they do not suffice to guarantee feasibility: On the one hand, we also have to consider Kuratowski subdivisions that only appear in partial planarizations due to dummy vertices. On the other hand, even those do not suffice: Let $R \subseteq \mathcal{CP}$ be edge pairs that are supposed to cross. The *realizability problem* is to decide whether there exists a drawing of G such that only the edge pairs R cross. Interestingly, even this seemingly simpler problem is still NP-hard for general graphs [23]. Hence, our simple set of x-variables cannot suffice to describe the crossing number polytope. The key problem is that when two edges f, g both cross an edge e, the *order* of these two crossings along e is of central importance and cannot be deduced in polynomial time (unless $\mathrm{P} = \mathrm{NP}$).

There are two approaches to tackle this problem. Both lead to a variable increase that, although polynomial, makes the models intractable in practice unless a dynamic column generation scheme is used. Assume in the following that we assign an arbitrary but fixed direction to each edge.

Subdivision-based exact crossing minimization (SECM)

Let $G^{[\ell]}$ be the graph obtained from G by splitting each edge into a chain of $\ell \in \mathbb{N}^+$ edges (henceforth called segments). Instead of directly using the above model on G, we consider $G^{[\ell]}$ instead. We observe that the corresponding set $\mathcal{CP}^{[\ell]}$ will not need to contain edge pairs (segment pairs, in fact) where both segments belong to the same original edge in G (the underlying G does not require self-crossings).

On $G^{[\ell]}$, we search for the smallest number of crossings under the restriction that each segment is involved in at most one crossing. This restriction is trivial to ensure via linear constraints (see later for details). The so-restricted crossing number is often called the *simple* crossing number, even though it is still NP-complete to decide. There can be at most $\chi := \min\{cr(G), |E| - 1\}$ crossings on an edge in the optimal drawing of G. Hence, we may use any upper bound on χ as ℓ to ensure that the optimal solution to the restricted crossing number problem on $G^{[\ell]}$ induces an optimal solution for the usual crossing number on G. Since there are instances where an edge e needs to be crossed by $\Omega(|E|)$ many other edges in the optimal solution, our transformation may increase the number of variables $\Theta(|E|^2)$-fold.

The benefit of considering the simple crossing number is that the realizability problem becomes linear time solvable. We say a subset $R \subset \mathcal{CP}^{[\ell]}$ is *simple* if each segment occurs at most once over all segment pairs in R (i.e., it is a potential solution to the simple crossing number). For such an R, its corresponding (partial) planarization $P(R)$ – obtained by substituting the crossings R in G with dummy vertices – is hence unique. We have R realizable iff $P(R)$ is planar.

Finally, we can ensure feasible solutions using more general Kuratowski-constraints. Let $\mathcal{K}(R)$ be the set of all Kuratowski subdivisions in $P(R)$. Each subdivision is specified by its edge set. Clearly, if the crossings R are part of the solution, each Kuratowski subdivision in $\mathcal{K}(R)$ will require at least one crossing. We have:

$$\sum_{c \in \mathcal{CP}(K)} x_c \geq 1 - \sum_{c \in R}(1 - x_c) \quad \forall \text{ simple } R \subseteq \mathcal{CP}, K \in \mathcal{K}(R) \tag{2}$$

In order to prove a lower bound of the crossing number, it suffices to understand that the constraints in the above model need to hold for any feasible solution. We do not need to argue about sufficiency (see also property ⟨2⟩ below).

Ordering-based exact crossing minimization (OECM)

The alternative formulation [12] introduces *linear ordering variables* to resolve the order of crossings along each edge without subdividing the input graph. This has some advantages regarding performance, e.g., since every Kuratowski constraint in this model can cover more than just one specific partial planarization. Technically, these linear orderings are modeled using $\Theta(|E|^3)$ additional variables that are linked to the crossing variables using several constraint classes. Overall, it has to be observed that the OECM model, while offering superior performance, is much harder to understand and requires even more technically intricate column generation schemes, book-keeping, and subalgorithms, compared to SECM.

3 Proof System

Without a proof system, one would need to check the ILP algorithms for correctness. All SECM and OECM implementations known to the authors are intertwined with the Open Graph Drawing Framework (OGDF, `www.ogdf.net`, [10]) and heavily utilize the ABACUS framework (`http://www.informatik.uni-koeln.de/abacus`, [19]); they are all written in C++. The core of both algorithms roughly spans across 8,000 Lines of Code (LOC) while the OGDF amounts to a total of about 170,000 LOC. Already the main code paths of the research code are hard to comprehend without intricate knowledge of the algorithms. Furthermore, the programs use sophisticated column generation routines, requiring complex book-keeping and special constraint liftings to prevent a decrease of the lower bound when adding variables. Tracking variables and constraints over an entire algorithm is disproportionately harder than simply verifying each branch-and-bound (B&B) leaf. Furthermore, there are several possibilities for hidden bugs due to numerical instabilities that may arise without any means of detection, or hidden buffer overruns when generating atypically many variables or constraints in one pass.

All these facts make a formal verification of the main algorithms intractable in practice. For comparison, our proof system proposes a verification procedure (written in Java) of less than 1,000 LOC (including rich documentation and comments), with a virtually complete test coverage.

In our context, a *proof* consists of three parts:

- a mathematical model (in our case the ILP formulation),
- a *witness* of the dual bound, and
- a primal *solution*, matching the above dual bound.

The first is independent of the specific instance but needs to be understood only once for a specific problem domain (crossing number, in our case). The latter two are instance-dependent. We want to make the proof as easily digestible by pure graph theorists as possible. To understand the proof it must be sufficient to understand the following:

⟨1⟩ *Solution.* One needs to be able to check the feasibility of a primal solution and evaluate its objective value. In our case, one needs to be able to recognize a feasible planarization of G, and to count the number of dummy vertices.

⟨2⟩ *Feasibility of the mathematical model.* It is only necessary to understand that all described constraints of the formulation are feasible; one need not concern oneself with understanding why the model is sufficient. Generally, it should be understood that any optimal fractional solution w.r.t. a subset of the constraints gives a feasible dual (in our case, lower) bound.

⟨3⟩ *Witness format.* The information contained in the witness that is required to verify the dual bound.

⟨4⟩ *Verification procedure.* The steps required to verify the dual bound claimed by the witness.

An ILP-based *proof system* should consist of two major components: the proof generation and the verification procedure. The arbitrarily complex proof generation produces a witness for the optimality of the primal solution. This witness contains all information required to create relaxed linear programs for several *subcases* (the leaves in the B&B tree), all of which yield the dual bound. B&B leaves naturally resemble an easily verifiable case distinction, as used ubiquitously in graph-theoretic proofs.

The verification of the witness could *theoretically* be done by hand. It follows from the nature of NP-complete problems that (unless $\mathrm{P} = \mathrm{NP}$) there *cannot* be a really "simple" proof for the dual bound in general. If we want a simple-to-check witness for the dual bound (which is, most importantly, checkable in polynomial time w.r.t. to its size), we *have* to live with the fact that the witness' size can grow exponentially with the size of G. In most cases, this sheer size will require us to introduce an – algorithmically very simple – computer-based verification procedure. Most importantly, the verification procedure only needs to check that the subcases described in the witness form linear programs that are subsets of the underlying mathematical model. It requires no knowledge about the generation of constraints, variables, or branches.

We can summarize the general design goals for an ILP-based proof system:

G1. *Simplicity of model.* There should be few classes of constraints and variables. Comprehensibility outweighs performance as long as the proof procedure is still "fast enough".

G2. *Column generation.* Column generation should only be used if ultimately required. The variable subsets need to be as simple as possible.

G3. *LP-solver flexibility/provability.* The LP-solver used during verification should be easily interchangeable or self-proving.

G4. *Few Branches.* Superfluous branching decisions should be eliminated from the witness to keep it small. This can, e.g., be achieved by starting the extraction with a supposedly optimal primal bound, see below.

G5. *Human-readability.* One should be able to investigate certain aspects of the proof by hand. To achieve this, we may allow redundancy as long as conflicts are detected easily by the verifier.

G6. *Standalone verification.* The verifier must *not* share any resources (most importantly code fragments) with the extraction procedure. The witness and the primal solution constitute the sole interchange of information between generation and verification.

G7. *Coding standards.* Adhering to well-established coding standards when implementing the verifier increases readability. Likewise, an established programming language should be used. The verification procedure should be described in detail such that one might re-implement it easily.

Although the OECM formulation offers better performance than SECM in practice, goal G1 lets us favor the comparably much easier to understand SECM formulation. It also allows us to sacrifice more constraints classes to adhere to G1. Concerning G2, both formulations require complex column generation schemes to be feasible in practice. However, as we will describe below, SECM allows us to propose a new column generation scheme

Algorithm 1: Proof generation

```
Require: graph $G$                        // we are interested in $cr(G)$
1: $P$, $UB \leftarrow$ OECM($G$)            // find presumably optimal planarization $P$
                                          //          (with objective value $UB$)
2: $W \leftarrow$ Modified-SECM($G$, $UB$)   // generate a witness $W$ for the lower bound,
                                          //          i.e., $UB - 1$ is infeasible to achieve
3: print $P$ and $W$                        // output the proof
```

that is considerably simpler than any of the previously published ones for either of the two formulations, while increasing the runtime and number of variables only mildly.

Alg. 1 gives an outline of the proof generation. In order to obtain a small proof (G4), we solve the crossing number problem *twice*: First, we use the fastest OECM variant together with strong upper bound heuristics to obtain the presumably optimal solution. This procedure can be seen as a black box, as we are only interested in the fact that the solution gives an upper bound – we can check the feasibility of the primal solution straight-forwardly. If our proof generation succeeds, this is the solution used as part of the overall proof. Now having this primal bound, we can start our modified SECM formulation (see below for details) without any primal heuristics and ask for a solution strictly better (at least one crossing less) than the obtained upper bound. From this second ILP run, we can extract all required information for each B&B leaf, to reconstruct each linear program that yields a lower bound on the number of crossings restricted to the solution space spanned by the branch. In general, the set of variables and constraints differ for any two leaves.

We observe that if OECM did *not* find an optimum solution (e.g., due to a hidden bug), we may already detect this now as SECM's dual bound does not match our upper bound.

3.1 Modified-SECM

There are two known column generation schemes for SECM [11]. The *algebraic pricing*, based on the standard Dantzig-Wolfe decomposition theory, performs relatively weak and uses a quite unstructured variable subset. The more efficient *combinatorial* column generation scheme (denoted as *sparse* column generation in the following) can decide upon the addition of variables in a purely combinatorial fashion, and requires the fewest active variables in general. However, the required variable subset structure (and hence the reasoning for its sufficiency) is too complicated to easily describe and comprehend for the purpose of a graph-theoretic proof. We hence propose a new column generation scheme – called *homogeneous* in the following – by means of describing an SECM variant that is slightly modified compared to the model described above.

Instead of a simple number, let $\ell\colon E \to \mathbb{N}$ be a mapping describing the *expansion status* of G, i.e., we consider each edge $e \in E$ of G to be subdivided into $\ell(e)$ segments. We define our ILP using the resulting graph G^ℓ. For notational simplicity, let $e_1, e_2, \dots, e_{\ell(e)}$ denote the segments of an original edge $e \in E$. As before, the new graph induces a set of segment pairs $\mathcal{CP}^\ell$ that may cross in an optimal solution. We explicitly allow (and expect) values $\ell(e)$ to be smaller than the upper bound of crossings over e. To this end, we allow at most one crossing over each segment *except for the first segment of each edge*: it may be crossed an arbitrary number of times. In general, this could lead to problems with testing realizability. However, this is of no concern to us, as we only require that our model is *feasible*, i.e., it allows to describe an optimal solution (see ⟨2⟩). This is trivially the case in this setting (already when $\ell(e) = 1$ for all $e \in E$).

Algorithm 2: Proof verification

```
Require: graph $G$, subcases $\mathcal{L}$ (=B&B leaves), claimed lower bound $b \in \mathbb{N}^+$
1: assert branchCoverage($\mathcal{L}$)
2: for all $\nu \in \mathcal{L}$ do
3:     $\ell \leftarrow$ expansion status at $\nu$
4:     $\mathcal{K} \leftarrow$ set of Kuratowski subdivisions observed at $\nu$
5:     for all $C \in \mathcal{K}$ do
6:         assert isKuratowski($G^\ell, C$)
7:     $P \leftarrow$ generateLinearProgram($G, \mathcal{K}$)
8:     assert lpsolve($P$) $> b - 1$
```

Symmetric solutions increase our set of subcases in the proof, often drastically. We can require w.l.o.g. that the crossings over an original edge may be *aligned* in such a way that there is only a crossing on segment $i > 2$ if there also is a crossing on segment $i - 1$. Segment 1 is typically not part of this alignment scheme, as it allows multiple crossings. However, observe that any edge $e = \{u, v\} \in E$ may be crossed at most $u_e := \min\{UB, |E|+1-\deg(u)-\deg(v)\}$ times in the optimum solution, where UB is an upper bound on $cr(G)$. If an edge is fully expanded, i.e., $\ell(e) = u_e$, we do allow at most one crossing over segment 1; to avoid symmetries we can assume to have less crossings on segment 1 than on segment $\ell(e)$. The following constraints establish these segment properties. They, together with the objective function (1) and the Kuratowski constraints (2) (both applied to the set $\mathcal{CP}^\ell$), form our full mathematical model.

$$\sum_{\{e_i,f\}\in\mathcal{CP}^\ell} x_{\{e_i,f\}} \le \begin{cases} 1 & \text{if } i = 2 \\ \sum_{\{e_{i-1},f\}\in\mathcal{CP}^\ell} x_{\{e_{i-1},f\}} & \text{else} \end{cases} \qquad \forall e \in E, 2 \le i \le \ell(e) \tag{3}$$

$$\sum_{\{e_1,f\}\in\mathcal{CP}^\ell} x_{\{e_1,f\}} \le \sum_{\{e_{\ell(e)},f\}\in\mathcal{CP}^\ell} x_{\{e_{\ell(e)},f\}} \qquad \forall e \in E : \ell(e) = u_e \tag{4}$$

Remark (Irrelevant to understanding the proof)

As in SECM, Kuratowski constraints are separated via a planarity-test based procedure on a rounded solution S. An effective column generation similar to [4] is achieved by starting with a unit vector ℓ and incrementing $\ell(e)$ whenever there are at least 2 crossings on e_1 in S (i.e., the realization problem cannot be solved uniquely).

3.2 Verification Procedure

Finally, we can focus on the actual verification steps necessary to prove the lower bound obtained by the above Modified-SECM model. Alg. 2 gives an overview.

Branch Coverage

We need to make sure that the entire solution space is covered. Therefore, we consider the variable fixings in all subcases. Since we only branch on single variables, we iteratively merge two subcases that differ by the assignment of a single variable, giving a more general subcase without this variable being fixed at all. In a valid proof, this procedure must end with a single subcase, which does not have any variable fixings at all.

Algorithm 3: Coverage verification

```
Require: subcases $\mathcal{L}$ (see text)
1: while $\exists \mu, \nu \in \mathcal{L}$ with $\exists c \in \mathcal{CP}^\ell : \mu \triangle \nu = \{(c,0),(c,1)\}$ do
2:     $L \leftarrow \{\mu \cap \nu\} \cup L \setminus \{\mu, \nu\}$
3: assert $\mathcal{L} = \{\varnothing\}$
```

The following pseudo-code shows how to algorithmically verify that the subcases span the whole solution space. Here, we consider each subcase ν to be a set of tuples from $\mathcal{CP}^\ell \times \{0,1\}$, i.e., a set of segment pairs that are specified to either cross (1) or not (0). Segment pairs not listed in ν are free to do either. Let $\triangle$ denote the symmetric difference.

Kuratowski Constraints

For each subcase, we need to check that only feasible constraints are considered. A Kuratowski constraint for a subgraph K that is not a Kuratowski subdivision would be an error. It would enforce a crossing that may not be necessary in the optimal solution. For each subcase, our witness explicitly stores each used Kuratowski subgraph K, together with the required crossings (R) that need to exist for K to arise[1]. More specifically, K is stored by means of Kuratowski paths $p_1^K, \dots, p_k^K$. This storage pattern allows for a simpler verification (see below) than a general Kuratowski verification routine as described in [27].

To verify that K is really a Kuratowski subdivision, we first check whether each p_i^K is in fact a valid path (only exploiting crossings of R, if any). Then we check that all paths are pairwise internally-disjoint (i.e., they are disjoint except for possibly common start/end vertices). Finally, the set of nodes that constitute the start or end of all Kuratowski paths is collected. The size of this set (5, 6) and the number of Kuratowski paths (10, 9) is verified according to the type of the subdivision (K_5, $K_{3,3}$, respectively). The structural verification of a K_5 subdivision simply checks whether all 5 nodes are directly connected to one another via Kuratowski paths. For a $K_{3,3}$, we perform a two-coloring (interpreting the paths as edges); each of the 6 nodes must be connected to exactly 3 distinct nodes of the opposite color.

Lower Bound

Finally, we need to verify the lower bound for each subcase. We can trivially generate a linear program (no integrality constraints) according to our model. From the expansion status ℓ we can construct $\mathcal{CP}^\ell$, the objective function (1) and the segment-oriented constraints (3) and (4). For each (already verified) Kuratowski subdivision considered in the subcase, we generate the corresponding constraint (2).

By writing this LP in a standard file, we can use *any* LP-solver (or multiple, to gain confidence) to verify that the LP's solution value is strictly larger than $UB - 1$. For a more formal proof, we may check the basis of the final tableau/the dual solution to verify the lower bound and/or use self-proving LP solvers [2, 13].

[1] While constraints *could* be stored without explicitly stating R, this would decrease the readability of the proof and the verification procedure, cf. G5.

4 Practice and Experiments

4.1 Web-Service

Already prior to our proof system, we offered a (free) web-service to compute the crossing number of an uploaded graph (see `http://crossings.uos.de`). Over the last years it has been used as a tool by several research groups worldwide, to help validate or falsify crossing number conjectures and ideas. We collected the thereby uploaded instances. However, the web-service would (formally) only give a primal solution, together with the assertion that this *should* be the optimum. Now, we relaunched the web-service to also hand out the formal proof. The user can download the stand-alone Java verification program, check it, and use it to verify her proof independently.

4.2 Experimental Evaluation

To determine the applicability of the proof system, we tested the algorithms on three benchmark sets: the 3110 non-trivial Rome graphs [14], the 1277 North graphs [15], and the 145 non-planar graphs (`http://crossings.uos.de/instances`) collected by our crossing number web-service.

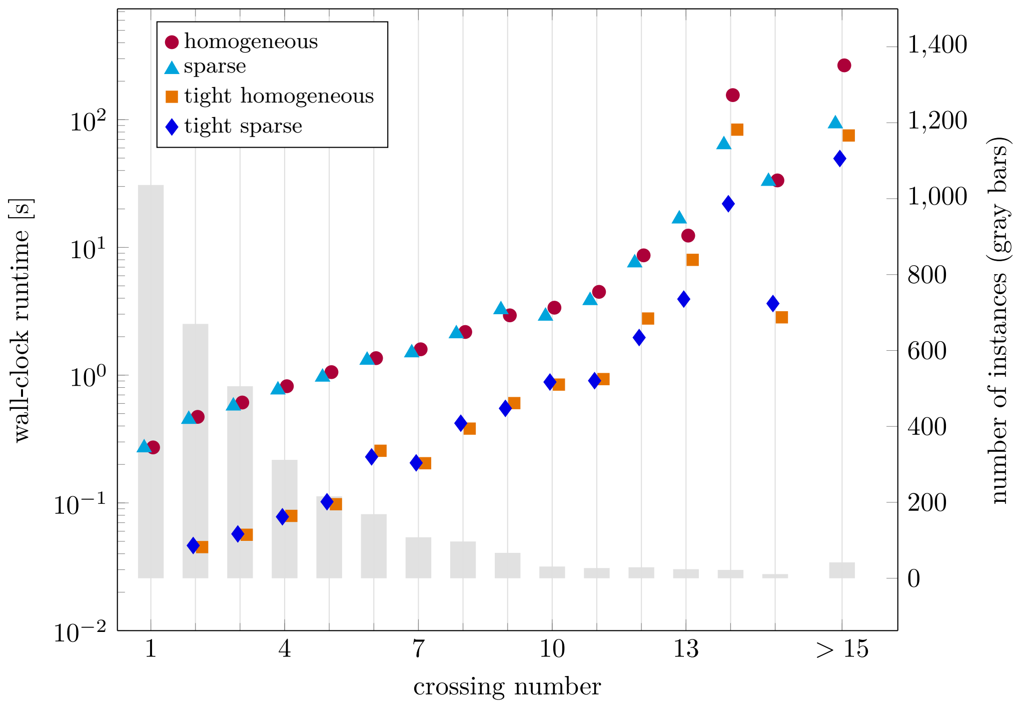
All experiments were conducted using an Intel Xeon E5-2430 v2, 2.50 GHz with 192 GB RAM running on Debian 8. We compiled with g++ 4.9.0 (64bit, -O3), used CPLEX 12.6.0 as the backend LP-solver, and applied a time-limit of 60 minutes for each computation. All algorithms except the verifier are implemented as part of the OGDF (using ABACUS as the ILP-framework). We compare the *sparse* to the newly introduced *homogeneous* column generation scheme, to understand the runtime costs of the simpler but supposedly weaker column generation scheme. For both schemes we consider the cases whether we start with a *tight* upper bound (the optimum) or not; the former is the setting we typically use within our proof system. Fig. 1 summarizes the results. While *tight homogeneous* requires more time than *tight sparse* on larger instances, it is still faster than *sparse* without a tight upper bound. On all instances with crossing number at most 22, *homogeneous* is at most 5 times slower than *sparse* (for both upper bound modes, considering only those instances solved by both schemes). Using the tight upper bound reduces the runtime to about 30% on average for *homogeneous*. Out of all 3393 instances solved by *tight sparse*, only 11 could not also be solved by *tight homogeneous* in time. Thus, we conclude that the increase in running time due to the simpler column generation is reasonable in practice. The runtime of the verification procedure is negligibly small.

5 Conclusion

We considered the problem of bridging the gap between "provably optimal" solutions obtained via mathematical programming and the demands on a verifiable formal proof. To this end, we laid out the general central design goals and steps to turn a mathematical program into a proof system.

We combined this with a showcase of how to automatically obtain a verifiable proof for the graph-theoretic crossing number problem, whose known ILP implementations are far from being formally checkable. To this end, we also introduced a novel column generation scheme for the problem's model, which, while much simpler, is still very effective in practice. The final proof system is available online for free academic use.

Figure 1 Running time of different SECM variants (see text). The tight variants finished in roughly 10^{-5} seconds for instances with crossing number 1.



Acknowledgements. We thank an anonymous reviewer for her valuable feedback including detailed information on related publications.

References

1 David Applegate, Robert E. Bixby, Vasek Chvátal, William J. Cook, Daniel G. Espinoza, Marcos Goycoolea, and Keld Helsgaun. Certification of an optimal TSP tour through 85, 900 cities. *Oper. Res. Lett.*, 37(1):11–15, 2009. `doi:10.1016/j.orl.2008.09.006`.

2 David Applegate, William J. Cook, Sanjeeb Dash, and Daniel G. Espinoza. Exact solutions to linear programming problems. *Oper. Res. Lett.*, 35(6):693–699, 2007. `doi:10.1016/j.orl.2006.12.010`.

3 Drago Bokal. On the crossing numbers of cartesian products with paths. *Journal of Combinatorial Theory, Series B*, 97(3):381–384, 2007. `doi:10.1016/j.jctb.2006.06.003`.

4 Christoph Buchheim, Markus Chimani, Dietmar Ebner, Carsten Gutwenger, Michael Jünger, Gunnar W. Klau, Petra Mutzel, and René Weiskircher. A branch-and-cut approach to the crossing number problem. *Discrete Optimization*, 5(2):373–388, 2008. `doi:10.1016/j.disopt.2007.05.006`.

5 Sergio Cabello. Hardness of approximation for crossing number. *Discrete & Computational Geometry*, 49(2):348–358, 2013. `doi:10.1007/s00454-012-9440-6`.

6 Sergio Cabello and Bojan Mohar. Adding one edge to planar graphs makes crossing number and 1-planarity hard. *SIAM Journal on Computing*, 42:1803–1829, 2013.

7 Markus Chimani. *Computing Crossing Numbers*. PhD thesis, TU Dortmund, 2008. URL: `http://hdl.handle.net/2003/25955`.

8 Markus Chimani. Facets in the crossing number polytope. *SIAM Journal on Discrete Mathematics*, 25(1):95–111, 2011. URL: http://epubs.siam.org/sidma/resource/1/sjdmec/v25/i1/p95_s1, doi:10.1137/09076965X.

9 Markus Chimani and Carsten Gutwenger. Non-planar core reduction of graphs. *Discrete Mathematics*, 309(7):1838–1855, 2009. doi:10.1016/j.disc.2007.12.078.

10 Markus Chimani, Carsten Gutwenger, Michael Jünger, Gunnar W. Klau, Karsten Klein, and Petra Mutzel. The open graph drawing framework (OGDF). In Roberto Tamassia, editor, *Handbook on Graph Drawing and Visualization.*, pages 543–569. Chapman & Hall/CRC, 2013.

11 Markus Chimani, Carsten Gutwenger, and Petra Mutzel. Experiments on exact crossing minimization using column generation. *ACM J. Experim. Alg.*, 14:4:3.4–4:3.18, 2010. doi:10.1145/1498698.1564504.

12 Markus Chimani, Petra Mutzel, and Immanuel Bomze. A new approach to exact crossing minimization. In *Proc. ESA*, volume 5193 of *LNCS*, pages 284–296, 2008. doi:10.1007/978-3-540-87744-8_24.

13 Marcel Dhiflaoui, Stefan Funke, Carsten Kwappik, Kurt Mehlhorn, Michael Seel, Elmar Schömer, Ralph Schulte, and Dennis Weber. Certifying and repairing solutions to large LPs how good are LP-solvers? In *Proc. Fourteenth SODA*, pages 255–256. ACM/SIAM, 2003.

14 Giuseppe Di Battista, Ashim Garg, Giuseppe Liotta, Roberto Tamassia, Emanuele Tassinari, and Francesco Vargiu. An experimental comparison of four graph drawing algorithms. *Computational Geometry*, 7(5–6):303–325, 1997. 11th ACM Symposium on Computational Geometry. doi:10.1016/S0925-7721(96)00005-3.

15 Guiseppe Di Battista, Ashim Garg, Guiseppe Liotta, Armando Parise, Roberto Tassinari, Emanuele Tassinari, Francesco Vargiu, and Luca Vismara. Drawing directed acyclic graphs: An experimental study. *International Journal of Computational Geometry & Applications*, 10(06):623–648, 2000. doi:10.1142/S0218195900000358.

16 Geoffrey Exoo, Frank Harary, and Jerald Kabell. The crossing numbers of some generalized petersen graphs. *Mathematica Scandinavica*, 48(1):184–188, 1981.

17 M. R. Garey and D. S. Johnson. Crossing number is NP-complete. *SIAM Journal on Algebraic Discrete Methods*, 4(3):312–316, 1983. doi:10.1137/0604033.

18 Petr Hliněný. Crossing number is hard for cubic graphs. *Journal of Combinatorial Theory. Series B*, 96(4):455–471, 2006. doi:10.1016/j.jctb.2005.09.009.

19 Michael Jünger and Stefan Thienel. The ABACUS system for branch-and-cut-and-price algorithms in integer programming and combinatorial optimization. *Softw., Pract. Exper.*, 30(11):1325–1352, 2000. doi:10.1002/1097-024X(200009)30:11<1325::AID-SPE342>3.0.CO;2-T.

20 Ken-ichi Kawarabayashi and Buce Reed. Computing crossing number in linear time. In *Proc. STOC*, pages 382–390, 2007. doi:10.1145/1250790.1250848.

21 Marián Klešč. The crossing numbers of join of the special graph on six vertices with path and cycle. *Discrete Math.*, 310(9):1475–1481, 2010. doi:10.1016/j.disc.2009.08.018.

22 Marián Klešč, R. Bruce Richter, and Ian Stobert. The crossing number of $C_5 \times C_n$. *Journal of Graph Theory*, 22(3):239–243, 1996. doi:10.1002/(SICI)1097-0118(199607)22:3<239::AID-JGT4>3.0.CO;2-N.

23 Jan Kratochvíl. String graphs. II. recognizing string graphs is NP-hard. *Journal of Combinatorial Theory, Series B*, 52(1):67–78, 1991. doi:10.1016/0095-8956(91)90091-W.

24 Casimir Kuratowski. Sur le problème des courbes gauches en topologie. *Fundamenta Mathematicae*, 15(1):271–283, 1930. URL: http://eudml.org/doc/212352.

25 Dan McQuillan, Shengjun Pan, and R. Bruce Richter. On the crossing number of K_{13}. *Journal of Combinatorial Theory, Series B*, 115:224–235, 2015. `doi:10.1016/j.jctb.2015.06.002`.

26 Dan McQuillan and R. Bruce Richter. On the crossing numbers of certain generalized petersen graphs. *Discrete Mathematics*, 104(3):311–320, 1992. `doi:10.1016/0012-365X(92)90453-M`.

27 Lars Noschinski, Christine Rizkallah, and Kurt Mehlhorn. Verification of certifying computations through autocorres and simpl. In *Proc. NASA Formal Methods - 6th International Symposium*, volume 8430 of *LNCS*, pages 46–61, 2014. `doi:10.1007/978-3-319-06200-6_4`.

28 Shengjun Pan and R. Bruce Richter. The crossing number of K_{11} is 100. *Journal of Graph Theory*, 56(2):128–134, 2007. `doi:10.1002/jgt.v56:2`.

29 Marcus Schaefer. The graph crossing number and its variants: a survey. *Electronic Journal of Combinatorics*, 2013.

30 Imrich Vrt'o. Crossing numbers of graphs: A bibliography. `ftp://ftp.ifi.savba.sk/pub/imrich/crobib.pdf`, 2014.

Strategic Contention Resolution with Limited Feedback

George Christodoulou[*1], **Martin Gairing**[†2], **Sotiris Nikoletseas**[‡3], **Christoforos Raptopoulos**[§4], **and Paul Spirakis**[¶5]

1 **Department of Computer Science, University of Liverpool, Liverpool, UK**
G.Christodoulou@liverpool.ac.uk

2 **Department of Computer Science, University of Liverpool, Liverpool, UK**
gairing@liverpool.ac.uk

3 **Computer Engineering and Informatics Department, University of Patras, Patras, Greece; and**
Computer Technology Institute & Press "Diophantus", Greece
nikole@cti.gr

4 **Computer Engineering and Informatics Department, University of Patras, Patras, Greece; and**
Computer Technology Institute & Press "Diophantus", Greece
raptopox@ceid.upatras.gr

5 **Department of Computer Science, University of Liverpool, Liverpool, UK; and**
Computer Engineering and Informatics Department, University of Patras, Patras, Greece; and
Computer Technology Institute & Press "Diophantus", Greece
P.Spirakis@liverpool.ac.uk

Abstract

In this paper, we study contention resolution protocols from a game-theoretic perspective. We focus on *acknowledgment-based* protocols, where a user gets feedback from the channel only when she attempts transmission. In this case she will learn whether her transmission was successful or not. Users that do not transmit will not receive any feedback. We are interested in equilibrium protocols, where no player has an incentive to deviate.

The limited feedback makes the design of equilibrium protocols a hard task as best response policies usually have to be modeled as Partially Observable Markov Decision Processes, which are hard to analyze. Nevertheless, we show how to circumvent this for the case of two players and present an equilibrium protocol. For many players, we give impossibility results for a large class of acknowledgment-based protocols, namely *age-based* and *backoff* protocols with finite expected finishing time. Finally, we provide an age-based equilibrium protocol, which has infinite expected finishing time, but every player finishes in linear time with high probability.

1998 ACM Subject Classification C.2.2 Network Protocols

Keywords and phrases contention resolution, acknowledgment-based protocols, game theory

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.30

[*] George Christodoulou was supported by EP/M008118/1, EP/K01000X/1 and LT140046 and part of this work was done while the first author was visiting the Simons Institute for the Theory of Computing.
[†] Martin Gairing was supported by EPSRC grant EP/L011018/1.
[‡] Sotiris Nikoletseas was partially supported by the MULTIPLEX project – 317532.
[§] Christoforos Raptopoulos was partially supported by the MULTIPLEX project – 317532.
[¶] Paul Spirakis was partially supported by the MULTIPLEX project – 317532 and the EEE/CS School initiative NeST of the University of Liverpool.

1 Introduction

Contention resolution in multiple access channels is one of the most fundamental problems in networking. In a multiple access channel (or broadcast channel) multiple users want to communicate with each other by sending messages into the channel. The channel is not centrally controlled, so two or more users can transmit their messages at the same time. If this happens then the messages collide and the transmission is unsuccessful. Contention resolution protocols specify how to resolve such conflicts, while simultaneously optimizing some performance measure, like channel utilization or average throughput.

In this paper we follow the standard assumption that time is divided into discrete time slots, messages are broken up into fixed sized packets, and one packet fits exactly into one time slot. Moreover, we consider one of the simplest possible scenarios where there are n users, each of them having a single packet that needs to be transmitted through the channel. When exactly one user attempts transmission in a given slot, the transmission is successful. However, if more than one users attempt transmission in the same slot, a collision occurs, their transmission fails and they need to retransmit their packages in later time slots.

Under centralized control of the users, avoiding collisions would be simple: exactly one user would transmit at each time step, alternating in a round-robin fashion. The complexity of the problem stems from the fact that there is no centralized control and therefore channel access has to be managed by a distributed protocol. There is a large body of literature that studies efficient distributed contention resolution protocols (see Section 1.2). However, these protocols work under the assumption that users will obediently follow the algorithm. In this paper we follow [9] by dropping this assumption. We model the situation as a non-cooperative stochastic game, where each user acts as a selfish *player* and tries to minimize the expected time before she transmits successfully. Therefore a player will only obey a protocol if it is in her best interest, given the other players stick to the protocol.

Fiat, Mansour, and Nadav [9] designed an incentive-compatible transmission protocol which guarantees that (with high probability) all players will transmit successfully in time linear in n. Their protocol works for a very simple channel feedback structure, where each player receives feedback of the form $0/1/2^+$ after each time step (*ternary* feedback), indicating whether zero, one, or more than one transmission was attempted. Christodoulou, Ligett and Pyrga [8] designed equilibrium protocols for *multiplicity* feedback, where each player receives as feedback the number of players that attempted transmission[1].

The above protocols fall in the class of *full-sensing* protocols [13] where the channel feedback is broadcasted to all sources. However, in wireless channels, there are situations where full-sensing is not possible because of the *hidden-terminal problem* [27]. In this paper, we focus on *acknowledgment-based* protocols, which use a more limited feedback model – the only feedback that a user gets is whether her transmission was successful or not. A user that does not transmit cannot "listen" to channel and therefore does not get any feedback. In other words, the only information that a user has is the history of her own transmission attempts. Acknowledgment-based protocols have been extensively studied in the literature (see e.g. [13] and references therein). *Age-based* and *backoff* protocols both belong to the class of acknowledgment-based protocols.

Age-based protocols can be described by a sequence of probabilities (one for each time-step) of transmitting in each time step. Those probabilities are given beforehand and do not change based on the transmission history. The well known ALOHA protocol [1] is a special

[1] They also assume non-zero transmission costs, as opposed to [9] and to this work.

age-based protocol, where – except for the first round – users always transmit with the same probability. In contrast, in *backoff* protocols, the probability of transmitting in the next time step only depends on the number of unsuccessful transmissions for the user. Here, a popular representative is the *binary exponential backoff* mechanism, which is also used by the Ethernet protocol [20].

The design and the limitations of acknowledgment-based protocols is well-understood [10, 18] if the users are not strategic. In this paper, we focus on the game-theoretic aspect of those protocols.

1.1 Our Results

We study the design of acknowledgment-based *equilibrium protocols*. A user gets feedback only when she attempts transmission, in which case she either receives an acknowledgment, in case of success, or she realizes that a collision occurred (by the lack of an acknowledgment). This model allows for very limited feedback, as opposed to *full-sensing* protocols studied in [9, 8] where all players, even those who did not attempt transmission receive channel feedback.

The feedback models used in [9, 8] allow players, at each given time, to know exactly the number of pending players. This information is very useful for the design of equilibrium protocols. In our case, we assume that the number of pending players is common knowledge only at the beginning. If a player chooses not to transmit during a time-slot, then she is not sure how many players are still in the game. From this time on, she can only sense the existence of other pending players when she participates in a collision.

The analysis of acknowledgment-based equilibrium protocols requires different techniques. In full-sensing protocols, a best response for a source can be modeled as an optimal policy of a Markov Decision Process (MDP) [9]. For an acknowledgment-based protocol, this is in general no longer possible, due to the uncertainty imposed by a non-transmission. However, the best response policy in this case can be modeled as a *Partially Observable Markov Decision Processes* (POMDP), which are more complicated to analyze.

Lack of information makes the design of equilibrium protocols a hard task. In particular, we show in Section 4 that it is impossible to design an age-based or backoff protocol that is in equilibrium and has finite expected finishing time [2]. These impossibility results contribute to a partial characterisation of such protocols and even hold for the case of two players. This stands in contrast to the full-sensing case for which the authors in [9] give an equilibrium protocol, where the k remaining players transmit with probability $\Theta\left(\frac{1}{\sqrt{k}}\right)$. This protocol finishes in finite but exponential time.

In Section 3, we introduce and analyze an equilibrium protocol for two players. An interesting feature of our protocol is that each player is using only limited information of her own history. More precisely, the probability of transmission in a time-slot, depends only on whether a player attempted transmission in the previous slot. Our proof reduces the POMDP for the best response policy to a *finite* MDP, which we then analyze. This reduction crucially relies on the nature of our protocol. We further show that our equilibrium protocol is the unique stationary equilibrium protocol.

For more than two players, we present an age-based equilibrium protocol. Although it has infinite expected finishing time, every player finishes in linear time with high probability. Our

[2] Note, that for more than two players, always transmitting is an equilibrium protocol with *infinite* expected finishing time [9].

protocol circumvents the lack of information by maintaining an estimation on the number of pending players, which with high probability is an upper bound on the actual number. The protocol uses a deadline mechanism similar to [9]. Their protocol exploits the existence of their finite time equilibrium protocol mentioned above. For our more restricted model it is not known if such a finite time protocol exists for more than two players. This is the main open question left from our work. We stress that our negative results exclude the possibility that such a protocol can be age-based or backoff.

1.2 Related Work

The ALOHA protocol, introduced by Abramson [1] (and modified by Roberts [25] to its slotted version), is a multiple-access communication protocol, which has been around since the 70's. Many subsequent papers study the efficiency of multiple-access protocols when packets are generated by some stochastic process (see for example [12, 11, 24]), while worst-case scenarios of bursty inputs, were studied in [5]. To model such a worst-case scenario, one needs n nodes, each of which must simultaneously transmit a packet; this is also the model we use in this work.

A large class of contention resolution protocols explicitly deals with *conflict resolution*; where if $k \geq 2$ users collide (out of a total of n users), then a resolution algorithm is called on to resolve this conflict (by ensuring that all the colliding packets are successfully transmitted), before any other source is allowed to use the channel [7, 6, 15, 28]. There have been many positive and negative results on the efficiency of protocols under various information models (see [13] for an overview of results). When k is known, [10] provides an $O(k + \log k \log n)$ *acknowledgment-based* algorithm, while [18] provides a matching lower bound. For the ternary model, [14] provides a bound of $\Omega(k(\log n / \log k))$ for all deterministic algorithms.

A variety of game theoretic models of slotted ALOHA have also been proposed and studied; see for example [2, 17, 3]. However, much of this work only considers transmission protocols that always transmit with the same fixed probability (perhaps as a function of the number of players in the game). Other game theoretic approaches have considered pricing schemes [29] and cases in which the channel quality changes with time and players must choose their transmission levels accordingly [19, 30, 4]. [16] studied a game-thoretic model that lies between the contention and congestion model, where the decision of *when* to submit is part of the action space of the players. As discussed in the previous section, the most relevant game-theoretic model to our work, is the one studied by Fiat, Mansour, and Nadav [9] and by Christodoulou, Ligett, and Pyrga [8]. In [8], efficient ϵ-equilibrium protocols are designed, but the authors assume non-zero transmission costs, in which case the efficient protocol of [9] does not apply. Their protocols use *multiplicity* feedback (the number of attempted transmissions) which again falls in the class of full-sensing protocols.

2 Model

Game Structure. Let $N = \{1, 2, \ldots, n\}$ be the set of agents, each one of which has a single packet that he wants to send through a common channel. All players know n. We assume time is discretized into slots $t = 1, 2, \ldots$. The players that have not yet successfully transmitted their packet are called *pending* and initially all n players are pending. At any given time slot t, a pending player i has two available actions, either to *transmit* his packet or to *remain quiet*. In a *(mixed) strategy*, a player i transmits his packet at time t with some probability that potentially depends on information that i has gained from the channel based on previous transmission attempts. If exactly one player transmits in a given slot t, then his transmission is *successful*, the successful player exits the game (i.e. he is no longer pending), and the game continues with the rest of the players. On the other hand, whenever two or more agents try to access the channel (i.e. transmit) at the same slot, a *collision* occurs and their transmissions fail, in which case the agents remain in the game. Therefore, in case of collision or if the channel is idle (i.e. no player attempts to transmit) the set of pending agents remains unchanged. The game continues until all players have successfully transmitted their packets.

Transmission protocols. Let $X_{i,t}$ be the indicator variable that indicates whether player i attempted transmission at time t. For any $t \geq 1$, we denote by $\vec{X}_t$ the transmission vector at time t, i.e. $\vec{X}_t = (X_{1,t}, X_{2,t}, \dots, X_{n,t})$. An *acknowlegment-based* protocol, uses very limited channel feedback. After each time step t, only players that attempted a transmission receive feedback, and the rest get no information. In fact, the information received by a player i who transmitted during t is whether his transmission was successful (in which case he gets an acknowledgement and exits the game) or whether there was a collision.

Let $\vec{h}_{i,t}$ be the vector of the *personal transmission history* of player i up to time t, i.e. $\vec{h}_{i,t} = (X_{i,1}, X_{i,2}, \dots, X_{i,t})$. We also denote by $\vec{h}_t$ the transmission history of all players up to time t, i.e. $\vec{h}_t = (\vec{h}_{1,t}, \vec{h}_{2,t}, \dots, \vec{h}_{n,t})$. In an acknowledgement-based protocol, the actions of player i at time t depend only (a) on his personal history $\vec{h}_{i,t-1}$ and (b) on whether he is pending or not at t. A *decision rule* $f_{i,t}$ for a pending player i at time t, is a function that maps $\vec{h}_{i,t-1}$ to a probability $\Pr(X_{i,t} = 1|\vec{h}_{i,t-1})$. For a player $i \in N$, a *(transmission) protocol* f_i is a sequence of decision rules $f_i = \{f_{i,t}\}_{t\geq 1} = f_{i,1}, f_{i,2}, \cdots$.

A transmission protocol is *anonymous* if and only if the decision rule assigns the same transmission probability to all players with the same personal history. In particular, for any two players $i \neq j$ and any $t \geq 0$, if $\vec{h}_{i,t-1} = \vec{h}_{j,t-1}$, it holds that $f_{i,t}(\vec{h}_{i,t-1}) = f_{j,t}(\vec{h}_{j,t-1})$. In this case, we drop the subscript i in the notation, i.e. we write $f = f_1 = \cdots = f_n$.

We call a protocol f_i for player i *age-based* if and only if, for any $t \geq 1$, the transmission probability $\Pr(X_{i,t} = 1|\vec{h}_{i,t-1})$ depends only (a) on time t and (b) on whether player i is pending or not at t. In this case, we will denote the transmission probability by $p_{i,t} \overset{def}{=} \Pr(X_{i,t} = 1|\vec{h}_{i,t-1}) = f_{i,t}(\vec{h}_{i,t-1})$.

A protocol is called *backoff* if the decision rule at time t is a function of the number of *unsuccessful* transmissions. We call a transmission protocol f_i *non-blocking* if and only if, for any $t \geq 1$ and any transition history $\vec{h}_{i,t-1}$, the transmission probability $\Pr(X_{i,t} = 1|\vec{h}_{i,t-1})$ is always smaller than 1. A protocol f_i for player i is a *deadline protocol with deadline* $t_0 \in \{1, 2, \dots\}$ if and only if $f_{i,t}(\vec{h}_{i,t-1}) = 1$, for any player i, any time slot $t \geq t_0$ and any transmission history $\vec{h}_{i,t-1}$. A *persistent player* is one that uses the deadline protocol with deadline 1.

Efficiency. Assume that all n players in the game employ an anonymous protocol f. We will say that f is *efficient* if and only if all players will have successfully transmitted by time $\Theta(n)$ with high probability (i.e. with probability tending to 1, as n goes to infinity).

Individual utility. Let $\vec{f} = (f_1, f_2, \dots, f_n)$ be such that player i uses protocol $f_i, i \in N$. For a given transmission sequence $\vec{X}_1, \vec{X}_2, \dots$, which is consistent with $\vec{f}$, define the *latency* or *success time* of agent i as $T_i \overset{def}{=} \inf\{t : X_{i,t} = 1, X_{j,t} = 0,\ \forall j \neq i\}$. That is, T_i is the time at which i successfully transmits. Given a transmission history $\vec{h}_t$, the n-tuple of protocols $\vec{f}$ induces a probability distribution over sequences of further transmissions. In that case, we write $C_i^{\vec{f}}(\vec{h}_t) \overset{def}{=} \mathbb{E}[T_i|\vec{h}_t, \vec{f}] = \mathbb{E}[T_i|\vec{h}_{i,t}, \vec{f}]$ for the expected latency of agent i incurred by a sequence of transmissions that starts with $\vec{h}_t$ and then continues based on $\vec{f}$. For anonymous protocols, i.e. when $f_1 = f_2 = \cdots = f_n = f$, we will simply write $C_i^f(\vec{h}_t)$ instead[3].

Equilibria. The objective of every agent is to minimize her expected latency. We say that $\vec{f} = \{f_1, f_2, \ldots, f_n\}$ is in *equilibrium* if for any transmission history $\vec{h}_t$ the agents cannot decrease their expected latency by unilaterally deviating after t; that is, for all agents i, for all time slots t, and for all decision rules f'_i for agent i, we have

$$C_i^{\vec{f}}(\vec{h}_t) \leq C_i^{(\vec{f}_{-i}, f'_i)}(\vec{h}_t),$$

where $(\vec{f}_{-i}, f'_i)$ denotes the protocol profile[4] where every agent $j \neq i$ uses protocol f_j and agent i uses protocol f'_i.

3 An equilibrium protocol for two players

In this section we show that there is an anonymous acknowledgment-based protocol in equilibrium, when $n = 2$.

We define the protocol f as follows: for any $t \geq 1$, player i and transmission history $\vec{h}_{i,t-1}$,

$$f_{i,t}(\vec{h}_{i,t-1}) = \begin{cases} \frac{2}{3}, & \text{if } X_{i,t-1} = 1 \text{ or } t = 1 \\ 1, & \text{if } X_{i,t-1} = 0. \end{cases} \tag{1}$$

▶ **Theorem 1.** *There is an anonymous acknowledgment-based equilibrium protocol for two players.*

Proof. We will show that protocol f is in equilibrium. Let Alice and Bob be the two players in the system. We will show that when Bob sticks with playing f, any deviation for Alice, at any possible slot, will be less profitable for her.

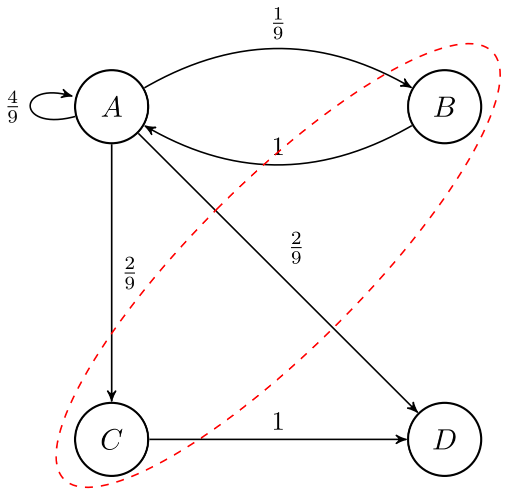
Let's denote by $C_i^{f,j}$, for $j \in \{0,1\}$, the expected success time for a pending player i given that in the last round he attempted transmission ($j = 1$) or not ($j = 0$) i.e., $C_i^{f,j} = \mathbb{E}[T_i | \vec{h}_t, f, X_{i,t} = j]$. The following claim asserts that the expected success time for Alice depends only on whether she attempted a transmission or not in the previous slot. For the proof, we compute the expected time to absorption for the Markov chain $\mathcal{M}$ shown in Figure 1a, starting from states A and B.

▶ **Claim 2.** $C_{Alice}^{f,j} = 2 + j$*, for* $j \in \{0,1\}$.

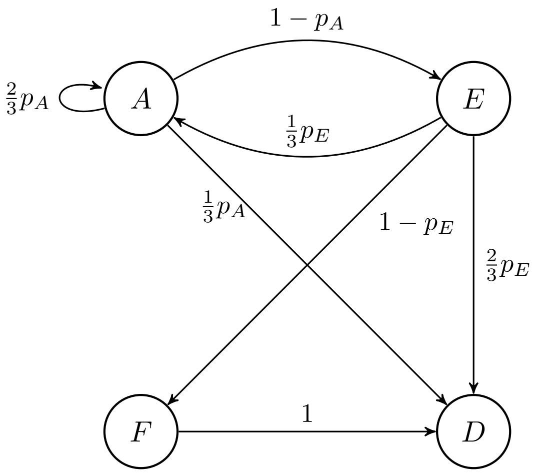
Proof. The situation from Alice's perspective can be modeled as a Markov chain $\mathcal{M}$ with state space $\{A, B, C, D\}$. A is the initial state where both players are pending (and they both know this). A is reached either in $t = 1$, or when Alice transmitted in the previous time step and there was a collision. State B models the case when both players are pending, but Alice does not know this, because she did not transmit in the previous time step. State C is reached when only Alice is pending; notice that, by definition of the protocol, there is no way for Alice to distinguish with certainty between states B and C if both herself and Bob

[3] Abusing notation slightly, we will also write $C_i^{\vec{f}}(\vec{h}_0)$ for the *unconditional* expected latency of player i induced by $\vec{f}$.

[4] For an anonymous protocol f, we denote by (f_{-i}, f'_i) the profile where agent $j \neq i$ uses protocol f and agent i uses protocol f'_i.



(a) Markov chain $\mathcal{M}$.



(b) Markov chain $\mathcal{M}'$.

Figure 1 Markov chains used in the analysis.

use f. Finally, D is the state in which Alice has successfully transmitted. The transition graph of $\mathcal{M}$ is shown in Figure 1a.

For example, we can see from the transition graph that the probability that we visit state A at time $t+1$, given that we are in B at t is given by $\Pr(\mathcal{M}_{t+1} = A|\mathcal{M}_t = B) = 1$. Indeed, if $\mathcal{M}_t = B$, neither player transmitted at t, so both will transmit with probability 1 at $t+1$, causing a collision, after which Alice (and also Bob) can deduce that all players are still pending. Similarly, $\Pr(\mathcal{M}_{t+1} = D|\mathcal{M}_t = C) = 1$, because, being at C means that only Bob transmitted (successfully) at t and so Alice will transmit (also successfully, being the only pending player) at $t+1$ with probability 1.

Clearly, $C^{f,1}_{Alice}$ is equal to the expected hitting time k^D_A that $\mathcal{M}$ needs to reach state D, given that we start from A. By definition, we have $k^D_C = 1$, $k^D_D = 0$, and by the Markov property, we get $k^D_B = k^D_A + 1$ and $k^D_A = 1 + \frac{4}{9}k^D_A + \frac{1}{9}k^D_B + \frac{2}{9}k^D_C + \frac{2}{9}k^D_D$. By rearranging terms and making the substitutions, we conclude that $C^{f,1}_{Alice} = 3$.

Calculating $C^{f,0}_{Alice}$ is a bit more tricky, because since Alice did not attempt transmission at the previous slot, she cannot be certain in which state she is, but she knows that is at state B with probability $1/3$ and in C with $2/3$. Therefore $C^{f,0}_{Alice} = \frac{1}{3}k^D_B + \frac{2}{3}k^D_C = 2$. ◀

It remains to be shown that for any transmission history up to any time t, the optimal (best-response) strategy for Alice is to follow f. Notice that this situation from Alice's point of view can be described by an infinite-horizon, undiscounted *Partially observable Markov Decision Process (POMDP)*, by the direct modification of the Markov chain $\mathcal{M}$ that is described in the proof of Claim 2. This process is partially observable due to the uncertainty created whenever Alice does not attempt transmission. This creates complications in the analysis, as general results about the existence of optimal stationary policies in MDPs [23], do not carry over immediately and also optimal policies are not always well-defined for undiscounted POMDPs with infinite horizon [22]. Fortunately, by exploiting the nature of our specific protocol f, and in particular the fact that a player using f never misses two transmissions in a row, we are able to circumvent this difficulty and model the situation as an MDP.

Following the notation in [21], the *state space* of the MDP is $\mathcal{I} = \{A, E, F, D\}$. The states are interpreted as follows: As in the Markov chain $\mathcal{M}$, state A describes the situation in which both players are pending and they both know it (this is reached just after a collision, or at time $t = 1$) and state D corresponds to the state in which Alice successfully transmitted. F is the state in which Alice did not transmit for two consecutive rounds. Since Bob follows f, he will have transmitted in one of these two rounds. Thus, in F Alice is the only pending player and she knows it. Note that in F the unique optimal strategy for Alice is to transmit in the next round. Finally, E is the state in which Alice is uncertain whether she is the only pending player in the system; this happens at t if she did not transmit at $t-1$, but transmitted at $t-2$ and there was a collision. State E essentially corresponds to a combination of states B and C in Figure 1a.

Since Alice clearly starts at state A, the *initial distribution* of the MDP is λ, where $\lambda_A = 1$ and $\lambda_E = \lambda_F = \lambda_D = 0$. The set of *actions* for Alice is $\mathcal{A} = [0, 1]$. In particular, if Alice decides to take action $a \in \mathcal{A}$ at time t, then she will transmit with probability a at t. Furthermore, the *cost function* of the MDP is $c(a) = (c_s(a) : s \in \mathcal{I})$ and we have $c_A(a) = c_E(a) = c_F(a) = 1$ and $c_D(a) = 0$ for all $a \in \mathcal{A}$. Finally, for the *transition matrix* of our MDP, notice that, since the MDP describes the situation from Alice's perspective, we calculate transition probabilities by "deferring" the relevant decisions taken by Bob until the time that Alice gets feedback . The transition matrix of our MDP is shown in equation (2) and it is explained in more detailed below.

$$P(a) = \begin{bmatrix} \frac{2a}{3} & 1-a & 0 & \frac{a}{3} \\ \frac{a}{3} & 0 & 1-a & \frac{2a}{3} \\ 0 & 0 & 1-a & a \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2}$$

In particular, we can see from (2) that the probability to visit state A in one step, given that we are at state E and the action taken is $a \in [0, 1]$, is $P_{E,A}(a) = \frac{a}{3}$. Indeed, this happens at some time t if at time $t-1$ Alice did not transmit but Bob did not transmit either; therefore, by definition of f, given that we are at E (i.e. Alice did not transmit at time $t-1$), the probability that we reach A is equal to the probability that Alice transmits at t (which happens with probability a) multiplied by the probability that Bob did not transmit at $t-1$ (which happens with probability $\frac{1}{3}$). Similarly, the probability that we visit D in one step, given that we are at state E and the action taken is $a \in [0, 1]$, is $P_{E,D}(a) = \frac{2a}{3}$, which is the probability that Alice transmits in the current step and Bob transmitted in the previous one (in which Alice did not transmit, thus Bob was successful).

By Lemma 5.4.2 and Theorem 5.4.3 from [21], there is a stationary policy (i.e. protocol) u^* that is optimal in the sense that it achieves the minimum expected total cost, given that we start at state A. The fact that u^* is stationary significantly reduces the search space of optimal strategies. In particular, this allows us to only consider strategies for which the actions taken by Alice (in the above MDP) depend only on the current state. In fact, we can further reduce the family of optimal strategies considered by noting that in any optimal strategy Alice will transmit with probability 1 when in state F; indeed, when Alice knows that she is the only pending player, she will decide to transmit with probability 1 in the next time step. Therefore, it only remains to determine the probability of transmission when we are at either state A of E; denote those by p_A and p_E respectively. Therefore, this leads to a Markov chain $\mathcal{M}'$ with state space $\mathcal{I}' = \mathcal{I}$ and transition probabilities that correspond to actions from the above MDP. The transition graph of $\mathcal{M}'$ is shown in Figure 1b.

Clearly, the expected latency of Alice when she uses protocol u^* and Bob uses protocol f is equal to the expected hitting time $k_A'^D$ that $\mathcal{M}'$ needs to reach state D, given that we start from A. By definition, we have $k_F'^D = 1, k_D'^D = 0$, and by the Markov property, we get

$k_A^{'D} = 1 + \frac{2}{3}p_A k_A^{'D} + (1-p_A)k_E^{'D}$ and $k_E^{'D} = 1 + \frac{1}{3}p_E k_A^{'D} + (1-p_E)k_F^{'D}$. Rearranging and after substitutions we get $k_A^{'D} = 3$ and $k_E^{'D} = 2$, for any $p_A, p_E \in [0,1]$. Comparing this to Claim 2, we conclude that if Bob uses f, a best response for Alice is to also follow f. This completes the proof of the Theorem. ◀

3.1 Uniqueness

We will say that a protocol is *stationary* if the decision rule for each player at some time t depends on the information state of the player at t. In particular, the protocol defined in equation (1) is stationary. In this section we show that there are no other stationary equilibria.

▶ **Theorem 3.** *For two players, the unique stationary anonymous protocol that is in equilibrium is the one defined in equation (1).*

Proof. For the sake of contradiction, assume that there is another stationary protocol that is in equilibrium. As in the analysis of protocol (1) in Section 3, we denote by A the state where both players know they are both pending. Let Alice be one of the two players. Notice that, every time Alice transmits, either there is a collision (in which case Alice returns to state A) or the transmission is successful (so Alice is no longer pending).

For $k = 1, 2, \dots$, let p_k denote the probability that Alice transmits in step k, given that she starts from A at $t = 0$ and she does not transmit in time steps 1 to $k-1$. Therefore, given that we start from A at time 0, the probability that Alice attempts to transmit for the first time after k steps is $p_k \prod_{k'=1}^{k-1}(1-p_{k'})$. In particular, in the equilibrium described in the previous section, we had $p_1 = \frac{2}{3}$ and $p_2 = 1$.

First, assume there is another stationary protocol g that is in equilibrium, for which $p_2 = 1$ and $p_1 = p \neq \frac{2}{3}$. Adjusting the transition probabilities in the Markov chain in Figure 1a accordingly, and doing the same analysis we can derive that the expected latency of Alice when both players use protocol g is $k_A^D = \frac{2-p}{2p(1-p)}$. We will show that for all $p \neq \frac{2}{3}$ a player has a profitable deviation. Indeed, first observe that $p > \frac{2}{3}$ implies $k_A^D > 3$. In this case Alice can improve her expected latency by not transmitting for two consecutive time steps and then (successfully) transmitting in the third time step. Second, for the case that $p < \frac{2}{3}$, persistently transmitting in each time step is a deviation which gives the deviator an expected latency of $\frac{1}{1-p}$. For $p < \frac{2}{3}$ this is strictly less than the expected latency $k_A^D = \frac{2-p}{2p(1-p)}$ that Alice has when both players use protocol g. From both cases, we conclude that there is no stationary protocol in equilibrium for which $p_2 = 1$ and $p_1 \neq \frac{2}{3}$.

Now assume that there is another stationary protocol z in equilibrium, for which $p_2 < 1$. Denote α_z the expected latency of Alice when both players use protocol z. Similarly denote $\alpha_{(z')}$ the expected latency of Alice when she unilaterally deviates from z to some other protocol z'. We will consider the following three protocols that Alice can use instead of z: (i) Using protocol $(1z)$, Alice will transmit in the first time step and then continue by following protocol z. (ii) Using protocol $(01z)$, Alice will not transmit in the first time step, but will transmit in the second time step and then follow the protocol z. (iii) Finally, using protocol $(001z)$, Alice will not transmit for the first two time steps, but will transmit in the third time step and then follow the protocol z. The expected latency of Alice when she uses each of those protocols while the other player uses z is given by:

$$\begin{aligned} \alpha_{(1z)} &= 1 + p_1\alpha_z \\ \alpha_{(01z)} &= 2 + (1-p_1)p_2\alpha_z \\ \alpha_{(001z)} &= 3 + (1-p_1)(1-p_2)p_3\alpha_z. \end{aligned}$$

Notice now that all three transmission sequences $(1), (0,1)$ and $(0,0,1)$ are consistent with z. Furthermore, z is acknowledgment-based, so Lemma 4 applies here. Therefore, the above expected latencies must all be equal to α_z. Using the identities $\alpha_z = \alpha_{(1z)} = \alpha_{(01z)}$ we get that $\alpha_z = 2 + p_2 < 3$. But clearly $3 \leq \alpha_{(001z)}$, which is a contradiction to the fact that $\alpha_z = \alpha_{(001z)}$. Thus, there is no equilibrium protocol with $p_2 < 1$. This completes the proof of the theorem. ◀

4 Age-based and backoff protocols

In this section, we focus on two special prominent classes of acknowledgment-based protocols, namely *age-based* and *backoff*, and we show that these cannot be implemented in equilibrium if we insist on finite expected latency.

In what follows, for any protocol f, any player i that uses f and any time t, we will say that $\vec{h}_{i,t}$ is *consistent with* f if and only if there is a non-zero probability that $\vec{h}_{i,t}$ will occur for player i.

Now we are ready to show in the next Lemma a useful property of all *acknowledgment-based* equilibrium protocols that is essentially an analogue of the property of Nash equilibria for finite games that all pure strategies in the support of a Nash equilibrium are best responses.

▶ **Lemma 4.** *Let $f \overset{def}{=} \{f_t\}_{t\geq 1}$ be an anonymous acknowledgment-based protocol and let $\pi \overset{def}{=} \pi_1, \pi_2, \ldots$ be any 0-1 sequence which is consistent with f. For any (finite) positive integer τ^*, define the protocol*

$$g = g(\tau^*) \overset{def}{=} \begin{cases} \pi_t, & \text{for } 1 \leq t \leq \tau^* \\ f_t, & \text{for } t > \tau^*. \end{cases} \tag{3}$$

We then have that, for any fixed player i, if f is in equilibrium, then

$$C_i^f(\vec{h}_0) = C_i^{(f_{-i},g)}(\vec{h}_0).$$

Proof. Since we consider acknowledgment-based protocols, for the sake of the analysis, we will assume that players continue to flip coins even after successfully transmitting, so that they eventually find out what their decisions would have been at any time t.[5]

For a fixed player i, we obtain

$$C_i^f(\vec{h}_0) = \mathbb{E}[T_i|\vec{h}_{i,0}, f] = \sum_{\vec{h}_{i,\tau^*}} \mathbb{E}[T_i|\vec{h}_{i,\tau^*}, f] \Pr\left\{\vec{h}_{i,\tau^*} \text{ happens for } i\right\}. \tag{4}$$

Notice now that, since f is acknowledgment-based, the event $\left\{\vec{h}_{i,\tau^*} \text{ happens for } i\right\}$ is independent of the transmission sequences of other players. Therefore, $\mathbb{E}[T_i|\vec{h}_{i,\tau^*}, f]$ is equal to the unconditional (i.e. conditioned on $\vec{h}_{i,0}$) expected latency of player i when she uses the protocol defined in equation (3), where the first τ^* terms of π are replaced by $(\pi_1, \ldots, \pi_{\tau^*}) = (\vec{X}_{i,1}, \ldots, \vec{X}_{i,\tau^*}) = \vec{h}_{i,\tau^*}$.[6] In particular, we have that $\mathbb{E}[T_i|\vec{h}_{i,\tau^*}, f] = \mathbb{E}[T_i|\vec{h}_{i,0}, (f_{-i}, g)] = C_i^{(f_{-i},g)}(\vec{h}_0)$.

[5] In fact, we only need this assumption to hold for any t which is at most some predefined fixed upper bound τ^*.

[6] Note that this observation is not true for general protocols and different kinds of feedback, which is why the present analysis cannot be used to prove an impossibility result in the case of protocols like those in [9].

Assume now for the sake of contradiction that there is a transmission history $\vec{h}_{i,\tau^*}$ for player i such that $\mathbb{E}[T_i|\vec{h}_{i,\tau^*}, f] \neq \mathbb{E}[T_i|\vec{h}_{i,0}, f]$. Clearly, if $\mathbb{E}[T_i|\vec{h}_{i,\tau^*}, f] < \mathbb{E}[T_i|\vec{h}_{i,0}, f]$, then the protocol g is a better protocol for player i, which contradicts the fact that f is in equilibrium. On the other hand, if $\mathbb{E}[T_i|\vec{h}_{i,\tau^*}, f] > \mathbb{E}[T_i|\vec{h}_{i,0}, f]$, then equation (4) implies that there must be another transmission history $\vec{h}'_{i,\tau^*}$ for which $\mathbb{E}[T_i|\vec{h}'_{i,\tau^*}, f] < \mathbb{E}[T_i|\vec{h}_{i,0}, f]$.

Therefore, we have that $C_i^{(f_{-i},g)}(\vec{h}_0) = \mathbb{E}[T_i|\vec{h}_{i,0}, (f_{-i}, g)] = \mathbb{E}[T_i|\vec{h}_{i,\tau^*}, f] = \mathbb{E}[T_i|\vec{h}_{i,0}, f] = C_i^f(\vec{h}_0)$, for any transmission history $\vec{h}_{i,\tau^*}$, and for any finite $\tau^* \geq 1$, thus also for any 0-1 sequence π that is consistent with f. ◀

The next corollary is an interesting consequence of Lemma 4 regarding *non-blocking* anonymous age-based protocols.

▶ **Corollary 5.** *Let $f \overset{def}{=} \{f_t\}_{t\geq 1}$ be a non-blocking anonymous age-based protocol. If the expected latency of a player using protocol f is finite, i.e. $\mathbb{E}[T_i|\vec{h}_{i,0}, f] < \infty$, then f is not in equilibrium.*

Proof. Assume for the sake of contradiction that f is in equilibrium and let $\tau^* \overset{def}{=} \left\lfloor \mathbb{E}[T_i|\vec{h}_{i,0}, f] \right\rfloor$ be finite, where i is a fixed player using f. Consider the protocol $g = g(\tau^*)$ as defined in (3), where the first τ^* terms of π are set equal to 0. Clearly, any player using g has expected latency at least $\tau^* + 1$, irrespectively of the transmissions of the other players. Notice also that π is consistent with f up to τ^*, since $\Pr\{\vec{h}_{i,\tau^*} = (0, \dots, 0)|f\} = \prod_{t=1}^{\tau^*}(1 - p_{i,t}) > 0$. Therefore, by Lemma 4 we have that $\tau^* + 1 > \mathbb{E}[T_i|\vec{h}_{i,0}, f] = \mathbb{E}[T_i|\vec{h}_{i,0}, (f_{-i}, g)] \geq \tau^* + 1$, which is a contradiction. We conclude that either f is not in equilibrium, or τ^* is ∞. ◀

We are now ready to show the main result of this section.

▶ **Theorem 6.** *There is no anonymous age-based protocol f for $n \geq 2$ players that is in equilibrium and has $\mathbb{E}[T_i|\vec{h}_{i,0}, f] < \infty$, for any player i.*

Proof. For the sake of contradiction, let's assume that $f = \{f_t\}_{t\geq 1}$ is an age-based protocol in equilibrium with finite expected latency, i.e. $\mathbb{E}[T_i|\vec{h}_0, f] < \infty$. The next claim asserts the existence of a finite positive integer τ^* where the protocol dictates transmission, with certain properties, which will be a useful ingredient for the rest of the proof.

▶ **Claim 7.** *Let f be an anonymous age-based protocol for n players that is in equilibrium and has $\mathbb{E}[T_i|\vec{h}_{i,0}, f] < \infty$, then there is a finite positive integer τ^* such that*
(a) $f_{\tau^*} = 1$,
(b) $f_{\tau^*-1} < 1$ *and*
(c) *there exist $\tau_1 < \cdots < \tau_{n-1} < \tau^*$, such that $f_{\tau_j} < 1$, for all $j = 1, \dots, n-1$.*

Proof. For any time t, define Z_t^f to be the number of non-blocking probabilities of the protocol f up to t, i.e. $Z_t^f \overset{def}{=} \sum_{t'\leq t}(1 - \lfloor f_{t'} \rfloor)$. Set $\tau' \overset{def}{=} \inf\{t : f_t = 1, Z_t^f \geq n-1\}$. Assume for the sake of contradiction that there does not exist a τ^* with the properties described in the claim. In particular, this means that $\tau' = \infty$. However, the latter can happen if one of the following cases is true:
(i) There is no finite τ such that $f_\tau = 1$.
(ii) There exists finite τ such that $f_\tau = 1$, $Z_t^f \leq n-2$ and $f_t = 1$, for all $t \geq \tau$.
(iii) There exists finite τ such that $f_\tau = 1$, $Z_t^f \leq n-2$ and $f_t < 1$, for all $t \geq \tau$.

We now prove that in all those cases we get a contradiction. Case (i) comes in contradiction with Corollary 5.

If case (ii) holds, then clearly, if all players use f, at most $n-2$ players can successfully transmit before τ and the rest will remain pending for ever. But this means that the expected latency of a player i using f is at least

$$\Pr\{i \text{ does not successfully transmit before } \tau | \vec{h}_{i,0}, f\} \cdot \infty = \infty,$$

which leads to a contradiction, since we assumed $\mathbb{E}[T_i|\vec{h}_{i,0}, f] < \infty$.

Suppose now that case (iii) holds. Consider the protocol g defined as follows:

$$g \overset{def}{=} \begin{cases} 0, & \text{if } f_t < 1, \text{ for } 1 \leq t \leq \mathbb{E}[T_i|\vec{h}_{i,0}, f] \\ 1, & \text{if } f_t = 1, \text{ for } 1 \leq t \leq \mathbb{E}[T_i|\vec{h}_{i,0}, f] \\ f_t, & \text{for } t > \mathbb{E}[T_i|\vec{h}_{i,0}, f]. \end{cases} \tag{5}$$

Let i be a fixed player (say Alice). Notice that, if all other players use f and Alice uses g, then Alice has expected latency strictly larger than $\mathbb{E}[T_i|\vec{h}_{i,0}, f]$; indeed, for any $t \leq \mathbb{E}[T_i|\vec{h}_{i,0}, f]$, Alice only attempts a transmission when $f_t = 1$ and there is at least one more other pending player using f, and so there is a collision. However, since the initial (deterministic) sequence of $\left\lfloor \mathbb{E}[T_i|\vec{h}_{i,0}, f] \right\rfloor$ transmissions of g is consistent with f, by Lemma 4 we have that $\mathbb{E}[T_i|\vec{h}_{i,0}, f] = \mathbb{E}[T_i|\vec{h}_{i,0}, (f_{-i}, g)] > \mathbb{E}[T_i|\vec{h}_{i,0}, f]$, which is a contradiction. This completes the proof of the claim. ◀

Take a τ^* as described in the above claim and consider the protocol Q defined as follows

$$Q \overset{def}{=} \begin{cases} 0, & \text{if } f_t < 1, \text{ for } 1 \leq t \leq \tau^* - 2 \\ 1, & \text{if } f_t = 1, \text{ for } 1 \leq t \leq \tau^* - 2 \\ 1, & \text{for } t = \tau^* - 1 \text{ and } t = \tau^* \\ f_t, & \text{for } t > \tau^*. \end{cases} \tag{6}$$

Notice that, since the initial (deterministic) sequence of transmissions of Q is consistent with f, by Lemma 4 we have that $\mathbb{E}[T_i|\vec{h}_{i,0}, f] = \mathbb{E}[T_i|\vec{h}_{i,0}, (f_{-i}, Q)]$.

Now consider the protocol Q', which is the same as Q, with the only difference[7] that $Q'_{\tau^*} = 0$. In fact, we show that, $\mathbb{E}[T_i|\vec{h}_{i,0}, (f_{-i}, Q')] < \mathbb{E}[T_i|\vec{h}_{i,0}, (f_{-i}, Q)]$ which implies $\mathbb{E}[T_i|\vec{h}_{i,0}, (f_{-i}, Q')] < \mathbb{E}[T_i|\vec{h}_{i,0}, f]$, which contradicts the assumption that f is in equilibrium.

Notice now that protocols Q and Q' are identical for any $t \neq \tau^*$, and if there are at least 3 pending players at τ^* (i.e. Alice and at least two others), then there would be a collision at τ^* no matter which of the two protocols Alice uses (i.e. the same players that were pending at τ^* would be pending at the start of time slot $\tau^* + 1$ as well). Therefore, the two protocols behave the same in this case. However, if there are exactly 2 pending players at τ^* (i.e. Alice and exactly one more, say Bob) the two protocols behave differently. Indeed, if Alice uses protocol Q, then there will be a collision at τ^*, leaving exactly 2 pending players at $\tau^* + 1$. However, if Alice uses protocol Q', then Bob will be able to successfully transmit at τ^*, leaving Alice the only pending player at time $\tau^* + 1$, which implies a strictly smaller expected latency. The proof is completed by noting that, by definition of τ^*, the probability that there will be exactly 2 players pending at τ^* is strictly positive (since there are at least $n-2$ steps before $\tau^* - 1$ with transmission probability strictly less than 1). ◀

[7] Note that Q' does not agree with f whenever $f_t = 1$, so Lemma 4 does not apply to Q'.

Now we conclude with the impossibility result for backoff protocols, the proof of which shares similarities to the proof of Corollary 5.

▶ **Theorem 8.** *There is no anonymous backoff protocol f in equilibrium for $n \geq 2$ players with $\mathbb{E}[T_i|\vec{h}_{i,0}, f] < \infty$, for any player i.*

Proof. Assume for the sake of contradiction that f is in equilibrium and let $\tau^* \overset{def}{=} \left\lfloor \mathbb{E}[T_i|\vec{h}_{i,0}, f] \right\rfloor$ be finite, where i is a fixed player using f. By definition, we have that $f_i = \{p_{i,k}\}_{k\geq 0}$, where $p_{i,k}$ denotes the transmission probability of player i after k unsuccessful transmissions. Notice also that we may assume without loss of generality that $p_{i,0} \neq 1$. Indeed, suppose there is finite integer $s > 0$, such that $p_{i,k'} = 1$, for all $k' < s$ and $p_{i,s} \neq 1$ (if s is not finite, then clearly f does not have finite expected latency). Then the protocol $f' = \{p'_{i,k}\}_{k\geq 0}$, with $p'_{i,k} = p_{i,k+s}$, for all $k \geq 0$ is also an equilibrium.

Consider now the protocol $g = g(\tau^*)$ defined in equation (3), where the first τ^* terms of π are set to 0. Clearly, any player using g has expected latency at least $\tau^* + 1$. Notice also that π is consistent with f up to τ^*, since $\Pr\{\vec{h}_{i,\tau^*} = (0, \dots, 0)|f\} = (1 - p_{i,0})^{\tau^*} > 0$. Therefore, by Lemma 4 we have that $\tau^* + 1 > \mathbb{E}[T_i|\vec{h}_{i,0}, f] = \mathbb{E}[T_i|\vec{h}_{i,0}, (f_{-i}, g)] \geq \tau^* + 1$, which is a contradiction. But this implies that, either f is not in equilibrium, or τ^* is ∞. ◀

5 An efficient protocol in equilibrium

In this section we present a deadline protocol for n players that is efficient, i.e. with high probability the latency of any player is $\Theta(n)$. Let $t_0 = t_0(n)$ be an integer, to be determined later and let $\beta \in (0, 1)$ be a fixed constant. We consider the following deadline protocol $\mathcal{Q}$ with deadline t_0, which is defined as follows: The $t_0 - 1$ time steps before the deadline are partitioned into $k + 1$ consecutive intervals $I_1, I_2, \dots, I_{k+1}$, where $k = k(n)$ is the unique integer satisfying $\beta^{k+1}n \leq \sqrt{n} < \beta^k n$. For any $j \in \{1, \dots, k+1\}$, define $n_j = \beta^j n$. For $j \in \{1, \dots, k\}$ the length of interval I_j is $\ell_j = \left\lfloor \frac{e}{\beta} n_j \right\rfloor$. Interval I_{k+1} is special and has length $\ell_{k+1} = n$. In particular, this gives

$$t_0 \overset{def}{=} 1 + \sum_{j=1}^{k+1} \ell_j \leq 1 + n + en \sum_{j=1}^{k} \beta^{j-1} = 1 + n + en \frac{1 - \beta^{k-1}}{1 - \beta} \leq n \left(1 + \frac{e}{1 - \beta}\right),$$

where the last inequality holds for any constant $\beta \in (0, 1)$ and $n \to \infty$. For any $t \geq 1$, the decision rule at time t for protocol $\mathcal{Q}$ is given by

$$\mathcal{Q}_t = \begin{cases} \frac{1}{n_j}, & \text{if } t \in I_j, j = 1, 2, \dots, k+1 \\ 1, & \text{if } t \geq t_0. \end{cases} \tag{7}$$

Notice that, by definition, $\mathcal{Q}$ is an age-based protocol. Furthermore, if at least two out of n players use protocol $\mathcal{Q}$, then, no matter what protocol the rest of the players use, there is a non-zero probability that there will be no successful transmission until the deadline t_0, and thus all players will remain pending for ever. In particular, this is at least the probability that the two players using $\mathcal{Q}$ attempt a transmission in every step until t_0, which happens with probability $\prod_{t=1}^{t_0-1} (\mathcal{Q}_t)^2 \geq \frac{1}{n^{t_0}} > 0$. Therefore, if there are at least two players using $\mathcal{Q}$, the expected latency of any player is ∞, hence $\mathcal{Q}$ is in equilibrium, for any $n \geq 3$ and deadline t_0.

In Theorem 11 we prove that $\mathcal{Q}$ is also efficient; when all players in the system use protocol $\mathcal{Q}$, then with high probability all players will successfully transmit before the deadline t_0.

For the proof, we use two elementary Lemmas that formalize the fact that, in each interval, a significant number of players successfully transmit with high probability. For the proofs, we employ standard concentration results from probability theory.

▶ **Lemma 9.** *Assume that all players in the system use protocol $\mathcal{Q}$. For any $j \in \{1, \ldots, k\}$, if the number of pending players before interval I_j is at most n_j, then after I_j, with probability at least $1 - \exp(-\frac{1}{3}\beta^{j+2}n)$ there will be at most n_{j+1} pending players.*

Proof. Fix $j \in \{1, \ldots, k\}$ and assume that the precondition of the lemma is fulfilled, i.e., before interval I_j there are at most n_j pending players. Let r_t denote the number of pending players at time t. In particular, for any $t \in I_j$, if the preconditions of the lemma is fulfilled, we have $r_t \leq n_j$. Therefore the probability of a successful transmission in round $t \in I_j$ is given by

$$r_t\mathcal{Q}_t(1-\mathcal{Q}_t)^{r_t-1} \geq r_t\mathcal{Q}_t(1-\mathcal{Q}_t)^{n_j-1} = r_t\frac{1}{n_j}\left(1-\frac{1}{n_j}\right)^{n_j-1} \geq \frac{1}{e}\frac{r_t}{n_j},$$

where in the last inequality we used the fact that $\left(1-\frac{1}{x}\right)^{x-1} \geq \frac{1}{e}$, for any $x > 1$. Therefore, for any round $t \in I_j$, either we already have $r_t \leq n_{j+1} = \beta n_j$ pending players, or the probability of a successful transmission in round t is at least $a \overset{def}{=} \frac{1}{e}\frac{n_{j+1}}{n_j} = \frac{\beta}{e}$.

Let now X_j be the random variable counting the number of successful transmissions in interval I_j. Notice that, by the above discussion, given that at the start of interval I_j there are at least n_{j+1} pending players, X_j stochastically dominates a Binomial random variable $Y_j \sim Bin(\ell_j, a)$, with mean value $\ell_j \cdot a$. Therefore, by a Chernoff bound (see [26]), we get

$$\Pr(X_j < (1-\beta)\ell_j \cdot a) \leq \Pr(Y_j < (1-\beta)\ell_j \cdot a) \leq \exp\left(-\frac{1}{2}\beta^2\ell_j \cdot a\right) \leq \exp\left(-\frac{1}{3}\beta^2 n_j\right),$$

where in the last inequality we used the fact that, by definition, $n_j \geq \sqrt{n}$, for all $j \leq k$, thus $\ell_j \cdot \frac{\beta}{e} \geq \frac{2}{3}n_j$. This directly implies the lemma. ◀

▶ **Lemma 10.** *If the number of pending players at the start of interval I_{k+1} is at most n_{k+1}, then after interval I_{k+1}, with probability at least $1 - \exp\left(-\frac{1}{3}n_{k+1}\right)$ all players will have successfully transmitted.*

Proof. Consider a fixed player (say Alice) that is pending at the start of interval I_{k+1}. Given that there are at most $n_{k+1} = \beta^{k+1}n$ pending players at any time step $t \in I_{k+1}$, the probability that Alice successfully transmits during t is at least

$$\mathcal{Q}_t(1-\mathcal{Q}_t)^{n_{k+1}-1} = \frac{1}{n_{k+1}}\left(1-\frac{1}{n_{k+1}}\right)^{n_{k+1}-1}.$$

Therefore, since $|I_{k+1}| = \ell_{k+1} = n$, the probability that Alice is still pending after interval I_{k+1} is at most

$$\left(1-\frac{1}{n_{k+1}}\left(1-\frac{1}{n_{k+1}}\right)^{n_{k+1}-1}\right)^n \leq \exp\left(-\frac{n}{n_{k+1}}\left(1-\frac{1}{n_{k+1}}\right)^{n_{k+1}-1}\right). \quad (8)$$

Recall that, by definition, k is the (unique) smallest integer satisfying $n_{k+1} \leq \sqrt{n} < n_k$. In particular, this implies that $n_{k+1} > \beta\sqrt{n}$, therefore n_{k+1} goes to ∞ as $n \to \infty$. Additionally, we have that $\frac{n}{n_{k+1}} \geq n_{k+1}$. Therefore, using the fact that $\left(1-\frac{1}{x}\right)^{x-1} \geq \frac{1}{e}$, for any $x > 1$, the right hand side of (8) is at most $\exp\left(-\frac{1}{e}n_{k+1}\right)$.

By the union bound, given that there are at most n_{k+1} pending players at the start of interval I_{k+1}, the probability that there is at least one pending player after I_{k+1} is at most $n_{k+1} \exp\left(-\frac{1}{e}n_{k+1}\right) \leq \exp\left(-\frac{1}{3}n_{k+1}\right)$, as stated in the Lemma. ◀

We are now ready to prove our main Theorem.

▶ **Theorem 11.** *Protocol $\mathcal{Q}$ is efficient. In particular, for any constant $\beta \in (0,1)$, when all players use $\mathcal{Q}$, the probability that there is a pending player after time $t_0 \leq n\left(1+\frac{e}{1-\beta}\right)$ is at most $\exp(-\Theta(\sqrt{n}))$.*

Proof. It suffices to show that with high probability every player will have successfully transmitted before t_0. Note that, the probability that there are still pending players at $t_0 = \Theta(n)$ is upper bounded by the probability that (a) there exists $j \in \{1, 2, \ldots, k\}$ such that, at the end of interval I_j there are more than n_{j+1} pending players, or (b) there are still pending players after interval I_{k+1}.

Therefore, by Lemma 9 and Lemma 10 and the union bound, the probability that not all players successfully transmit before t_0 is at most

$$\exp\left(-\frac{1}{3}n_{k+1}\right) + \sum_{j=1}^{k} \exp\left(-\frac{1}{3}\beta^2 n_j\right). \tag{9}$$

Since $n_j \geq n_{k+1} \geq \beta\sqrt{n}$, for any $j \in \{1, 2, \ldots, k\}$, the above upper bound becomes $(k+1)\exp\left(-\Theta(\sqrt{n})\right)$. The proof is concluded by noting that, by definition of k, we have $k = \Theta(\log n)$. ◀

We note that, in our analysis, $\beta \in (0,1)$ can be any constant arbitrarily close to 0, therefore, by Theorem 11, the upper bound on the latency of protocol $\mathcal{Q}$ can be as small as $(1+e)n + o(n)$ with high probability.

References

1 N. Abramson. The ALOHA system: Another alternative for computer communications. In *Proceedings of the November 17-19, 1970, fall joint computer conference*, pages 281–285. ACM New York, NY, USA, 1970.

2 E. Altman, R. El Azouzi, and T. Jiménez. Slotted aloha as a game with partial information. *Comput. Netw.*, 45(6):701–713, 2004. doi:10.1016/j.comnet.2004.02.013.

3 E. Altman, D. Barman, A. Benslimane, and R. El Azouzi. Slotted aloha with priorities and random power. In *Proc. IEEE Infocom*, 2005.

4 V. Auletta, L. Moscardelli, P. Penna, and G. Persiano. Interference games in wireless networks. In *WINE*, pages 278–285, 2008.

5 M. Bender, M. Farach-Colton, S He, B. Kuszmaul, and C. Leiserson. Adversarial contention resolution for simple channels. In *SPAA'05*, pages 325–332. ACM, 2005. doi:10.1145/1073970.1074023.

6 J. Capetanakis. Generalized tdma: The multi-accessing tree protocol. *IEEE Transactions on Communications*, 27(10):1476–1484, 1979.

7 J. Capetanakis. Tree algorithms for packet broadcast channels. *IEEE Transactions on Information Theory*, 25(5):505–515, 1979.

8 George Christodoulou, Katrina Ligett, and Evangelia Pyrga. Contention resolution under selfishness. *Algorithmica*, 70(4):675–693, 2014.

9 A. Fiat, Y. Mansour, and U. Nadav. Efficient contention resolution protocols for selfish agents. In *SODA'07*, pages 179–188, Philadelphia, PA, USA, 2007. SIAM.

10 Mihály Geréb-Graus and Thanasis Tsantilas. Efficient optical communication in parallel computers. In *SPAA'92*, pages 41–48, New York, NY, USA, 1992. ACM. doi:10.1145/140901.140906.

11 L. A. Goldberg and P. D. MacKenzie. Analysis of practical backoff protocols for contention resolution with multiple servers. *J. Comput. Syst. Sci.*, 58(1):232–258, 1999. doi:10.1006/jcss.1998.1590.

12 L. A. Goldberg, P. D. Mackenzie, M. Paterson, and A. Srinivasan. Contention resolution with constant expected delay. *J. ACM*, 47(6):1048–1096, 2000. doi:10.1145/355541.355567.

13 Leslie Ann Goldberg. Notes on contention resolution. *http://www.cs.ox.ac.uk/people/leslieann.goldberg/contention.html*, 2002.

14 A. Greenberg and S. Winograd. A lower bound on the time needed in the worst case to resolve conflicts deterministically in multiple access channels. *J. ACM*, 32(3):589–596, 1985. doi:10.1145/3828.214125.

15 Hayes J. An adaptive technique for local distribution. *IEEE Transactions on Communications*, 26(8):1178–1186, 1978.

16 Elias Koutsoupias and Katia Papakonstantinopoulou. Contention issues in congestion games. In *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II*, pages 623–635, 2012. doi:10.1007/978-3-642-31585-5_55.

17 R.T. Ma, V. Misra, and D. Rubenstein. Modeling and analysis of generalized slotted-aloha mac protocols in cooperative, competitive and adversarial environments. In *ICDCS'06*, page 62, Washington, DC, USA, 2006. IEEE. doi:10.1109/ICDCS.2006.56.

18 P. D. MacKenzie, C. G. Plaxton, and R. Rajaraman. On contention resolution protocols and associated probabilistic phenomena. *J. ACM*, 45(2):324–378, 1998. doi:10.1145/274787.274816.

19 I. Menache and N. Shimkin. Efficient rate-constrained nash equilibrium in collision channels with state information. In *INFOCOM 2008.*, pages 403–411, 2008.

20 R. Metcalfe and D. Boggs. Distributed packet switching for local computer networks. *Communications of the ACM*, 19:395–404, 1976.

21 J.R. Norris. *Markov Chains.* Cambridge University Press, 1998.

22 Loren K. Platzman. Optimal infinite-horizon undiscounted control of finite probabilistic systems. *SIAM Journal on Control and Optimization*, 18(4):362–380, 1980.

23 Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons, Inc., 1994.

24 P. Raghavan and E. Upfal. Stochastic contention resolution with short delays. Technical report, Weizmann Science Press of Israel, Jerusalem, Israel, Israel, 1995.

25 L. Roberts. Aloha packet system with and without slots and capture. *SIGCOMM Comput. Commun. Rev.*, 5(2):28–42, April 1975. doi:10.1145/1024916.1024920.

26 Sheldon R. Ross. *A First Course in Probability.* Pearson, 2012.

27 F.A. Tobagi and L. Kleinrock. Packet switching in radio channels: Part ii–the hidden terminal problem in carrier sense multiple-access and the busy-tone solution. *IEEE Transactions on Communications*, 23(12):1417–1433, 1975.

28 B. S. Tsybakov and V. A. Mikhailov. Free synchronous packet access in a broadcast channel with feedback. *Problems of Information Transmission*, 14(4):259–280, 1978.

29 D. Wang, C. Comaniciu, and U. Tureli. Cooperation and fairness for slotted aloha. *Wirel. Pers. Commun.*, 43(1):13–27, 2007. doi:10.1007/s11277-006-9240-5.

30 D. Zheng, W. Ge, and J. Zhang. Distributed opportunistic scheduling for ad-hoc communications: an optimal stopping approach. In *MobiHoc'07*, pages 1–10. ACM, 2007. doi:10.1145/1288107.1288109.

Cell-Probe Lower Bounds for Bit Stream Computation

Raphaël Clifford[1], Markus Jalsenius[2], and Benjamin Sach[3]

1 Department of Computer Science, University of Bristol, Bristol, UK
2 Department of Computer Science, University of Bristol, Bristol, UK
3 Department of Computer Science, University of Bristol, Bristol, UK

Abstract

We revisit the complexity of online computation in the cell probe model. We consider a class of problems where we are first given a fixed pattern F of n symbols and then one symbol arrives at a time in a stream. After each symbol has arrived we must output some function of F and the n-length suffix of the arriving stream. Cell probe bounds of $\Omega(\delta \lg n/w)$ have previously been shown for both convolution and Hamming distance in this setting, where δ is the size of a symbol in bits and $w \in \Omega(\lg n)$ is the cell size in bits. However, when δ is a constant, as it is in many natural situations, the existing approaches no longer give us non-trivial bounds.

We introduce a *lop-sided information transfer* proof technique which enables us to prove meaningful lower bounds even for constant size input alphabets. Our new framework is capable of proving amortised cell probe lower bounds of $\Omega(\lg^2 n/(w \cdot \lg\lg n))$ time per arriving *bit*. We demonstrate this technique by showing a new lower bound for a problem known as pattern matching with address errors or the L_2-rearrangement distance problem. This gives the first non-trivial cell probe lower bound for any online problem on bit streams that still holds when the cell size is large.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Cell-probe lower bounds, algorithms, data streaming

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.31

1 Introduction

We revisit the complexity of online computation in the cell probe model. In recent years there has been considerable progress towards the challenging goal of establishing lower bounds for both static and dynamic data structure problems. A third class of data structure problems which fall somewhere between these two classic settings, is online computation in a streaming setting. Here one symbol arrives at a time and a new output must be given after each symbol arrives and before the next symbol is processed. The key conceptual difference to a standard dynamic data structure problem is that although each arriving symbol can be regarded as a new update operation at a prespecified index, there is only one type of query which is to output the latest value of some function of the stream.

Online pattern matching is particularly suited to study in this setting and cell probe lower bounds have previously been shown for different measures of distance including Hamming distance, inner product/convolution and edit distance [3, 4, 5]. All these previous cell probe lower bounds have relied on only one proof technique, the so-called *information transfer technique* of Pătraşcu and Demaine [14]. In loose terms the basic idea is as follows. First one defines a random input distribution over updates. Here we regard an arriving symbol as an update and after each update we perform one query which simply returns the latest distance between a predefined pattern and the updated suffix of the stream. One then has to argue that knowledge of the answers to ℓ consecutive queries is sufficient to infer at least a constant fraction of the information encoded by ℓ consecutive updates that occurred in the past. If one can show this is true for all power of two lengths ℓ then a logarithmic lower bound per update/query operation follows.

In recent years a consensus has been arrived at that the most natural cell size is $w \in \Omega(\lg n)$. This is for two main reasons. The first is simply that a cell should be large enough to be able to address all of memory. The second, more practical reason is that lower bounds that we derive directly give time lower bounds for problems analysed in the popular word-RAM model. When cells are of this size a cell probe lower bound of $\Omega(\delta \lg n/w)$ for both online Hamming distance and convolution using the information transfer technique has been shown, where δ is the number of bits in an input symbol, w is the cell size in bits and n is the length of the fixed pattern [3, 4]. When $\delta \geq w \geq \lg n$, there is also a matching upper bound in the cell probe model and so no further progress is possible. However, when the symbol size δ does not grow with the input size as is often the case in applied settings, the best lower bound that is derivable reduces trivially to be constant. This is an unfortunate situation as a particularly natural setting of parameters is when the input alphabet is of constant size but the cell size is not.

This small input alphabet, large cell size setting has received some study in the past. Using a sophisticated variant of the information transfer technique, Pătraşcu and Demaine [14] proved an $\Omega(\lg n/ \lg\lg n)$ cell probe lower bound for the classic prefix sum problem when the random update *values* contain $\delta = O(1)$ bits and the cell size is $\Theta(\lg n)$. However, as they themselves highlight in their paper, their proof technique relies on the fact that the update *indices* contain $\Omega(\lg n)$ random bits and it is this information which is then used to provide the lower bound. This is in contrast to our streaming setting where both the update and query indices are fixed and the update values contain only a constant number of bits each.

In this paper we introduce a new variant of the information transfer technique which we call the *lop-sided information transfer technique.* This will enable us to give meaningful lower bounds for precisely this setting, that is when $\delta \in O(1)$, $w \in \Omega(\lg n)$ and both the query and update indices are fixed. Our proof technique will rely on being able to show for specific problems that we need only ℓ query answers to infer at least a constant fraction of the information encoded in the previous $\ell \lg \ell$ updates.

We demonstrate our new framework by applying it to a pattern matching problem with address errors known as L_2-rearrangement distance. This measure of distance, which was first studied in SODA 2006 [1, 2], arises in pattern matching problems where errors occur not in the content of the data but in the addresses where the data is stored. Our proof technique is fundamentally combinatorial in nature. We demonstrate an input distribution which has the property that individual bits of the $\Theta(\lg n)$ sized outputs encode individual bits of the stream. In this way we can infer $\Omega(\ell \lg \ell)$ updates from only $O(\ell)$ outputs as we require. We believe our proof technique is also directly applicable to other simpler distance measures such as the Hamming distance. However establishing the key technical lemma (Lemma 5) appears to be out of reach at present.

The cell probe model and previous lower bounds

Our bounds hold in a particularly strong computational model, the *cell-probe model*, introduced originally by Minsky and Papert [11] in a different context and then subsequently by Fredman [7] and Yao [17]. In this model, there is a separation between the computing unit and the memory, which is external and consists of a set of cells of w bits each. The computing unit cannot remember any information between operations. Computation is free and the cost is measured only in the number of cell reads or writes (cell-probes). This general view makes the model very strong, subsuming for instance the popular word-RAM model.

The first techniques known for establishing dynamic data structure lower bounds had historically been based on the chronogram technique of Fredman and Saks [8] which can at best give bounds of $\Omega(\lg n/\lg\lg n)$. In 2004, Pǎtraşcu and Demaine gave us the first $\Omega(\lg n)$ lower bounds for dynamic data structure problems [14]. Their technique is based on information theoretic arguments which also form the basis for the work we present in this paper. Pǎtraşcu and Demaine also presented ideas which allowed them to express more refined lower bounds such as trade-offs between updates and queries of dynamic data structures. For a list of data structure problems and their lower bounds using these and related techniques, see for example [12]. More recently, a further breakthrough was made by Larsen who showed lower bounds of roughly $\Omega((\lg n/\lg\lg n)^2)$ time per operation for dynamic weighted range counting problem and polynomial evaluation [9, 10]. Subsequent application of this new proof technique has also provided the same lower bound for dynamic matrix-vector multiplication [15]. These lower bounds remain the state of the art for any dynamic structure problem to this day. It is particularly relevant that Larsen's lower bound for dynamic weighted range counting problem cannot yet be applied to the unweighted range counting problem due to a very similar limitation in proof technique to the one we address in this paper.

1.1 Our Results

The lop-sided information transfer technique

In the standard formulation of Demaine and Pǎtraşcu's information transfer technique [13], two adjacent time intervals $[t_0, t_1]$ and $[t_1+1, t_2]$ are considered, with equal, power of two length ℓ. To apply this technique in a streaming setting, the core argument one has to make is that for the given problem, knowledge of the outputs during $[t_1+1, t_2]$ is sufficient to infer a constant fraction of the information encoded by updates during $[t_0, t_1]$. As this information about the updates can be inferred *from* the outputs, the algorithm must know this information *to compute* the outputs. In particular this implies that while computing the outputs during $[t_1+1, t_2]$, the algorithm must probe sufficiently many cells written during $[t_0, t_1]$ to uniquely recover this information. We can think of these cell probes as being associated with interval length ℓ and offset t_0 which uniquely defines the two intervals. The final lower bound is obtained by summing the cell probe lower bounds associated with every power-of-two length ℓ and $t_0 = \ell, 2\ell, 3\ell \ldots$. This final step relies crucially on the fundamental property of the information transfer technique that this summation step does not double count cell probes. In particular that a cell probe associated with some t_0, ℓ is not also associated with some other t_0', ℓ'.

As the argument is information theoretic, to obtain a *logarithmic* lower bound via this approach, both the ℓ updates during $[t_0, t_1]$ and the ℓ outputs during $[t_1+1, t_2]$ must contain $\Omega(\ell\lg\ell)$ bits. However in the bit streaming setting, each update contains $O(1)$ bits so the updates in $[t_0, t_1]$ contain only $O(\ell)$ bits in total.

To overcome this we increase the size of the interval $[t_0, t_1]$ to have length $\ell\lg\ell$ so that both intervals contain $\Omega(\ell\lg\ell)$ bits as required. Unfortunately this modification breaks the fundamental property of the information transfer technique that there is no double counting of cell probes. In fact, direct application of our approach causes each cell probe to be counted $\Theta(\lg n)$ times, negating the possibility of a non-trivial lower bound.

To overcome this we place gaps in time between the end of one interval and the start of another and argue carefully both that not too much of the information can be lost in these gaps and that we can still sum the cell probes over a sufficient number of distinct interval lengths without too much double counting. Our hope is that this new technique will lead to a new class of cell probe lower bounds which could not be proved with existing methods.

Online pattern matching with address errors (L_2-rearrangement distance)

We give an explicit distance function for which we can now obtain the first unconditional online cell probe lower bound for symbol size $\delta = 1$. Consider two strings S_1 and S_2 both of length n where S_2 is a permutation of S_1. Now consider the set of permutations Π so that for all $\pi \in \Pi$, $S_1[\pi(0), \ldots, \pi(n-1)] = S_2$. The L_2-rearrangement distance is defined to be $\min_{\pi\in\Pi} \sum_{j=0}^{n-1}(j - \pi(j))^2$ [2]. In other words, the cost of a permutation is the sum of the square of the number of positions each character is moved. The distance is the minimum cost of any permutation. If Π is empty, that is S_2 is in fact not a permutation of S_1, then the L_2-rearrangement distance is defined to be ∞. As an example, the L_2-rearrangement distance between strings 11100 and 10110 is $0+1+1+2^2+0=6$. In the online L_2-rearrangement problem we are given a fixed pattern $F \in \{0,1\}^n$ and the stream arrives one symbol at a time. After each symbol arrives we must output the L_2-rearrangement distance between F and the most recent n-length suffix of the stream. This online version can be solved in $O(\lg^2 n)$ time per arriving symbol in the word-RAM model [6].

Our technique allows us to recover $\Omega(\lg n)$ distinct bits of the stream from each output. This is achieved by constructing F and carefully choosing a highly structured random input distribution for the incoming stream in such a way that the contributions to the output from different regions of the stream have different magnitudes. We can then use the result to extract distinct information about the stream from different parts of each output.

Using this approach we get the following cell probe lower bound:

▶ **Theorem 1** (Online L_2-rearrangement). *In the cell-probe model with $w \in \Omega(\lg n)$ bits per cell, for any randomised algorithm solving the online L_2-rearrangement distance problem on binary inputs there exist instances such that the expected amortised number of probes per arriving value is*

$$\Omega\left(\frac{\lg^2 n}{w \cdot \lg\lg n}\right).$$

2 Lop-sided information transfer

In this section we will formally define our variant of information transfer, which is a particular set of cells probed by the algorithm, and explain how a bound on the size of the information transfer can be used when proving the overall lower bound of Theorems 1. Our lower bound holds for any randomised algorithm on its worst case input. This will be achieved by applying *Yao's minimax principle* [16]. As a result, from this point onwards we consider an arbitrary deterministic algorithm running with some fixed array F on a random input of n stream values over the binary alphabet $\Sigma = \{0,1\}$. The algorithm may depend on F. As is common in the literature we will refer to the choice of F and the distribution of stream values as the *hard distribution.*

We will let $U \in \{0,1\}^n$ denote the *update array* which describes a sequence of n update operations corresponding to values that arrive in the stream. We will usually refer to the t-th update as the *arrival* of the value $U[t]$. Observe that just after arrival t, the values

$U[t+1, n-1]$ are still not known to the algorithm. We will proceed under the assumption that before the 0-th update, $U[0]$ arrives, the stream contains at least n symbols chosen arbitrarily from the support of the stream distribution. All logarithms are in base two.

2.1 Notation – Two intervals and a gap

In order to define the concept of information transfer from one interval of arriving values in the stream to another interval of arriving values, we first define the set L which contains the interval lengths that we will consider,

$$L = \left\{ n^{1/4} \cdot (\lg n)^{2i} \;\middle|\; i \in \left\{0, 1, 2, \ldots, \frac{\lg n}{4 \lg\lg n}\right\} \right\}.$$

To avoid cluttering the presentation with floors and ceilings, we assume throughout that the value of n is such that any division or power nicely yields an integer. Whenever it is impossible to obtain an integer we assume that suitable floors or ceilings are used. In particular, L contains only integers.

In contrast to the original information transfer method, we define *three* intervals $[t_0, t_1]$, $[t_1+1, t_2-1]$ and $[t_2, t_3]$, referred to as the *left interval*, the *gap* and the *right interval*, respectively. These intervals are functions of a length $\ell \in L$ and an offset $t \in [n/2]$. The left interval has length $\ell \lg \ell$, the gap has length $4\ell/\lg n$ and the right interval has length ℓ. Precisely we define the following four values:

$$t_0 = t, \qquad t_1 = t_0 + \ell \lg \ell - 1, \qquad t_2 = t_1 + \frac{4\ell}{\lg n} + 1, \qquad t_3 = t_2 + \ell - 1.$$

Formally the values t_0, t_1, t_2 and t_3 are functions of ℓ and t but for brevity we will often write just t_0 instead of $t_0(\ell, t)$, and so on, whenever the parameters ℓ and t are obvious from context.

We now highlight some useful properties of these intervals which are easily verified. First observe that the intervals are disjoint and that all intervals are contained in $[0, n-1]$ for sufficiently large n. Second, suppose that $\ell' \in L$ is one size larger than $\ell \in L$, that is $\ell' = \ell \cdot (\lg n)^2$. For ℓ' the length of the gap is $4\ell'/\lg n$, which is sufficiently large that it spans the length of the left interval, the right interval and the gap associated with ℓ. This second property will be particularly important in proving that we do not over-count cell probes.

2.2 Information transfer over gaps

Towards the definition of information transfer, we define, for $\ell \in L$ and $t \in [n/2]$, the subarray $U_{\ell,t} = U[t_0, \ldots, t_1]$ to represent the $\ell \lg \ell$ values arriving in the stream during the left interval. We define the subarray $A_{\ell,t}$ to represent the ℓ outputs during the right interval $[t_2, \ldots, t_3]$. Lastly we define $\widetilde{U}_{\ell,t}$ to be the concatenation of $U[0, (t_0 - 1)]$ and $U[(t_1+1), (n-1)]$. That is, $\widetilde{U}_{\ell,t}$ contains all values of U except for those in $U_{\ell,t}$.

For $\ell \in L$ and $t \in [n/2]$ we first define the *information transfer to the gap*, denoted $\mathcal{G}_{\ell,t}$, to be the set of memory cells c such that c is probed during the left interval $[t_0, t_1]$ of arriving values and also probed during the arrivals of the values $U[t_1+1, t_2-1]$ in the gap. Similarly we define the information transfer to the right interval, or simply *the information transfer*, denoted $\mathcal{I}_{\ell,t}$, to be the set of memory cells c such that c is probed during the left interval $[t_0, t_1]$ of arriving symbols and also probed during the arrivals of symbols in the right interval $[t_2, t_3]$ but *not* in the gap. That is, any cell $c \in \mathcal{G}_{\ell,t}$ cannot also be contained in the information transfer $\mathcal{I}_{\ell,t}$.

The cells in the information transfer $\mathcal{I}_{\ell,t}$ may contain information about the values in $U_{\ell,t}$ that the algorithm uses in order to correctly produce the outputs $A_{\ell,t}$. However, since cells that are probed in the gap are not included in the information transfer, the information transfer might not contain all the information about the values in $U_{\ell,t}$ that the algorithm uses while outputting $A_{\ell,t}$. We will see that the gap is small enough that a large fraction of the information about $U_{\ell,t}$ has to be fetched from cells in the information transfer $\mathcal{I}_{\ell,t}$.

Since cells in the information transfer are by definition probed at some point by the algorithm, we can use $\mathcal{I}_{\ell,t}$ to measure, or at least give a lower bound for, the number of cell probes. As a shorthand we let $I_{\ell,t} = |\mathcal{I}_{\ell,t}|$ denote the size of the information transfer $\mathcal{I}_{\ell,t}$. Similarly we let $G_{\ell,t} = |\mathcal{G}_{\ell,t}|$ denote the size of the information transfer to the gap. By adding up the sizes $I_{\ell,t}$ of the information transfers over all $\ell \in L$ and certain values of $t \in [n/2]$, we get a lower bound on the total number of cells probed by the algorithm during the n arriving values in U. The choice of the values t is crucial as we do not want to over-count the number of cell probes. In the next two lemmas we will deal with the potential danger of over-counting.

For a cell $c \in \mathcal{I}_{\ell,t}$, we write *the probe of c with respect to $\mathcal{I}_{\ell,t}$* to refer to the first probe of c during the arrivals in the right interval. These are the probes of the cells in the information transfer that we count.

▶ **Lemma 2.** *For any $\ell \in L$ and $t, t' \in [n/2]$ such that $|t - t'| \geq \ell$, if a cell c is in both $\mathcal{I}_{\ell,t}$ and $\mathcal{I}_{\ell,t'}$ then the probe of c with respect to $\mathcal{I}_{\ell,t}$ and the probe of c with respect to $\mathcal{I}_{\ell',t'}$ are distinct.*

Proof. Since t and t' are at least ℓ apart, the right intervals associated with t and t', respectively, must be disjoint. Hence the probe of c with respect to $\mathcal{I}_{\ell,t}$ and the probe of c with respect $\mathcal{I}_{\ell,t'}$ must be distinct. ◀

From the previous lemma we know that there is no risk of over-counting cell probes of a cell over information transfers $\mathcal{I}_{\ell,t}$ under a fixed value of $\ell \in L$, as long as no two values of t are closer than ℓ. The proof follows directly from the fact that as $|t - t'| \geq \ell$, the corresponding right intervals for t and t' do not overlap. Distinctness then follows directly from the definition of information transfer. In the next lemma we consider information transfers under different values of $\ell \in L$. The proof follows from the property introduced in Section 2.1 that if (wlog.) $\ell' > \ell$, the gap associated with ℓ' is spans all three intervals associated with ℓ. This implies that either the right intervals for t and t' do not overlap or the left intervals do not overlap. In either case, once again, distinctness follows directly from the definition of information transfer.

▶ **Lemma 3.** *For any $\ell, \ell' \in L$ such that $\ell \neq \ell'$, and any $t, t' \in [n/2]$, if a cell c is in both $\mathcal{I}_{\ell,t}$ and $\mathcal{I}_{\ell',t'}$ then the probe of c with respect to $\mathcal{I}_{\ell,t}$ and the probe of c with respect to $\mathcal{I}_{\ell',t'}$ must be distinct.*

Proof. Let p be the probe of c with respect to $\mathcal{I}_{\ell,t}$, and let p' be the probe of c with respect $\mathcal{I}_{\ell',t'}$. We will show that $p \neq p'$. Suppose without loss of generality that $\ell < \ell'$. From the properties of the intervals that were given in the previous section we know that the length of the gap associated with ℓ' is larger than the sum of lengths of the left interval, the gap and the right interval associated with ℓ.

Suppose for contradiction that $p = p'$. By definition of $\mathcal{I}_{\ell,t}$, the cell c is probed also in the left interval associated with ℓ. Let p_{first} denote any such cell probe. Because the gap associated with ℓ' is so large, p_{first} must take place either in the right interval or the gap associated with ℓ'. If p_{first} is in the gap, then c cannot be in $\mathcal{I}_{\ell',t'}$. If p_{first} is in the right interval then p' cannot equal p. ◀

In order to give a lower bound for the total number of cell probes performed by the algorithm over the n arrivals in U we will define, for each $\ell \in L$, a set $T_\ell \subseteq [n/2]$ of arrivals, such that for any distinct $t, t' \in T_\ell$, $|t - t'| \geq \ell$. It then follows from Lemmas 2 and 3 that

$$\sum_{\ell \in L} \sum_{t \in T_\ell} I_{\ell,t}$$

is a lower bound on the number of cell probes. Our goal is to give a lower bound for the expected value of this double-sum. The exact definition of T_ℓ will be given in Section 3.3 once we have introduced relevant notation.

3 Proving the lower bound

In this section we give the overall proof for the lower bound of Theorem 1. Let $\ell \in L$ and let $t \in [n/2]$. Suppose that $\widetilde{U}_{\ell,t}$ is fixed but the values in $U_{\ell,t}$ are drawn at random in accordance with the distribution for U, conditioned on the fixed value of $\widetilde{U}_{\ell,t}$. This induces a distribution for the outputs $A_{\ell,t}$. We want to show that if the entropy of $A_{\ell,t}$ is large, conditioned on the fixed $\widetilde{U}_{\ell,t}$, then the information transfer $\mathcal{I}_{\ell,t}$ is large, since only the variation in the inputs $U_{\ell,t}$ can alter the outputs $A_{\ell,t}$. We will soon make this claim more precise.

3.1 Upper bound on entropy

We write $H(A_{\ell,t} \mid \widetilde{U}_{\ell,t} = \widetilde{u}_{\ell,t})$ to denote the entropy of $A_{\ell,t}$ conditioned on fixed $\widetilde{U}_{\ell,t}$. Towards showing that high conditional entropy $H(A_{\ell,t} \mid \widetilde{U}_{\ell,t} = \widetilde{u}_{\ell,t})$ implies large information transfer we use the information transfer $\mathcal{I}_{\ell,t}$ and the information transfer to the gap, $\mathcal{G}_{\ell,t}$, to describe an encoding of the outputs $A_{\ell,t}$. The following lemma gives a direct relationship between $I_{\ell,t} + G_{\ell,t}$ and the entropy which is applicable to both of our online problems. A marginally simpler version of the lemma, stated with different notation, was first given in [14] under the absence of gaps.

▶ **Lemma 4.** *Under the assumption that the address of any cell can be specified in w bits, for any $\ell \in L$ and $t \in [n/2]$, the entropy $H(A_{\ell,t} \mid \widetilde{U}_{\ell,t} = \widetilde{u}_{\ell,t}) \leq 2w + 2w \cdot \mathbb{E}[I_{\ell,t} + G_{\ell,t} \mid \widetilde{U}_{\ell,t} = \widetilde{u}_{\ell,t}]$.*

Proof. The expected length of any encoding of $A_{\ell,t}$ under fixed $\widetilde{U}_{\ell,t}$ is an upper bound on the conditional entropy of $A_{\ell,t}$. We use the information transfer $\mathcal{I}_{\ell,t}$ and the information transfer to the gap, $\mathcal{G}_{\ell,t}$, to define an encoding of $A_{\ell,t}$ in the following way. For every cell $c \in \mathcal{I}_{\ell,t} \cup \mathcal{G}_{\ell,t}$ we store the address of c, which takes at most w bits under the assumption that a cell can hold the address of any cell in memory. We also store the contents of c that it holds at the very end of the left interval, just before the beginning of the gap. The contents of c is specified with w bits. In total this requires $2w \cdot (I_{\ell,t} + G_{\ell,t})$ bits.

We will use the algorithm, which is fixed, and the fixed values $\widetilde{u}_{\ell,t}$ of $\widetilde{U}_{\ell,t}$ as part of the decoder to obtain $A_{\ell,t}$ from the encoding. Since the encoding is of variable length we also store the size $I_{\ell,t}$ of the information transfer and the size $G_{\ell,t}$ of the information transfer to the gap. This requires at most $2w$ additional bits.

In order to prove that the described encoding of $A_{\ell,t}$ is valid we now describe how to decode it. First we simulate the algorithm on the fixed input $\widetilde{U}_{\ell,t}$ from the first arrival $U[0]$ until just before the left interval when the first value in $U_{\ell,t}$ arrives. We then skip over all inputs in $U_{\ell,t}$ and resume simulating the algorithm from the beginning of the gap, that is when the value $U[t_1+1]$ arrives. We simulate the algorithm over the arrivals in the gap and the right interval until all values in $A_{\ell,t}$ have been outputted. For every cell being read, we check if it is contained in either the information transfer $\mathcal{I}_{\ell,t}$ or the information transfer to the gap $\mathcal{G}_{\ell,t}$ by looking up its address in the encoding. If the address is found then the contents of the cell is fetched from the encoding. If not, its contents is available from simulating the algorithm on the fixed inputs $\widetilde{U}_{\ell,t}$. ◀

3.2 Lower bound on entropy

Lemma 4 above provides a direct way to obtain a lower bound on the expected value of $I_{\ell,t}+G_{\ell,t}$ if given a lower bound on the conditional entropy $H(A_{\ell,t} \mid \widetilde{U}_{\ell,t}=\widetilde{u}_{\ell,t})$. In Lemma 5 we provide such an entropy lower bound for L_2-rearrangement distance. The proof is deferred to Section 4.

▶ **Lemma 5.** *For the L_2-rearrangement distance problem there exists a real constant $\kappa>0$ and, for any n, a fixed array $F\in\{0,1\}^n$ such that for all $\ell\in L$ and all $t\in[n/2]$ such that $t \bmod 4 = 0$, when U is chosen uniformly at random from $\{0101,1010\}^{\frac{n}{4}}$ then,*

$$H(A_{\ell,t} \mid \widetilde{U}_{\ell,t}=\widetilde{u}_{\ell,t}) \geq \kappa\cdot\ell\cdot\lg n, \text{ for any fixed } \widetilde{u}_{\ell,t}.$$

Before we proceed with the lower bound on the information transfer we make a short remark on the bounds that this lemmas gives. Observe that the maximum conditional entropy of $A_{\ell,t}$ is bounded by the entropy of $U_{\ell,t}$, which is $O(\ell\lg\ell)$ since the length of the left interval is $\ell\lg\ell$. Recall also that the values in L range from $n^{1/4}$ to $n^{3/4}$. Thus, for a constant κ, the entropy lower bound is tight up to a multiplicative constant factor.

3.3 A lower bound on the information transfer and quick gaps

In this section we prove our main lower bound results. We assume that κ is the constant and F is the fixed array of Lemma 5, and that U is chosen uniformly at random from $\{0101,1010\}^{\frac{n}{4}}$.

By combining the upper and lower bounds on the conditional entropy from Lemmas 4 and 5 we have that there is a hard distribution and a real constant $\kappa>0$ such that,

$$\mathbb{E}[I_{\ell,t}+G_{\ell,t} \mid \widetilde{U}_{\ell,t}=\widetilde{u}_{\ell,t}] \geq \frac{\kappa\cdot\ell\cdot\lg n}{2w}-1 \text{ for any } \widetilde{u}_{\ell,t}.$$

We may remove the conditioning by taking expectation over $\widetilde{U}_{\ell,t}$ under random U. Thus,

$$\mathbb{E}[I_{\ell,t}+G_{\ell,t}] \geq \frac{\kappa\cdot\ell\cdot\lg n}{2w}-1, \text{ or equivalently,}$$

$$\mathbb{E}[I_{\ell,t}] \geq \frac{\kappa\cdot\ell\cdot\lg n}{2w}-1-\mathbb{E}[G_{\ell,t}]. \tag{1}$$

Recall that our goal is to give a lower bound for

$$\mathbb{E}\left[\sum_{\ell\in L}\sum_{t\in T_\ell} I_{\ell,t}\right]=\sum_{\ell\in L}\sum_{t\in T_\ell}\mathbb{E}\left[I_{\ell,t}\right], \text{ where } T_\ell \text{ contains suitable values of } t.$$

Using inequality (1) would immediately provide such a lower bound, however, there is an imminent risk that the $\mathbb{E}[G_{\ell,t}]$ terms could devalue such a bound into something trivially small. Now, for this to happen, the algorithm must perform sufficiently many cell probes in the gap. Since the length of the gap is considerably shorter than the right interval, a cap on the worst-case number of cell probes per arriving value would certainly ensure that $\mathbb{E}[G_{\ell,t}]$ stays small, but as we want a stronger amortised lower bound we need something more refined. The answer lies in how we define T_ℓ. We discuss this next.

For $\ell \in L$ and $f \in [\ell]$ we first define $T_{\ell,f} = \left\{ f + i\ell \;\middle|\; i \in \{0,1,2,\dots\} \text{ and } f + i\ell \leq \frac{n}{2} \right\}$ to be the set of arrivals. The values in $T_{\ell,f}$ are evenly spread out, distance ℓ apart, starting at f. We may think of f as the offset of the sequence of values in $T_{\ell,f}$. The largest value in the set is no more than $n/2$. We will define the set T_ℓ to equal a subset of one of the sets $T_{\ell,f}$ for some f. More precisely, we will show that there must exist an offset f such that at least half of the values $t \in T_{\ell,f}$ have the property that the time spent in the gap associated with ℓ and t is small enough to ensure that the information transfer to the gap is small. We begin with some definitions.

▶ **Definition 6** (Quick gaps and sets). For any $\ell \in L$ and $t \in [n/2]$ we say that the gap associated with ℓ and t is *quick* if the expected number of cell probes during the arrivals in the gap is no more than $\kappa\ell \lg n/(4w)$, where κ is the constant from Lemma 5. Further, for any $f \in [\ell]$ we say that the set $T_{\ell,f}$ is *quick* if, for at least half of all $t \in T_{\ell,f}$, the gap associated with ℓ and t is quick.

The next lemma says that for sufficiently fast algorithms there is always an offset f such that $T_{\ell,f}$ is quick. The proof intuition is that if $T_{\ell,f}$ is not quick for any offset f then the whole algorithm must be slow which gives a contradiction.

▶ **Lemma 7.** *Suppose that the expected total number of cell probes over the n arrivals in U is less than $\kappa n(\lg^2 n)/(32w)$. Then, for any $\ell \in L$, there is an $f \in [\ell]$ such that $T_{\ell,f}$ is quick.*

Proof. In accordance with the lemma, suppose that the expected total number of cell probes over the n arrivals in U is less than $\kappa n(\lg^2 n)/(32w)$. For contradiction, suppose that there is no $f \in [\ell]$ such that $T_{\ell,f}$ is quick. We will show that the expected number of cell probes over the n arrivals must then be at least $\kappa n(\lg^2 n)/(32w)$.

For any $f \in [\ell]$, let $R_f \subseteq [n]$ be the union of all arrivals that belong to a gap associated with ℓ and any $t \in T_{\ell,f}$. Let P_f be the number of cell probes performed by the algorithm over the arrivals in R_f. Thus, for any set $T_{\ell,f}$ that is *not quick* we have by linearity of expectation $\mathbb{E}[P_f] \geq \frac{|T_{\ell,f}|}{2} \cdot \frac{\kappa \cdot \ell \cdot \lg n}{4w} = \frac{n/2}{2\ell} \cdot \frac{\kappa \cdot \ell \cdot \lg n}{4w} = \frac{\kappa \cdot n \cdot \lg n}{8w}$.

Let the set of offsets $\mathcal{F} = \left\{ i \cdot \frac{4\ell}{\lg n} \;\middle|\; i \in \left[\frac{\lg n}{4}\right] \right\} \subseteq [\ell]$. The values in $\mathcal{F}$ are spread out with distance $4\ell/\lg n$, which equals the gap length. Thus, for any two distinct $f, f' \in \mathcal{F}$, the sets R_f and $R_{f'}$ are disjoint. We therefore have that the total running time over all n arrivals in U must be bounded below by $\sum_{f\in\mathcal{F}} P_f$. Under the assumption that no $T_{\ell,f}$ is quick, we have that the expected total running time is at least $\mathbb{E}\left[\sum_{f\in\mathcal{F}} P_f\right] = \sum_{f\in\mathcal{F}} \mathbb{E}[P_f] \geq |\mathcal{F}| \cdot \frac{\kappa \cdot n \cdot \lg n}{8w} = \frac{\lg n}{4} \cdot \frac{\kappa \cdot n \cdot \lg n}{8w} = \frac{\kappa \cdot n \cdot \lg^2 n}{32w}$, which is the contradiction we wanted. Thus, under the assumption that the running time over the n arrivals in U is less than $\kappa n(\lg^2 n)/(32w)$ there must be an $f \in [\ell]$ such that $T_{\ell,f}$ is quick. ◀

We now proceed under the assumption that the expected running time over the n arrivals in U is less than $\kappa n(\lg^2 n)/(32w)$. If this is not the case then we have already established the lower bound of Theorem 1.

Let f be a value in $[\ell]$ such that $T_{\ell,f}$ is a quick set. Such an f exists due to Lemma 7. We now let $T_\ell \subseteq T_{\ell,f}$ be the set of all $t \in T_{\ell,f}$ for which the gap associated with ℓ and t is quick. Hence $|T_\ell| \geq |T_{\ell,f}|/2 = n/(4\ell)$. Since $G_{\ell,t}$ cannot be larger than the number of cell probes in the gap, we have by the definition of a quick gap that for any $t \in T_\ell$,

$$\mathbb{E}\left[G_{\ell,t}\right] \leq \frac{\kappa \cdot \ell \cdot \lg n}{4w}.$$

By combining the inequalities we can finally provide a non-trivial lower bound on the sum of the information transfers:

$$\begin{aligned}
\sum_{\ell \in L, t \in T_\ell} \mathbb{E}\left[I_{\ell,t}\right] &\geq \sum_{\ell \in L, t \in T_\ell} \left(\frac{\kappa \cdot \ell \cdot \lg n}{2w} - 1 - \mathbb{E}[G_{\ell,t}]\right) \\
&\geq \sum_{\ell \in L, t \in T_\ell} \left(\frac{\kappa \cdot \ell \cdot \lg n}{2w} - 1 - \frac{\kappa \cdot \ell \cdot \lg n}{4w}\right) \geq \frac{\kappa \cdot \lg n}{5w} \sum_{\ell \in L, t \in T_\ell} \ell \\
&\geq \frac{\kappa \cdot \lg n}{5w} \sum_{\ell \in L} (|T_\ell| \cdot \ell) \geq \frac{\kappa \cdot \lg n}{5w} \sum_{\ell \in L} \left(\frac{n}{4\ell} \cdot \ell\right) = \frac{\kappa \cdot n \cdot \lg n}{20w} \cdot |L| \\
&\geq \frac{\kappa \cdot n \cdot \lg n}{20w} \cdot \frac{\lg n}{4 \lg \lg n} \in \Theta\left(\frac{n \cdot \lg^2 n}{w \cdot \lg \lg n}\right).
\end{aligned}$$

By Lemmas 2 and 3 this lower bound is also a bound on the expected total number of cell probes performed by the algorithm over the n arrivals in U. The amortised time per arriving value is obtained by dividing the running time by n, concluding the proof of Theorem 1.

4 The hard distribution for L_2-rearrangement

In this section we prove Lemma 5. Recall that U is chosen uniformly at random from $\{0101, 1010\}^{\frac{n}{4}}$. For each $\ell \in L$ there is a subarray of F of length $\ell \lg \ell + \ell$. Each such subarray, which we denote F_ℓ, is at distance $4\ell/\lg n + 1$ from the right-hand end of F, which is one more than the length of the gap associated with ℓ. By the properties discussed in Section 2.1 we know that the length of the gap associated with ℓ' is larger than the length of F_ℓ plus the length of the gap associated with ℓ. Hence there is no overlap between the subarrays F_ℓ and $F_{\ell'}$.

Given any of the subarrays F_ℓ and an array U_ℓ of length $(\ell \lg \ell)$, we write $F_\ell \odot U_\ell$ to denote the $(\ell/4)$-length array that consists of the L_2-rearrangement distances between U_ℓ and every *fourth* $(\ell \lg \ell)$-length substring of F_ℓ. More precisely, for $4i \in [\ell]$, the value of $F_\ell \odot U_\ell[i]$ is the L_2-rearrangement distance between $F_\ell[4i, 4i + \ell \lg \ell - 1]$ and U_ℓ.

The main focus of this section is proving Lemma 8 which can be seen as an analogue of Lemma 5 for a fixed length of ℓ:

▶ **Lemma 8.** *There exists a real constant $\epsilon > 0$ such that for all n and $\ell \in L$ there is a subarray F_ℓ for which the entropy of $F_\ell \odot U_\ell$ is at least $\epsilon \cdot \ell \lg \ell$ when U_ℓ is drawn uniformly at random from $\{0101, 1010\}^{\frac{\ell}{4} \lg \ell}$. F_ℓ contains an equal number of 0s and 1s.*

In order to finish the description of the array F we choose each subarray F_ℓ in accordance with Lemma 8. Any region of F that is not part of any of the subarrays F_ℓ is filled with repeats of '01'. This ensures that these regions contain an equal number of zeros and ones. This concludes the description of the array F.

The proof of Lemma 5 then follows from Lemma 8 by arguing that the outputs in $F_\ell \odot U_\ell$ can be calculated from the outputs in $A_{\ell,t}$ by subtracting the contributions from $F_{\ell'} \odot U_{\ell'}$ for all $\ell' \neq \ell$. As each required value from $U_{\ell'}$ is contained in $\widetilde{U}_{\ell,t}$ which is fixed to equal $\widetilde{u}_{\ell,t}$, we have that $H(A_{\ell,t} \mid \widetilde{U}_{\ell,t} = \widetilde{u}_{\ell,t}) \geq H(F_\ell \odot U_\ell)$ as required. This argument requires that for each output, the globally optimal (lowest cost) permutation is always compatible with the locally optimal permutation of each U_ℓ. In particular we need to rule out the possibility of characters from some U_ℓ being moved to positions in $F_{\ell'}$ for $\ell \neq \ell'$. The proof (and the lower bound in general) relies on a key property of L_2-arrangement (proven in Lemma 3.1 from [1]) which states that under the optimal permutation, the i-th one (resp. zero) in one string is moved to the i-th one (resp. zero) in the other. By controlling how the zeros and ones are distributed in U and F, we can limit how far any character is moved. For brevity the details are left for the full version.

We are now ready to prove Lemma 5, the lower bound on the conditional entropy of $A_{\ell,t}$.

Proof of Lemma 5. Let F be the array described above and let U be drawn uniformly at random from $\{0101, 1010\}^{\frac{n}{4}}$. Let $\ell \in L$ and $t \in [n/2]$. Thus, conditioned on any fixed $\widetilde{U}_{\ell,t}$, the distribution of $U_{\ell,t}$ is uniform on $\{0101, 1010\}^{\frac{\ell}{4}\lg \ell}$.

Recall that $U_{\ell,t}$ arrives in the stream between arrival t_0 and t_1, after which $4\ell/\lg n$ values arrive in the gap. Thus, at the beginning of the right interval, at arrival t_2, $U_{\ell,t}$ is aligned with the $(\ell \lg \ell)$-length suffix of the subarray F_ℓ of F. Over the ℓ arrivals in the right interval, $U_{\ell,t}$ slides along F_ℓ. We now prove that since all values in $\widetilde{U}_{\ell,t}$ are fixed, the outputs $A_{\ell,t}$ uniquely specify $F_\ell \odot U_{\ell,t}$. The analogous property for convolution was immediate. First observe that by construction the prefix of F up to the start of F_ℓ contains an equal number of 0s and 1s. Similarly for F_ℓ itself and the suffix from F_ℓ to the end of F. Once in every four arrivals, the substring of U aligned with F is guaranteed (by construction) to also have an equal number of 0s and 1s. Therefore the L_2-rearrangement distance is finite. It was proven in Lemma 3.1 from [1] that (rephrased in our notation) under the optimal rearrangement permutation, the k-th one (resp. zero) in F is moved to the k-th one (resp. zero) in U. Therefore, every element of U_ℓ is moved to an element in F_ℓ. We can therefore recover any output in $F_\ell \odot U_{\ell,t}$ by taking the corresponding output in $A_{\ell,t}$ and subtracting, the costs of moving the elements that are in U but not in U_ℓ. It is easily verified that as t is divisible by four, the corresponding output in $A_{\ell,t}$ is one of those guaranteed to have an equal number of 0s and 1s. Thus, by Lemma 8, the conditional entropy

$$H(A_{\ell,t} \mid \widetilde{U}_{\ell,t} = \widetilde{u}_{\ell,t}) \geq \epsilon \cdot \ell \cdot \lg \ell, \geq \frac{\epsilon}{4} \cdot \ell \cdot \lg n,$$

since $\ell \geq n^{1/4}$. By setting the constant κ to $\epsilon/4$ we have proved Lemma 5. ◀

4.1 High entropy for fixed ℓ – the proof of Lemma 8

In this section we prove Lemma 8. We begin by explaining the high-level approach which will make one final composition of both F_ℓ and U_ℓ into subarrays. For any $j \geq 0$, let $U_\ell^j = U_\ell[\ell \cdot j, \ell \cdot (j+1) - 1]$ i.e. U_ℓ^j is the j-th consecutive ℓ-length subarray of U_ℓ. The key property that we will need is given in Lemma 9 which intuitively states that given half of the bits in U_ℓ, we can compute the other half with certainty.

▶ **Lemma 9.** *Let U_ℓ be chosen arbitrarily from $\{0101, 1010\}^{\frac{\ell}{4}}$. Given F_ℓ, $F_\ell \odot U_\ell$ and U_ℓ^{2j+1} for all $j \geq 0$, it is possible to uniquely determine U_ℓ^{2j} for all $j \geq 0$.*

We briefly justify why Lemma 8 is in-fact a straight-forward corollary of Lemma 9. If we pick U_ℓ uniformly at random from $\{0101, 1010\}^{\frac{\ell}{4}}$ then by Lemma 9, the conditional entropy, $H(F_\ell \odot U_\ell \mid U_\ell^{2j+1}$ for all $j)$ is $\Omega(\ell \lg \ell)$. This is because we always recover $\Theta(\lg \ell)$ distinct U_ℓ^{2j}, each of which is independent and has entropy $\Omega(\ell)$ bits. It then immediately follows that $H(F_\ell \odot U_\ell) \geq H(F_\ell \odot U_\ell \mid U_\ell^{2j+1}$ for all $j)$ as required. We also require for Lemma 8 that F_ℓ

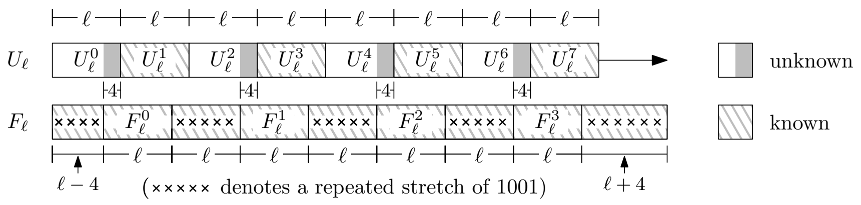

U_ℓ
U_ℓ^0 U_ℓ^1 U_ℓ^2 U_ℓ^3 U_ℓ^4 U_ℓ^5 U_ℓ^6 U_ℓ^7
ℓ
4
F_ℓ
F_ℓ^0 F_ℓ^1 F_ℓ^2 F_ℓ^3
$\ell - 4$
(××××× denotes a repeated stretch of 1001)
$\ell + 4$
unknown
known

Figure 1 We can determine $U_\ell^{2j}[\ell-4,\ell-1]$ if we know F_ℓ, every U_ℓ^{2j+1} and $F_\ell \odot U_\ell$.

contains an equal number ones and zeros. This follows immediately from the description of F_ℓ below.

4.2 The subarray F_ℓ

We now give the description of F_ℓ which requires one final decomposition into subarrays which is also shown in Figure 1 below. For each $j \in [\lfloor(\lg \ell)/2\rfloor]$, F_ℓ contains a subarray F_ℓ^j of length ℓ. Intuitively, each subarray F_ℓ^j will be responsible for recovering U_ℓ^{2j}. These subarrays occur in order in F_ℓ. Before and after each F_ℓ^j there are stretches of repeats of the string 1001. Specifically, before F_ℓ^1 there are $\ell/4-1$ repeats the string 1001. Between each F_ℓ^j and F_ℓ^{j+1} there are $\ell/4$ repeats of the string 1001 and after $F_\ell^{\lfloor(\lg \ell)/2\rfloor-1}$ there are $\ell/4+1$ repeats. These repeats of 1001 are simply for structural padding and as we will see the contribution of these repeated 1001 strings to the L_2-rearrangement distance is independent of U_ℓ. This follows because the cost of permuting 1001 into 1010 or 0101 is always 2.

Finally, the structure of F_ℓ^j is as follows $F_\ell^j = 10^{(2^j+3)}1^{(\ell/4-1)}0^{(\ell/4-(2^j+3))}$. Here 0^z (resp. 1^z) is a string of z zeros (resp. ones). Intuitively, the reason that the stretch of 0s at the start of F_ℓ^j is the exponentially increasing with j is so that the number of positions the second one in F_ℓ^j (immediately after the stretch of 0s) is forced to move is also exponentially increasing with j as demonstrated in Figure 2 below. This is will allow us to recover each U_ℓ^{2j} from a different bit in the outputs. This will claim will be made precise in the proof below.

4.3 Recovering half of the updates – the proof of Lemma 9

We are now in a position to prove Lemma 9. Our main focus will be on first proving that given F_ℓ, U_ℓ^{2j+1} for all j and $F_\ell \odot U_\ell$, we can uniquely determine $U_\ell^{2j}[\ell-4,\ell-1]$ for each $j \geq 0$. That is, for each j whether the last four symbols of U_ℓ^{2j} are 0101 or 1010. This is shown diagrammatically in Figure 1. We will then argue that by a straight-forward repeated application of this argument we can in-fact recover the whole of U_ℓ^{2j} for all $j \geq 0$.

We will begin by making some simplifying observations about $(F_\ell \odot U_\ell)[0]$. Recall that $(F_\ell \odot U_\ell)[0]$ was defined to be the L_2-rearrangement distance between $F_\ell[0,|U_\ell|-1]$ and U_ℓ. The first observation is that the distance is finite because both strings contain an equal number of zeros and ones.

The L_2-rearrangement distance $(F_\ell \odot U_\ell)[0]$ can be expressed as the sum of the *contributions* from moving each $U_\ell[i]$, over all $i \in [m]$. Let the contribution of $U_\ell[i]$, denoted, $\mathrm{CT}(i)$ be the square of the number of positions that $U_\ell[i]$ is moved by under the optimal permutation. We then have that $(F_\ell \odot U_\ell)[0] = \sum_i \mathrm{CT}(j)$. Finally, we let $D^\star$ be the sum of the contributions of the locations in every $U_\ell^{2j}[\ell-4,\ell-1]$, i.e. $D^\star = \sum_j \sum_{k=0}^{3} (\mathrm{CT}(2j\cdot\ell+(\ell-4)+k)$. We will also refer to the contribution of a substring which is defined naturally to be the sum of the contributions of its constituent characters. For example the contribution of the substring U_ℓ^j is equal to $\sum\{\mathrm{CT}(r) \,|\, r \in [\ell\cdot j, \ell\cdot(j+1)-1]\}$.

Our proof will be in two stages. First we will prove in Lemma 10 that we can compute $D^\star$ from F_ℓ, $F_\ell \odot U_\ell$ and U_ℓ^{2j+1} for all $j \geq 0$. Second we will prove that for any $j > 0$, we can determine $U_\ell^{2j}[\ell-4, \ell-1]$ from $D^\star$.

In the proof of Lemma 10 we argue that $D^\star$ can be calculated directly from $(F_\ell \odot U_\ell)[0]$ by subtracting the contributions of U_ℓ^{2j+1} and $U_\ell^{2j}[0, \ell-5]$ for all $j \geq 0$. More specifically, we will prove that the contribution of any U_ℓ^{2j+1} can calculated from U_ℓ^{2j+1} and F_ℓ, which are both known. In particular, the contribution of any U_ℓ^{2j+1} is independent of every unknown U_ℓ^{2j}. Further, we will prove that although U_ℓ^{2j} is unknown, the contribution of $U_\ell^{2j}[0, \ell-5]$, always equals $\ell/2 - 2$, regardless of the choice of U_ℓ.

▶ **Lemma 10.** *$D^\star$ can be computed from F_ℓ, $F_\ell \odot U_\ell$ and U_ℓ^{2j+1} for all $j \geq 0$.*

Proof. In this proof we rely heavily on Lemma 3.1 from [1] which states that under the optimal permutation, the i-th one (resp. zero) in U_ℓ is moved to the i-th one (resp. zero) in $F_\ell[0, |U_\ell|-1]$. For any j, consider, U_ℓ^{2j} and U_ℓ^{2j+1}. The number of ones in U_ℓ^{2j} (resp. U_ℓ^{2j+1}) is fixed, independent of the choice of U_ℓ. In particular there are exactly $\ell/2$ zeros and $\ell/2$ ones. It is easily verified that, by construction, $F_\ell[2j \cdot \ell, (2j+2)\cdot\ell - 1]$ also contains exactly ℓ zeros and ℓ ones. Therefore, the i-th one (resp. zero) in U_ℓ^{2j} is moved to the i-th one (resp. zero) in $F_\ell[2j \cdot \ell, (2j+2)\cdot\ell - 1]$. Similarly, the i-th one (resp. zero) in U_ℓ^{2j} is moved to the $(i+\ell/2)$-th one (resp. zero) in $F_\ell[2j \cdot \ell, (2j+2)\cdot\ell - 1]$

Consider any U_ℓ^{2j+1} which is known. By the above observation, we can therefore determine which position in $F_\ell[2j \cdot \ell, (2j+2)\cdot\ell - 1]$, each character in U_ℓ^{2j+1} is moved to under the optimal permutation. From this we can then directly compute the contribution of each U_ℓ^{2j+1} to $(F_\ell \odot U_\ell)[0]$.

Consider any U_ℓ^{2j} which is unknown. As observed above, the i-th one (resp. zero) in U_ℓ^{2j} is moved to the i-th one (resp. zero) in $F_\ell[2j \cdot \ell, (2j+2)\cdot\ell - 1]$. By construction, we have that $F_\ell[2j \cdot \ell, (2j+1)\cdot\ell - 5]$ consists entirely of repeats of 1001. Further for any i, we have that $U_\ell^{2j}[4i, 4i+3]$ is either 1010 or 0101. Therefore for all $i < \ell/4$ we have that the two ones (resp. zeros) in $U_\ell^{2j}[4i, 4i+3]$ are moved to the two ones (resp. zeros) in $F_\ell[2j \cdot \ell + 4i, 2j \cdot \ell + 4i + 3] = 1001$. The key observation is that regardless of whether $U_\ell^{2j}[4i, 4i+3] = 1010$ or 0101, the contribution of $U_\ell^{2j}[4i, 4i+3]$ is 2. Therefore for any U_ℓ, the contribution of $U_\ell^{2j}[0, \ell-5]$ is always $\ell/2 - 2$.

Finally, the value of $D^\star$ is can be calculated directly from $(F_\ell \odot U_\ell)[0]$ as claimed by subtracting the calculated contributions of U_ℓ^{2j+1} and $U_\ell^{2j}[0, \ell-5]$ for all $j \geq 0$. ◀

In Lemma 12 we will prove that we can compute $U_\ell^{2j}[\ell-4, \ell-1]$ from $D^\star$ (for any sufficiently large j). The intuition behind this is given by Lemma 11 which gives an explicit formula for the contribution of $U_\ell^{2j}[\ell-4, \ell-1]$. Observe that the contribution depends only on whether $U_\ell^{2j}[\ell-4, \ell-1]$ equals 1010 ($v_j = 1$) or 0101 ($v_j = 0$). In the proof we begin by arguing that under the optimal permutation, the two ones (resp. zeros) in $U_\ell^{2j}[\ell-4, \ell-1]$ are moved to the leftmost two ones (resp. zeros) in F_ℓ^j as illustrated in Figure 2. The key observation is that regardless of whether v_j equals 0 or 1, by construction the right one in $U_\ell^{2j}[\ell-4, \ell-1]$ is moved exponentially far (as a function of j). Furthermore, in the $v_j = 1$ case the right one moves one position further than in the $v_j = 0$ case. As the contribution is the square of the number of positions a character moves, this creates a exponentially large change in the contribution. The exact contribution given in the Lemma can be calculated straightforwardly by considering each of the four symbols in $U_\ell^{2j}[\ell-4, \ell-1]$ individually.

▶ **Lemma 11.** *For any j, let $v_j = 1$ if $U_\ell^{2j}[\ell-4, \ell-1] = 1010$ and $v_j = 0$ otherwise. The contribution of $U_\ell^{2j}[\ell-4, \ell-1]$ is exactly $v_j \cdot 2^{j+1} + 2^{2j} + 2$.*

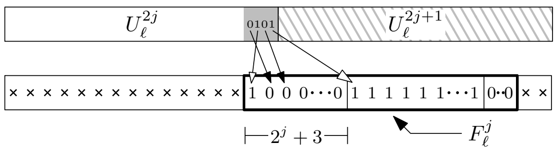


Figure 2 The permutation of the symbols in $U_\ell^{2j}[\ell-4,\ell-1]$ under the optimal permutation. The highlighted region is F_ℓ^j.

We can now prove Lemma 12 which follows almost immediately from Lemma 11.

▶ **Lemma 12.** *For any $j \geq 0$, it is possible to compute $U_\ell^{2j}[\ell-4,\ell-1]$ from $D^\star$.*

Proof. Let $D_2^\star$ equal $D^\star - \sum_j(2^{2j}+2)$ which can be calculated directly from $D^\star$. An alternative and equivalent definition of $D_2^\star$ follows from Lemma 11 and is given by $D_2^\star = \sum_j v_j \cdot 2^{j+1}$. We can therefore compute v_j and hence $U_\ell^{2j}[\ell-4,\ell-1]$ by inspecting the $(j+1)$-th bit in the binary representation of $D_2^\star$. ◀

Recall from Lemma 10 that $D^\star$ can in turn be computed from F_ℓ, $F_\ell \odot U_\ell$ and U_ℓ^{2j+1} for all $j \geq 0$. Therefore as claimed, given F_ℓ, $F_\ell \odot U_\ell$ and U_ℓ^{2j+1} for all $j \geq 0$, we can uniquely determine $U_\ell^{2j}[\ell-4,\ell-1]$ for each $j \geq 0$. Lemma 9 now follows almost immediately by repeat application of this argument as we now set out.

Recovering the rest of $U_{\ell,(2j)}$

So far we have only proven that we can recover $U_\ell^{2j}[\ell-4,\ell-1]$ for all j. The claim that we can in-fact recover the whole of U_ℓ^{2j} follows by repeatedly application of the argument above. Specifically, once we have recovered $U_\ell^{2j}[\ell-4,\ell-1]$ for all j, we can use this additional information (and $(F_\ell \odot U_\ell)[1]$ instead of $(F_\ell \odot U_\ell)[0]$) to recover $U_\ell^{2j}[\ell-8,\ell-5]$ for all j and so on. More formally we proceed by induction on increasing k by observing that using the above argument given F_ℓ, $(F_\ell \odot U_\ell)[k]$, U_ℓ^{2j+1} for all $j \geq 0$ and $U_\ell^{2j+1}[\ell-4k,\ell-1]$ for all $j \geq 0$ we can recover $U_\ell^{2j+1}[\ell-4k-4,\ell-4k-1]$ for all j.

References

1 A. Amir, Y. Aumann, G. Benson, A. Levy, O. Lipsky, E. Porat, S. Skiena, and U. Vishne. Pattern matching with address errors: rearrangement distances. In *SODA'06: Proc. 17th ACM-SIAM Symp. on Discrete Algorithms*, pages 1221–1229. ACM Press, 2006.

2 A. Amir, Y. Aumann, G. Benson, A. Levy, O. Lipsky, E. Porat, S. Skiena, and U. Vishne. Pattern matching with address errors: Rearrangement distances. *Journal of Computer System Sciences*, 75(6):359–370, 2009.

3 R. Clifford and M. Jalsenius. Lower bounds for online integer multiplication and convolution in the cell-probe model. In *ICALP'11: Proc. 38th International Colloquium on Automata, Languages and Programming*, pages 593–604, 2011. `arXiv:1101.0768`.

4 R. Clifford, M. Jalsenius, and B. Sach. Tight cell-probe bounds for online hamming distance computation. In *SODA'13: Proc. 24th ACM-SIAM Symp. on Discrete Algorithms*, pages 664–674, 2013. `arXiv:1207.1885`.

5 R. Clifford, M. Jalsenius, and B. Sach. Cell-probe bounds for online edit distance and other pattern matching problems. In *SODA'15: Proc. 26th ACM-SIAM Symp. on Discrete Algorithms*, 2015. `arXiv:1407.6559`.

6 R. Clifford and B. Sach. Pattern matching in pseudo real-time. *Journal of Discrete Algorithms*, 9(1):67–81, 2011.

7 M. Fredman. Observations on the complexity of generating quasi-Gray codes. *SIAM Journal on Computing*, 7(2):134–146, 1978.

8 M. Fredman and M. Saks. The cell probe complexity of dynamic data structures. In *STOC'89: Proc. 21st Annual ACM Symp. Theory of Computing*, pages 345–354, 1989.

9 K. Green Larsen. The cell probe complexity of dynamic range counting. In *STOC'12: Proc. 44th Annual ACM Symp. Theory of Computing*, pages 85–94, 2012.

10 K. Green Larsen. Higher cell probe lower bounds for evaluating polynomials. In *FOCS'12: Proc. 53rd Annual Symp. Foundations of Computer Science*, pages 293–301, 2012.

11 M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry.* MIT Press, 1969.

12 M. Pătraşcu. *Lower bound techniques for data structures.* PhD thesis, MIT, 2008.

13 M. Pătraşcu and E. D. Demaine. Tight bounds for the partial-sums problem. In *SODA'04: Proc. 15th ACM-SIAM Symp. on Discrete Algorithms*, pages 20–29, 2004.

14 M. Pătraşcu and E. D. Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM Journal on Computing*, 35(4):932–963, 2006.

15 R.Clifford, A. Grønlund, and K. Green Larsen. New unconditional hardness results for dynamic and online problems. In *FOCS'15: Proc. 56th Annual Symp. Foundations of Computer Science*, pages 1089–1107, 2015.

16 A. Yao. Probabilistic computations: Toward a unified measure of complexity. In *FOCS'77: Proc. 18th Annual Symp. Foundations of Computer Science*, pages 222–227, 1977.

17 A. Yao. Should tables be sorted? *Journal of the ACM*, 28(3):615–628, 1981.

Stochastic Streams: Sample Complexity vs. Space Complexity

Michael Crouch[1], Andrew McGregor*[2], Gregory Valiant[3], and David P. Woodruff[4]

1 Bell Labs, Dublin, Ireland
2 University of Massachusetts, Amherst, MA, USA
3 Stanford University, Stanford, CA, USA
4 IBM Research Almaden, San Jose, CA, USA

Abstract

We address the trade-off between the computational resources needed to process a large data set and the number of samples available from the data set. Specifically, we consider the following abstraction: we receive a potentially infinite stream of IID samples from some unknown distribution D, and are tasked with computing some function $f(D)$. If the stream is observed for time t, how much memory, s, is required to estimate $f(D)$? We refer to t as the sample complexity and s as the space complexity. The main focus of this paper is investigating the trade-offs between the space and sample complexity. We study these trade-offs for several canonical problems studied in the data stream model: estimating the collision probability, i.e., the second moment of a distribution, deciding if a graph is connected, and approximating the dimension of an unknown subspace. Our results are based on techniques for simulating different classical sampling procedures in this model, emulating random walks given a sequence of IID samples, as well as leveraging a characterization between communication bounded protocols and statistical query algorithms.

1998 ACM Subject Classification F.2.3 Tradeoffs between Complexity Measures

Keywords and phrases data streams, sample complexity, frequency moments, graph connectivity, subspace approximation

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.32

1 Introduction

Big data systems must process data at multiple levels of a hierarchy, starting at local per-process or per-container monitoring and extending up to aggregate statistics for entire databases or data centers. When designing monitoring or analytics for these systems, some of the first decisions which must be made are which levels of the hierarchy should hold the analytics, and consequently what computational resources are available for processing data without introducing significant overhead. In this paper, we initiate the theoretical investigation of one of the fundamental trade-offs involved in architecting these systems: the amount of memory needed to process incoming data and the total amount of data the system must collect.

In algorithmic terms, we consider the following abstraction: we receive a stream of IID samples from some unknown distribution D, and are tasked with estimating some function $f(D)$ of the distribution. Two natural questions about this task have been studied extensively:

* This work was partially supported by NSF Awards CCF-0953754, CCF-1320719, and a Google Research Award.

1. The statistics question is how to bound the *sample complexity*: how many samples are required to estimate $f(D)$ to some prescribed accuracy with high probability?
2. The data stream question is how to bound the *space complexity*: how much memory is required to compute or approximate the estimator for $f(D)$?

In real systems, of course, both questions are important. Requiring more samples adds processing overhead to the system, and increases the time necessary for the system to detect and react to changes in the underlying distribution. Requiring additional memory increases the overhead on the system, and may make some analytics impractical, or require them to be relocated to separate machines or systems.

Despite the clear importance of this trade-off, our work is the first to our knowledge which explicitly examines the trade-off between the number of samples which must be taken from a stream and the amount of memory necessary to process these samples. We begin this investigation with a study of three canonical problems from the data streaming literature: estimating the collision probability, also known as the second frequency moment [2], undirected connectivity [13, 14, 35], and rank estimation [10, 25, 6]. For all three problems, we find trade-offs between the sample and space requirements.

Sufficient Statistics and Data Streams. The goal of space-efficiency in statistical estimation is not new. In the study of *sufficient statistics* [15] the goal is to prove that it suffices to maintain a small number of statistics about the input when estimating certain parameters of the source distribution D. For example, to estimate μ if $D \sim N(\mu, 1)$, it is sufficient to maintain the sum and count of the samples; other information can be discarded. However, for non-parametric problems sufficient statistics typically do not exist. Our work could be seen as “approximate sufficient statistics” – statistics about the stream of samples that can be computed online that will suffice to estimate the relevant properties of the input with high probability.

In contrast to the majority of data stream research, in our setting we do not need to consider adversarially-ordered streams since the assumption is that the input stream is generated by a stochastic process. There is a growing body of work on randomly-ordered streams [20, 33, 29, 8, 27, 17, 7]. Some work has also explicitly considered streams of IID samples [9, 41, 19, 30]. There has also been work on hypothesis testing given limited space [23, 21].

Subsampling vs. Supersampling. Other related work includes a paper by an overlapping set of authors [26] (see also [37]) that considered the problem of processing data streams whose arrival rate was so high that it was not possible to observe every element of the stream. Consequently, it was presumed that the stream was first *subsampled* and then properties of the original stream had to be deduced from the samples. In contrast, in this work we essentially consider oversampling, or *supersampling* the data set. The motivation is two-fold. First, in many applications there is an abundance of redundant data, e.g., from sensors that continually monitor a static environment, and it makes sense to find a way to capitalize on this data. Second, it may be preferable from a computational point of view, to run a fast light-weight algorithm on a lot of data rather than a computationally expensive algorithm on a small amount of data.

A number of early works consider learning in online settings, and we only mention a few here. There is a line of work on hypothesis testing and statistical estimation with finite memory, see, e.g., [11, 24], but the lower bounds in such models come from finite precision constraints, and do not seem directly related to the model or problems that we study. Other work [31] considers lower bounds for algorithms given data in a specific form, i.e., in so-called oracle models, and the limitations of such assumptions on the model have been exploited to design faster algorithms in some cases [32].

More recently, [38, 39] consider the question of learning with memory or communication constraints in the setting in which the algorithm has access to a stream of randomly drawn labeled examples. Their algorithmic "memory bounded" model corresponds to our model. We note that these works study a different set of problems than the ones from the data stream literature that we focus on, reinforcing the naturalness of the model. The lower bounds in these works use a more general communication model than our streaming machines: given a communication bound of b bits per example, an algorithm must compress the ith example according to an arbitrary function that may depend on the compressed versions of the first $i-1$ examples; the algorithm may then compute an arbitrary function of the compressed examples. The authors show tight connections between communication bounded protocols, and statistical query algorithms, and we leverage their characterization to show a communication lower bound, and ultimately space lower bound in Section 4.

Our Results. We study the trade-off between sample and space complexity for canonical problems in the data stream literature such as estimating the collision probability or second frequency moment [2], undirected connectivity [13, 14, 35], and rank estimation [10, 25, 6]; see also the references therein. We obtain the following tradeoffs:

1. *Collision Probability.* Suppose $D = D_p$ is a distribution $(p_1, \ldots, p_n)$ over $[n]$. Then, to estimate $F_2(D) = \sum_i p_i^2$ up to a factor $(1+\epsilon)$ with probability $1-\delta$, for any[1] $t \geq t^* = \tilde{\Omega}_{\epsilon,\delta}(n^{1/2})$ it is sufficient for

   $$s \cdot t = \tilde{O}_{\epsilon,\delta}(n)\,.$$

   Moreover, we show a lower bound that $s \cdot t^{1.5} = \tilde{\Omega}(n^{5/4})$ space is necessary to return a constant factor approximation with constant probability, which is tight when s and t are $\tilde{\Theta}_{\epsilon,\delta}(\sqrt{n})$. We also point out an error in existing work, which if fixable, would result in the tight $s \cdot t = \tilde{\Omega}(n)$ tradeoff. We present these results in Section 2.
2. *Graph Connectivity.* Suppose $D = D_G$ is the uniform distribution of the set of edges of an undirected, unweighted graph G. We show in Section 3 that for $t \geq t^* = \Omega(|E| \log |E|)$, it suffices for

   $$s^2 \cdot t = \tilde{O}(|E| \cdot |V|^2)\,.$$

3. *Subspace Approximation.* In Section 4 we consider the problem of determining whether a set of samples from $\{0,1\}^n$ is being drawn from the entire space or from some rank $n/2$ subspace. We first note an $s = O(\log n)$-bit space streaming algorithm which achieves sample complexity $t = O(2^n)$. More interestingly, we then show that this is near-optimal by showing that any streaming algorithm using $s \leq n/8$ space requires

   $$2^s \cdot t = \Omega(2^{n/8})\ .$$

   So even if one allows say, $s = n/16$ bits of space, one still needs $2^{\Omega(n)}$ examples.

We believe one of the main contributions of our work is the new framework for analyzing tradeoffs between the sample and space complexity of streaming algorithms. We have chosen

[1] $\tilde{O}_{\epsilon,\delta}(f(n))$ and $\tilde{\Omega}_{\epsilon,\delta}$ omit $\text{poly}(\log(n/\delta), 1/\epsilon)$ terms.

this set of problems because it is representative of well-studied sub-areas in the data stream literature such as estimating statistical quantities such as frequency moments, graph problems in the semi-streaming model, and numerical linear algebra problems.

2 Collision Probability

The collision probability is a fundamental quantity quantifying how far a distribution is from the uniform distribution, the latter having the smallest possible collision probability. Estimating the empirical collision probability is well-studied for worst case data streams of a given length, dating back to the seminal work of Alon, Matias, and Szegedy [2]. It has applications to estimating self-join sizes in databases, and is also known as the *repeat rate* or *Gini's Index of homogeneity*, which is used to compute the *surprise index* [16].

Here we consider a stream of independent samples $\langle a_1, a_2, a_3, \ldots \rangle$ from an unknown discrete distribution p over $[n]$. Let $p_j = \mathbb{P}[a_i = j]$ and let $F_2 = \sum_j p_j^2$.

2.1 The Upper Bound

▶ **Theorem 1.** *For $t = \tilde{\Omega}_{\epsilon,\delta}(n^{1/2})$, estimating F_2 up to a factor $(1 \pm \epsilon)$ given t samples with probability at least $1-\delta$ is possible in $\tilde{O}_{\epsilon,\delta}(1 + n/t)$ bits of space.*

Proof. We can assume δ is a constant, since the algorithm generalizes to smaller δ simply by increasing the sample complexity by a factor of $\log(1/\delta)$, running our algorithm $\log(1/\delta)$ times in sequence on each of $\log(1/\delta)$ independent groups of samples, and taking the median. Note $t = \Omega(n^{1/2})$ was shown necessary by Bar-Yossef [4]; with fewer samples estimating F_2 is information-theoretically impossible.

Our algorithm partitions the input stream into t/w groups of w consecutive samples. We now analyze one specific group of w samples. Suppose the samples are $a_1, \ldots, a_w$. For each pair $i \neq j \in \{1, 2, \ldots, w\}$, let $X_{i,j}$ be an indicator random variable which is 1 iff $a_i = a_j$. Let $X = \frac{1}{\binom{w}{2}} \sum_{i \neq j} X_{i,j}$. Then $\mathbf{E}[X]$ is equal to $\mathbf{E}[X_{i,j}]$ for an arbitrary $i \neq j$, and the latter is precisely $F_2 = \sum_i p_i^2$. We also have, assuming $i \neq j$ and $i' \neq j'$ in the following:

$$\begin{aligned}
\binom{w}{2}^2 \mathbf{Var}[X] &= \sum_{(i,j)=(i',j')} \mathbf{E}[X_{i,j}X_{i',j'}] - \mathbf{E}[X_{i,j}]\mathbf{E}[X_{i',j'}] \\
&\quad + \sum_{|\{i,j,i',j'\}|=3} \mathbf{E}[X_{i,j}X_{i',j'}] - \mathbf{E}[X_{i,j}]\mathbf{E}[X_{i',j'}] \\
&\quad + \sum_{|\{i,j,i',j'\}|=4} \mathbf{E}[X_{i,j}X_{i',j'}] - \mathbf{E}[X_{i,j}]\mathbf{E}[X_{i',j'}] \\
&= \sum_{(i,j)} \mathbf{E}[X_{i,j}^2] - \mathbf{E}^2[X_{i,j}] + \sum_{|\{i,j,i',j'\}|=3} \mathbf{E}[X_{i,j}X_{i',j'}] - \mathbf{E}[X_{i,j}]\mathbf{E}[X_{i',j'}] \\
&\quad + \sum_{|\{i,j,i',j'\}|=4} \mathbf{E}[X_{i,j}]\mathbf{E}[X_{i',j'}] - \mathbf{E}[X_{i,j}]\mathbf{E}[X_{i',j'}] \\
&\leq \sum_{(i,j)} \mathbf{E}[X_{i,j}^2] + \sum_{|\{i,j,i',j'\}|=3} \mathbf{E}[X_{i,j}X_{i',j'}] - \mathbf{E}[X_{i,j}]\mathbf{E}[X_{i',j'}],
\end{aligned}$$

where the first equality follows by expanding the variance into covariances, the second equality uses independence of disjoint indices, and the inequality cancels the terms for which

$|\{i,j,i',j'\}| = 4$ and drops non-positive terms. Therefore,

$$\begin{aligned}
\mathbf{Var}[X] &\le \frac{1}{\binom{w}{2}^2}\left(\sum_{(i,j)} \mathbf{E}[X_{i,j}^2] + \sum_{|\{i,j,i',j'\}|=3} \mathbf{E}[X_{i,j}X_{i',j'}]\right)\\
&= \frac{\mathbf{E}[X]}{\binom{w}{2}} + \Theta\left(\frac{1}{w^4}\right) \sum_{\text{distinct } i,j,k} \mathbf{E}[X_{i,j}X_{j,k}]\\
&= \frac{F_2}{\binom{w}{2}} + \Theta\left(\frac{1}{w^4}\right) \sum_{\text{distinct } i,j,k} \Pr[X_{i,j}=1 \mid X_{j,k}=1]\cdot \Pr[X_{j,k}=1]\\
&= \frac{F_2}{\binom{w}{2}} + \Theta\left(\frac{1}{w^4}\right) \sum_{\text{distinct } i,j,k} \sum_{\ell} \Pr[X_{i,j}=1 \mid a_j=\ell]\cdot \Pr[a_j=\ell \mid X_{j,k}=1]\cdot F_2\\
&= \frac{F_2}{\binom{w}{2}} + \Theta\left(\frac{1}{w^4}\right) \sum_{\text{distinct } i,j,k} \sum_{\ell} p_\ell \cdot \frac{p_\ell^2}{F_2}\cdot F_2\\
&= \frac{F_2}{\binom{w}{2}} + \Theta\left(\frac{F_3}{w}\right),
\end{aligned}$$

where we have used the law of total probability and the definitions, and here $F_3 = \sum_j p_j^3$.

In the stream, we will use $O(w \log n)$ bits of space to compute X as above. Then, since we have partitioned the samples into $q = t/w$ groups (which we can assume is an integer w.l.o.g.), we will compute an independent estimate X for each group, and take their average, obtaining a random variable Y. Thus Y can be computed in $O(w \log n)$ bits of space. Then $\mathbf{E}[Y] = \mathbf{E}[X] = F_2$ and $\mathbf{Var}[Y] = \mathbf{Var}[X]/q$. By Chebyshev's inequality,

$$\begin{aligned}
\Pr[|Y - F_2| \ge \epsilon F_2] &\le \frac{\mathbf{Var}[X]}{q\epsilon^2 F_2^2} \le \frac{1}{\binom{w}{2}F_2 q\epsilon^2} + O\left(\frac{F_3}{wq\epsilon^2 F_2}\right)\\
&= O\left(\frac{n}{w^2 q\epsilon^2}\right) + O\left(\frac{\sqrt{n}}{wq\epsilon^2}\right),
\end{aligned}$$

where the final inequality uses that $F_2 \ge 1/n$ in the first expression, while the second expression uses Hölder's inequality to show that $F_3 \le \sqrt{n}F_2$.

Plugging in $q = t/w$, this probability is $O(n/(tw\epsilon^2) + \sqrt{n}/(t\epsilon^2))$. Thus, for $t = \Omega(n^{1/2}\epsilon^{-2})$ samples, we can set $w = O(n/t)$ and have this probability be smaller than an arbirarily small constant. Note that as a sanity check, we need $t = \Omega(n^{1/2})$ to ensure $w \le t$, as required by the definition of these variables. This completes the proof. ◀

2.2 The Lower Bound

Our lower bound relies on a result by Andoni et al. [3] for the random order data stream model, which is different than the model we consider in this paper. We note that an improvement to the work of Andoni et al. [3] was claimed in [18]; however, after communication with the authors, the proof given in [18] seems to have a bug and is currently not known to be fixable. If the result in [18] is fixable, then we obtain an optimal tradeoff of $s \cdot t = \tilde{\Omega}(n)$; otherwise we obtain the slightly weaker tradeoff presented here.

The work of [3] concerns distinguishing between the following two cases with constant probability, and shows that $\Omega(t/r^{2.5})$ space is required:

1. (Case 1) A sequence of t samples from a distribution p^{no} that is uniform on some subset $S \subseteq [t]$ of size $\Theta(t)$.
2. (Case 2) A sequence of t samples from a distribution p^{yes} such that $p_i^{\text{yes}} = r/t$ for some $i \in [t]$ and uniform on some subset $T \subseteq [t] \setminus \{i\}$.

By combining this result with a hashing technique we establish the following result.

▶ **Theorem 2.** *Any constant factor approximation of $F_2(D)$ with constant probability given a sequence of t IID samples on $[n]$ requires $\Omega(n^{5/4}/((\log^{2.5} n) \cdot t^{1.5}))$ bits of space.*

Proof. Let $h : [t] \to [n]$ be a fully-random hash function and consider the problem of distinguishing p^{no} and p^{yes} where we set $r = c \log n \cdot t \cdot n^{-1/2}$ for some constant $c > 0$. By applying h on each distribution (i.e., applying h to each observed sample) we generate two new distributions q^{no} and q^{yes} over $[n]$ where:

$$q_i^{\text{no}} = \sum_{j:h(j)=i} p_j^{\text{no}} \quad \text{and} \quad q_i^{\text{yes}} = \sum_{j:h(j)=i} p_j^{\text{yes}} .$$

Note that with high probability $\max_i q_i^{\text{no}} = O(\log n \cdot 1/n)$ and hence $F_2(q^{\text{no}}) \leq n \cdot O((\log n \cdot 1/n)^2) = O(\log^2 n \cdot n^{-1/2})$. However, $\max_i q_i^{\text{yes}} \geq r/t$ and so $F_2(q^{\text{yes}}) \geq r^2/t^2 = c^2 \cdot \log^2 n \cdot n^{1/2}$. Hence, for a sufficiently large value of the constant $c > 0$ we can ensure that any constant approximation of F_2 distinguishes between q^{yes} and q^{no} and hence, also distinguishes between p^{yes} and p^{no}. However, by the result of Andoni et al. [3] we know that this requires $\Omega(t/r^{2.5}) = \Omega(n^{2.5(1/2)}/((\log^{2.5} n) \cdot t^{1.5}))$ bits of space. ◀

3 Connectivity

In this section, we consider a graph model where our input stream is a sequence of t random samples drawn (with replacement) from the graph's entire edge set E. This model is appropriate for random processes where the "graph" involved is defined only implicitly, and is not available for querying. For example, on a large social network, we can imagine a graph where nodes represent users and weighted edges represent user frequencies of interaction. We may not have enough space or processing power to analyze the history of interactions between users directly, or to provide random access to this history; according to [40] an estimated 50 billion text messages per day were sent in 2014. However, the stream of ongoing interactions approximates random samples from the weighted edge distribution.

We begin the study of this model by first examining unweighted graphs, and by examining the canonical problem of determining connectivity for such graphs. Our algorithm uses the sampled edges to simulate classical random walks on the graph. In Section 3.1, we discuss how to simulate and tightly analyze random walks in our model. The main technical difficulty is ensuring independence when simulating multiple random walks in parallel. Then, in Section 3.2, we adapt a connectivity algorithm of Feige [14] to achieve the required space/sample trade-off. Since our algorithm provides a general technique for space/sample trade-offs in simulating random walks, we believe it will be a useful starting point for examining additional problems in this model.

For notational convenience, denote the graph by $G = (V, E)$, the number of nodes by $n = |V|$, and the number of edges by $m = |E|$.

3.1 Technique: Emulating Classical Random Walks

Consider the following basic algorithm: given a node v, we sample edges until we receive an edge $\{v, u\}$ for some u. At this point, we move to node u, and repeat. We refer to this method as a *sampling walk*. Note that the expected time to leave v is $m/d(v)$ samples[2] where

[2] This is because the number of samples is geometrically distributed with parameter $d(v)/m$.

$d(v)$ is the degree of a node v, and so a single step of a classical random walk may require $\Omega(m)$ samples if v has low degree.

An Inefficient Connectivity Algorithm. This basic algorithm already leads to a $O(\log n)$ space algorithm which uses $O(m^2n^2)$ samples in expectation. This follows by starting a sampling walk at node 1 and emulating a classical walk until it traverses nodes $2, 3, \ldots, n$ in order. The expected length of this walk is $O(mn^2)$ because the cover time of G, i.e., the expected length of walk until it visits all nodes (see e.g., [28]), is $O(mn)$ and there are $n-1$ segments in the traversal. Hence, emulating the random walk takes $O(m^2n^2)$ samples in expectation. The space use is $O(\log n)$ bits because the algorithm just needs to remember the current node and the furthest node that has been reached in the sequence. In what follows, we will improve upon the number of samples required and generalize to algorithms that use more space.

The Loopy Graph and an Improved Analysis. The first improvement comes via a better analysis. At a node v with $d(v)$ neighbors, there are $d(v)$ possible samples which would result in a move, and $m - d(v)$ samples which would not. We can thus view our sampling-based walk on G as a classical random walk on a new graph H formed by adding $m - d(v)$ self-loops to each vertex v in G. We call H the "loopy graph".

This view of the sampling walk illuminates its properties. Specifically, H's cover time is $O(mn^2)$ since there are mn edges and n nodes. Hence, the above "inefficient" algorithm actually only requires $(n-1) \times \text{cover-time}(H) = O(mn^3)$ samples. We will also subsequently use the fact that since H is m-regular, its stable distribution is uniform across all nodes.

3.1.1 Multiple Independent Random Walks

Random walks experience dramatic speedups in cover time, hitting time, etc., when they are split into multiple shorter walks; [12] provides a recent survey and results. These speedups naturally require the walks to be independent. In this section, we consider performing $p \leq n$ random walks in parallel, with the starting point of each walk chosen independently and uniformly from the nodes (and thus according to the stationary distribution on H). Running these p walks will require $O(p \log n)$ space. The main theorem of this section establishes that it is possible to efficiently perform p independent, parallel walks in H.

▶ **Theorem 3.** *Given $p \leq n$ parallel random walks in H, each starting at an independently-chosen uniformly random node, we can simulate one independent step of each walk using $O(\log n/\log\log n)$ total samples.*

Issue 1: Multiple Walks can use a Sampled Edge. The first issue we encounter is that a single sample may be a valid move for multiple walks. If we allow multiple walks to use the same sample, we introduce obvious dependence; if we only allow one of our walks to use the sample, we are "slowing down" walks that have collisions, and again introducing dependence.

When multiple walks are at the same node, we will handle them independently in the following way. We partition the p walks into $B_1 \cup B_2 \cup \ldots \cup B_r$ where each B_i contains at most one walk at each node. We process each batch in turn and hence the total number of samples required equals the number of samples required for a batch multiplied by the number of batches. The next lemma establishes that it suffices to consider $r = O(\log n/\log\log n)$ batches.

▶ **Lemma 4.** *With high probability, no node ever contains more than $O(\log n/\log\log n)$ walks.*

Proof. Consider a fixed node at a fixed time. Let Z be the number of walks in this node. Note that $Z \sim Bin(p, 1/n)$ since each walk is independent and is equally likely to be at any node. Hence $\mathbb{E}[Z] = p/n \leq 1$. By an application of the Chernoff bound, for some large constant c we have $\mathbb{P}[Z \geq c\log n/\log\log n] \leq n^{-10}$. The lemma follows by taking the union bound over the n nodes and poly n time-steps. ◀

Henceforth, we assume that at most one walk is at each node, i.e., we analyze how many samples are required to process a single batch. The remaining case where a sampled edge may be valid for multiple walks is if there are walks at both endpoints. To solve this problem, we randomly orient each sampled edge so that it is valid for only one walk. This increases the expected number of samples required by a factor of 2.

Issue 2: Negative Correlation. We have reduced the problem to the following situation: we have p distinct nodes $u_1, \ldots, u_p$ and can sample arcs uv uniformly from the set $E^+ = \{uv : \{u, v\} \in E_G\}$, i.e., the set of arcs formed by bidirecting each edge in E_G. Note that $|E^+| = 2|E_G|$. The goal is to generate a set of arcs $\{u_1v_1, \ldots, u_pv_p\}$ such that each arc is chosen independently and for each i,

$$v_i = \begin{cases} v \in_R \Gamma(u_i) & \text{with probability } d_G(v_i)/|E^+| \\ u_i & \text{with probability } 1 - d_G(v_i)/|E^+| \end{cases} \tag{1}$$

where $\Gamma(u_i) = \{v : \{u, v\} \in E\}$ is the neighborhood of u_i in G, and where $v \in_R \Gamma(u_i)$ denotes an edge drawn randomly from the uniform distribution over the set $\Gamma(u_i)$.

Consider the following procedure: draw a single sample $uv \in_R E^+$ and, for each i, set $v_i = v$ if $u = u_i$, or $v_i = u_i$ otherwise. This procedure picks each v_i according to the desired distribution:

$$\mathbb{P}[v_i = u_i] = 1 - d_G(v_i)/|E^+|$$

and conditioned on $\{v_i \neq u_i\}$, v_i is uniformly chosen from $\Gamma(u_i)$. Unfortunately, the procedure obviously does not satisfy the independence requirement because the events $\{u_i = v_i\}$ and $\{u_j = v_j\}$ are negatively correlated. However, the following theorem establishes that, with only $O(1)$ samples from E^+ in expectation, it is possible to sample independently according to the desired distribution.

▶ **Theorem 5** (Efficient Parallel Sampling). *There exists an algorithm that returns samples $(v_1, \ldots, v_p)$ drawn from the desired distribution (1) while using at most $2e - 1$ samples from E^+ in expectation.*

Proof. Our algorithm operates in rounds; each round uses at most 2 samples from E^+. At the beginning of a round, suppose we have already assigned values to $v_1, \ldots, v_i$ for $i \geq 0$. Then the round proceeds as follows:

1. Sample $uv \in E^+$:
   - **a.** If $u \notin \{u_{i+1}, \ldots, u_p\}$ then set $v_{i+1} = u_{i+1}, \ldots, v_p = u_p$
   - **b.** If $u = u_j$ for some $j \in \{i+1, \ldots, p\}$ then sample an additional arc $wx \in E^+$
     - **i.** If $w \in \{u_{i+1}, \ldots, u_{j-1}\}$ then set $v_{i+1} = u_{i+1}, \ldots, v_{j-1} = u_{j-1}, v_j = u_j$
     - **ii.** If $w \notin \{u_{i+1}, \ldots, u_{j-1}\}$ then set $v_{i+1} = u_{i+1}, \ldots, v_{j-1} = u_{j-1}, v_j = v$

and we repeat the process until all $v_1, \ldots, v_p$ have been assigned.

To analyze the algorithm we define $T_j = \{u_j v : \{u_j, v\} \in E_G\}$ to be the set of $d_G(u_j)$ arcs leaving u_j and note that because $u_1, \ldots, u_p$ are distinct, $T_1, \ldots, T_p$ are disjoint. Also define A_j to be the event that $\{v_j \neq u_j\}$. Then, in any round in which v_j hasn't yet been assigned:

$$\begin{aligned}\mathbb{P}\left[v_j \text{ is assigned and } A_j | v_j \text{ is assigned}\right] &= \frac{|T_j|}{|E^+| - \sum_{k=i+1}^{j-1} |T_k|} \cdot \frac{|E^+| - \sum_{k=i+1}^{j-1} |T_k|}{|E^+|} \\ &= \frac{|T_j|}{|E^+|} \qquad (2)\end{aligned}$$

and hence v_j is chosen according the desired distribution.

We next show that each v_j is chosen independently. First observe that, conditioned on A_j, v_j is independent of $(v_1, \ldots, v_{j-1}, v_{j+1}, \ldots, v_p)$. Hence, it suffices to show that all A_j are independent. Note that the RHS of (2) does not depend on decisions made in previous rounds. Hence, we may deduce that A_j is independent of the outcome of rounds before v_j is assigned. Hence, for any $1 \leq i_1 < i_2 < \ldots < i_r \leq p$,

$$\begin{aligned}\mathbb{P}\left[A_{i_1} \cap \ldots \cap A_{i_r}\right] &= \mathbb{P}\left[A_{i_1}\right] \mathbb{P}\left[A_{i_2} | A_{i_1}\right] \ldots \mathbb{P}\left[A_{i_r} | A_{i_1} \cap \ldots \cap A_{i_{r-1}}\right] \\ &= \mathbb{P}\left[A_{i_1}\right] \mathbb{P}\left[A_{i_2}\right] \ldots \mathbb{P}\left[A_{i_r}\right] .\end{aligned}$$

The worst case for the expected number of samples is achieved when $p = |E^+|$ and each set is of size 1. For the algorithm not to terminate in a given round, we need $u \in \{u_{i+1}, \ldots, u_p\}$ and hence the index of the sampled u needs to strictly increase over previous rounds. The probability of this happening for r rounds is $\binom{m}{r}/m^r$ and the expected number of rounds which do not terminate is $\sum_{r=1}^{m} \binom{m}{r}/m^r \leq e - 1$. Because each non-terminating round involves two samples, the expected total number of samples is thus at most $2e - 1$. ◀

3.2 Connectivity Algorithm and Analysis

Our algorithm adapts a technique of Feige [13] for determining graph connectivity via a two step process. We first test whether G contains any connected components containing k or fewer nodes (for some $k < n$ to be chosen). If all of the connected components of G contain at least k nodes, we choose $O((n \log n)/k)$ nodes at random, and verify that they are all connected to each other. Note that we can expect to have chosen a vertex from each connected component. If we find that all of our chosen vertices are connected, we conclude that G is connected; otherwise, we conclude that G is disconnected.

Our connectivity algorithm thus relies on algorithms for two problems: 1) determining whether the graph has any connected components below a certain size, and 2) determining whether a set of nodes is mutually connected in the graph. In the next two sections, we develop algorithms with sample/space tradeoffs for each of these two problems. We then use them in an algorithm for determining whether the graph G is connected.

3.2.1 Finding Small Components

Our first subproblem is to determine whether the graph has any connected components below a certain size. Given a node v, let the set of nodes in the connected component containing v be denoted $\mathrm{cc}(v)$.

▶ **Lemma 6.** *Given a node v of G and a parameter r, we can distinguish between the case where $|\mathrm{cc}(v)| < r$ and the case where $|\mathrm{cc}(v)| > 2r$ with constant probability using $O(mr^2)$ samples and $\tilde{O}(1)$ space.*

Proof. We perform a sampling walk of length $O(mr^2)$ samples. During this walk, we maintain a 1.1-approximation of the number of distinct vertices visited using an F_0 estimator [22]. If the estimated number of vertices visited is at least $3r/2$, we conclude that $|\mathrm{cc}(v)| \geq r$; otherwise, we conclude that $|\mathrm{cc}(v)| \leq 2r$.

If $|\mathrm{cc}(v)| \leq r$, we will clearly visit at most r nodes. Our algorithm correctly concludes this so long as the F_0 estimator returns the promised approximation. If $|\mathrm{cc}(v)| \geq 2r$, we need to argue that in $O(mr^2)$ samples we will hit at least $2r$ distinct nodes (except with constant probability). This follows from a result by Barnes [5, Thm 1.3] that states that for any connected (multi-)graph, it takes $O(\mathcal{M}\mathcal{N})$ time in expectation to hit either $\mathcal{N}$ distinct nodes or $\mathcal{M}$ distinct edges. Using $\mathcal{M} = 2mr$ and $\mathcal{N} = 2r$ establishes the result. ◀

▶ **Theorem 7.** *We can determine whether G has a connected component of size less than 2^k using $O(p)$ space and $\tilde{O}(2^k \cdot mn/p)$ samples for any $p \leq n$.*

Proof. Our algorithm has k rounds, each corresponding to a value $r = 1, 2, 4, \ldots, 2^{k-1}$. In each round we reach one of two conclusions: 1) G has no connected components with size in the range $[r, 2r]$ or 2) there exists a connected component of G of size $< 3r$. All graphs satisfy at least one of these conclusions. We then determine G has no connected component of size less than 2^k if we never reach the first conclusion.

At a given value of r, we choose $O(n \log n/r)$ nodes, so that we hit any connected component of at least r nodes with high probability. From each node, we perform $\tilde{O}(1)$ random walks of length $\tilde{O}(mr^2)$ samples; from Lemma 6 this will suffice to determine with high probability whether any of these nodes is in a connected component of size $\leq 2r$.

We choose p nodes at a time, and perform p walks in parallel. From Theorem 3 we can perform each set of p walks using $\tilde{O}(mr^2)$ samples. The number of samples required for each r value is then $O(\frac{n}{rp} mr^2) = O(mnr/p)$, and we thus require a total number of samples $O(mn2^k/p)$. ◀

3.2.2 Checking Mutual Connectivity

The remaining subproblem is to determine whether a set of randomly-chosen nodes is mutually connected.

▶ **Lemma 8.** *We can determine whether a set of $O(p)$ randomly-chosen nodes is mutually connected in G using $\tilde{O}(p)$ space and $\tilde{O}(mn^2/p^2)$ samples for any $p \leq n$.*

Proof. In the context of traditional random walks, Feige [14] provides a method for testing whether two nodes s and t are connected using space $\tilde{O}(p)$ and a total of $\tilde{O}(mn/p)$ random-walk steps. They proceed by choosing p "landmark" nodes; we then run $O(\log n)$ random walks from each landmark and from s and t. Each walk is of length $\tilde{O}(mn/p^2)$. During these walks we build a union-find data structure indicating which sets of landmark nodes are connected. At the end of the algorithm, we conclude that s and t are connected if they are in the same union-find component.

Since H is regular, the landmark selection process chooses each node with equal probability. Using Feige's algorithm on the p randomly-chosen landmarks determines whether this set of p nodes is mutually connected. The graph H has n nodes and mn edges, so from [14] each walk should be of length $\tilde{O}(mn^2/p^2)$. Using Theorem 3 we can simulate the p walks with a total of $\tilde{O}(mn^2/p^2)$ samples. ◀

We are now ready to prove our main connectivity result.

▶ **Theorem 9.** *Given sampling access to a graph G, we can determine with high probability whether G is connected using $O(p \log n)$ space and $\tilde{O}(mn^2/p^2)$ samples, for any $p \leq n$.*

Proof. We use Theorem 7 with $2^k = n/p$ to verify that G has no connected components of size less than n/p. If it has such a component, then G is disconnected. If not, we choose $O(p \log n)$ random vertices, hitting each remaining component with high probability. Using Lemma 8, we test that these vertices are mutually connected. Since we have chosen enough vertices to hit every connected component, this suffices to show that the graph is connected. Each of the two subproblems requires $O(mn^2/p^2)$ samples and $\tilde{O}(p)$ space, so these are the sample and space requirements of our algorithm. ◀

4 Rank Estimation

In this section we study the rank estimation problem, namely, that of distinguishing if a stream of vectors is coming from a full dimensional subspace or a subspace of low rank. This is quite useful in data streams and machine learning tasks, for which the vectors correspond to examples. If the subspace is rank-deficient, then one might be interested in a low rank approximation to it. This problem is also relevant in testing codewords, which has been studied in the streaming model in [36].

We will start with a lower bound and then mention a simple matching upper bound for a natural setting of parameters. The lower bound is described in terms of communication complexity, though it yields a lower bound on the memory for data stream algorithms via a standard simulation whereby the state of the streaming algorithm is the message between players in a communication protocol. We mention this after presenting the lower bound for the communication game.

The authors of [39] show a tight connection between learning tasks that can be accomplished via bounded communication algorithms and learning tasks that can be accomplished via "statistical query" algorithms. They operate in the following quite general communication model, and we leverage their result to show a space-sample lower bound trade-off for a streaming version of the rank estimation problem: the task is to distinguish

1. (Case 1) A sequence of t samples chosen uniformly from some rank $n/2$ subspace $S \subseteq \{0,1\}^n$.
2. (Case 2) A sequence of t samples chosen uniformly from $\{0,1\}^n$.

A "statistical query" algorithm for this task, in the language of [39], is an algorithm that adaptively proposes a sequence of functions $f_1, f_2, \ldots$ with $f_i : \{0,1\}^n \to [-1,1]$, and receives estimates of $E_x[f_i(x)]$ that have been corrupted via some adversarial noise. We say that there exists an n-query statistical query algorithm with tolerance τ for this testing task if, for every rank $n/2$ subspace S, after asking n statistical queries $f_1, \ldots, f_n$, with responses $r_1, \ldots, r_n$ that satisfy $|r_i - \mathrm{E}_{x \leftarrow unif[S]}[f_i(x)]| \leq \tau$, the algorithm will output *rank n/2* with probability at least 3/4, and if the responses satisfy $|r_i - \mathrm{E}_{x \leftarrow unif[\{0,1\}^n]}[f_i(x)]| \leq \tau$, then the algorithm will output *full rank* with probability at least 3/4, where the probability is over the randomness of the algorithm that decides on the next query f_i given $f_1, \ldots, f_{i-1}$ and $r_1, \ldots, r_{i-1}$.

▶ **Proposition 10.** *Any statistical query algorithm for distinguishing the uniform distribution over rank $n/2$ subspaces of $\{0,1\}^n$ from a uniform distribution over $\{0,1\}^n$ using statistical queries of tolerance $\tau > \frac{1}{2^{n/8}}$ requires at least $\Theta(2^{n/4})$ statistical queries.*

The following lemma is the core to the proof.

▶ **Lemma 11.** *Let $f : \{0,1\}^n \to [-1,1]$ be a function with $\sum_{x\in\{0,1\}^n} \frac{f(x)}{2^n} = \mu$, and let S be a rank $n/2$ subspace of $\{0,1\}^n$. Define the random variable X_f by choosing S uniformly at random from the set of all rank $n/2$ subspaces of $\{0,1\}^n$, and then set $X_f = E_{x\leftarrow S}[f(x)]$.*

$$\Pr[|X_f - \mu| > \frac{1}{2^{n/8}}] = O(1/2^{n/4}).$$

Proof. First note that $\mathrm{E}[X_f] \in [\mu - 1/2^{n/2}, \mu + 1/2^{n/2}]$, as the distribution obtained by selecting a random rank $n/2$ subspace S, then choosing a random $x \in S$ places probability $1/2^{n/2}$ on the zero vector $\vec{0}$, and the remaining $2^n - 1$ vectors have equal weight. We will now upper bound $\mathrm{Var}[X_f]$, and then apply Chebyshev's inequality. In the following calculations, $\#S$ denotes the number of rank $n/2$ subspaces of $\{0,1\}^n$.

$$\begin{aligned}\mathrm{E}[X_f^2] &= \mathrm{E}_S\left[\left(\frac{1}{2^{n/2}}\sum_{x\in S} f(x)\right)^2\right] = \mathrm{E}_S\left[\frac{1}{2^n}\sum_{x\in S} f(x)^2 + \frac{1}{2^n}\sum_{x,x'\in S, x\neq x'} f(x)f(x')\right] \\ &\le 1/2^{n/2} + \frac{1}{2^n}\frac{1}{\#S}\sum_S \sum_{x,x'\in S,x\neq x'} f(x)f(x'),\end{aligned}$$

which is equal to

$$2^{-n/2} + \frac{2^{-n}}{\#S}\left(2f(\vec{0})\frac{\#S}{2^{n/2}-1}\sum_{x'\neq 0} f(x') + \sum_{x\neq\vec{0}} f(x)\frac{\#S}{(2^{n/2}-1)(2^{n/2}-2)}\sum_{x'\notin\{\vec{0},x\}} f(x')\right).$$

Noting that $|f(x)| \le 1$, and $\sum_{x\neq\vec{0}} f(x) \in [2^n\mu - 1, 2^n\mu + 1]$, and $\sum_{x'\notin\{\vec{0},x\}} f(x') \in [2^n\mu - 2, 2^n\mu + 2]$, the above expression lies in the range $\mu \pm O(1/2^{n/2})$. Hence $\mathrm{Var}[X_f] = O(1/2^{n/2})$, and $\Pr[|X_f - \mu| \ge 1/2^{n/8}] \le O(1/2^{n/4})$, as desired. ◀

Proof of Proposition 10. Let distribution D be defined to be the uniform distribution over $\{0,1\}^n$, and let S be a rank $n/2$ subspace that has been chosen uniformly at random from the set of rank $n/2$ subspaces of $\{0,1\}^n$, and define D_S to be the uniform distribution over S. Let $f_1, \ldots, f_n$ be a sequence of $n = \Theta(2^{n/4})$ statistical queries, corresponding to the responses $r_1, \ldots, r_n$ with $r_i = \mathrm{E}_{x\leftarrow D}[f(x)]$. We will now show that, with probability greater than $1/2$ over the randomness of the choice of subspace S, it will hold that $|r_i - \mathrm{E}_{x\leftarrow D_S}[f_i(x)]| \le 1/2^{n/8}$. Indeed, this follows immediately from Lemma 11 via a union bound over the n events. To conclude, this shows that after n queries of tolerance at most $1/2^{n/8}$, with probability at least $1/2$, no information is given regarding whether the responses correspond to D versus D_S for some S, hence the probability of correctly guessing "D" versus "D_S" can be at most $3/4$ in one of the two cases. ◀

For the purposes of our lower bound, Lemma 3.3 from [39] (restated below) immediately translates the statistical query lower bound of Proposition 10 into a lower bound on the number of samples required by any algorithm in the following bounded communication model: there is one player per example, and the t players each speak once, one after the other. The i-th player sees the message sent of each of the first $i-1$ players, and then sends its message.

▶ **Lemma 12** (Lemma 3.3 from [39] (restated))**.** *If a given testing (or learning) task can be solved with probability $> 1-\delta$ using t examples via a communication bounded protocol that employs s bits of communication per example then it can be solved with probability $> 1-2\delta$ via a statistical query algorithm that asks $2st$ statistical queries of tolerance $\tau = \delta/(t2^s)$.*

Proposition 10 and Lemma 12 yield the following:

▶ **Theorem 13.** *Any bounded-communication protocol that distinguishes samples from a rank $n/2$ subspace of $\{0,1\}^n$ with constant probability of success above $1/2$ that uses $s \leq n/8$ bits of communication per example requires at least $\Theta(2^{n/8-s})$ samples.*

Streaming Lower Bound: This communication bound implies an equivalent bound on the bits of memory required by any streaming algorithm, via the standard technique of noting that the memory contents of the streaming algorithm after seeing each example "communicates" information between examples of the input stream. Note that this lower bound holds even if the lower bound of Theorem 13 were instead only to hold in the communication model in which the i-th player only sees the message sent of the $(i-1)$-st player.

Nearly Matching Upper Bound: Our lower bound is tight to within constant factors for the natural case of $s = O(\log n)$ and $t = O(2^n)$, where a naïve algorithm suffices: choose a random coordinate unit vector $x = e_i$. Note that for any rank $n/2$ space, at most $n/2$ of these can lie in it. Take $O(2^n)$ samples, looking to see if any of them equal x. If we are sampling from a rank-n space, then we will find x with high constant probability; if we are sampling from a rank $n/2$ space, we will find it with probability at most $1/2$.

We note that very recently the work of Raz [34] improves upon the framework of [39]. It may be possible to apply it to strengthen the above theorem to require $b = \Omega(n^2)$ bits of space with fewer than $2^{\Omega(n)}$ samples, though it is not immediate. In either case, our result shows an exponential number of samples is needed to achieve polylogarithmic memory by a data stream algorithm, and we do not know how to prove this in a different way, e.g., by using more standard reductions from communication complexity.

5 Conclusions

We have introduced a new model for analyzing tradeoffs between sample and space complexity of streaming algorithms.

There are a number of open questions, a few of which we list here:

1. (Collision Probability) For the collision probability question, what is the optimal space complexity? We conjecture that $s \cdot t = \tilde{\Omega}(n)$ should hold. After resolving the dependence on n, a next natural question would be to understand the dependence on ϵ and δ. Recent work [1] may be helpful in this regard.
2. (Frequeny Moments) Can one obtain optimal tradeoffs for other frequency moments $F_k = \sum_j p_j^k$? In this work we focused solely on F_2. Perhaps more generally one could provide a general set of techniques for analyzing various distribution learning problems in this framework.
3. (Connectivity) What lower bounds can one show for testing connectivity? It seems we can show some preliminary lower bounds via a reduction from the set disjointness problem, though currently they are far from the upper bounds.
4. (General) The techniques used for the different problems studied here are not directly related to each other. Is it possible to develop a more general framework which unifies the results for these problems?

Acknowledgments. We thank the anonymous referees for their helpful comments.

References

1 Jayadev Acharya, Alon Orlitsky, Ananda Theertha Suresh, and Himanshu Tyagi. The complexity of estimating rényi entropy. *CoRR*, abs/1408.1000, 2014.

2 Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999.

3 Alexandr Andoni, Andrew McGregor, Krzysztof Onak, and Rina Panigrahy. Better bounds for frequency moments in random-order streams. *CoRR*, abs/0808.2222, 2008.

4 Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Sampling algorithms: lower bounds and applications. In *STOC*, pages 266–275, 2001. doi:10.1145/380752.380810.

5 Greg Barnes and Uriel Feige. Short Random Walks on Graphs. *SIAM Journal on Discrete Mathematics*, 9(1):19, 1996. doi:10.1137/S0895480194264988.

6 Marc Bury and Chris Schwiegelshohn. Sublinear estimation of weighted matchings in dynamic data streams. *CoRR*, abs/1505.02019, 2015. URL: http://arxiv.org/abs/1505.02019.

7 Amit Chakrabarti, Graham Cormode, and Andrew McGregor. Robust lower bounds for communication and stream computation. In *STOC*, pages 641–650, 2008. doi:10.1145/1374376.1374470.

8 Amit Chakrabarti, T. S. Jayram, and Mihai Patrascu. Tight lower bounds for selection in randomly ordered streams. In *SODA*, pages 720–729, 2008. doi:10.1145/1347082.1347161.

9 Steve Chien, Katrina Ligett, and Andrew McGregor. Space-efficient estimation of robust statistics and distribution testing. In *ICS*, pages 251–265, 2010.

10 Kenneth L. Clarkson and David P. Woodruff. Numerical linear algebra in the streaming model. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 – June 2, 2009*, pages 205–214, 2009.

11 Thomas M. Cover, Michael A. Freedman, and Martin E. Hellman. Optimal finite memory learning algorithms for the finite sample problem. *Information and Control*, 30(1):49–85, 1976.

12 Klim Efremenko and Omer Reingold. How Well Do Random Walks Parallelize? In *APPROX-RANDOM*, volume 5687, pages 476–489, Berlin, Heidelberg, 2009. doi:10.1007/978-3-642-03685-9.

13 Uriel Feige. A fast randomized logspace algorithm for graph connectivity. *Theor. Comput. Sci.*, 169(2):147–160, 1996. doi:10.1016/S0304-3975(96)00118-1.

14 Uriel Feige. A Spectrum of Time-Space Trade-offs for Undirected s-t Connectivity. *Journal of Computer and System Sciences*, 54(2):305–316, April 1997. doi:10.1006/jcss.1997.1471.

15 R. A. Fisher. On the mathematical foundations of theoretical statistics. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 222:309–368, 1922.

16 I.J. Good. Surprise indexes and p-values. *J. Statistical Computation and Simulation*, 32:90–92, 1989.

17 Michael Greenwald and Sanjeev Khanna. Efficient online computation of quantile summaries. In *ACM International Conference on Management of Data*, pages 58–66, 2001. doi:10.1145/375663.375670.

18 Sudipto Guha and Zhiyi Huang. Revisiting the direct sum theorem and space lower bounds in random order streams. In *ICALP*, pages 513–524, 2009. doi:10.1007/978-3-642-02927-1_43.

19 Sudipto Guha and Andrew McGregor. Space-efficient sampling. In *AISTATS*, pages 169–176, 2007.

20 Sudipto Guha and Andrew McGregor. Stream order and order statistics: Quantile estimation in random-order streams. *SIAM Journal on Computing*, 38(5):2044–2059, 2009. doi:10.1137/07069328X.

21 Martin E Hellman and Thomas M Cover. Learning with finite memory. *The Annals of Mathematical Statistics*, pages 765–782, 1970.

22 Daniel M. Kane, Jelani Nelson, and David P. Woodruff. An optimal algorithm for the distinct elements problem. In *PODS*, pages 41–52, 2010. doi:10.1145/1807085.1807094.

23 Jack Koplowitz. Necessary and sufficient memory size for m-hypothesis testing. *Information Theory, IEEE Transactions on*, 21(1):44–46, 1975.

24 Frank Thomson Leighton and Ronald L. Rivest. Estimating a probability using finite memory. *IEEE Trans. Information Theory*, 32(6):733–742, 1986.

25 Yi Li, Huy L. Nguyen, and David P. Woodruff. On sketching matrix norms and the top singular vector. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1562–1581, 2014.

26 Andrew McGregor, A. Pavan, Srikanta Tirthapura, and David P. Woodruff. Space-efficient estimation of statistics over sub-sampled streams. In *PODS*, pages 273–282, 2012. doi:10.1145/2213556.2213594.

27 Andrew McGregor and Paul Valiant. The shifting sands algorithm. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 453–458, 2012.

28 Michael Mitzenmacher and Eli Upfal. *Probability and computing – randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.

29 J. Ian Munro and Mike Paterson. Selection and sorting with limited storage. *Theor. Comput. Sci.*, 12:315–323, 1980.

30 S. Muthukrishnan. Stochastic data streams. In *MFCS*, page 55, 2009. doi:10.1007/978-3-642-03816-7_5.

31 AS Nemirovsky and DB Yudin. *Problem Complexity and Method Efficiency in Optimization.* J. Wiley @ Sons, New York, 1983.

32 Yurii Nesterov. Smooth minimization of non-smooth functions. *Math. Program.*, 103(1):127–152, 2005.

33 Piyush Rai, Hal Daumé III, and Suresh Venkatasubramanian. Streamed learning: One-pass SVMs. In *IJCAI*, pages 1211–1216, 2009.

34 Ran Raz. Fast learning requires good memory: A time-space lower bound for parity learning. *CoRR*, abs/1602.05161, 2016.

35 Omer Reingold. Undirected connectivity in log-space. *Journal of the ACM*, 55(4):1–24, September 2008. doi:10.1145/1391289.1391291.

36 Atri Rudra and Steve Uurtamo. Data stream algorithms for codeword testing. In *Automata, Languages and Programming*, pages 629–640. Springer, 2010.

37 Florin Rusu and Alin Dobra. Sketching sampled data streams. In *ICDE*, pages 381–392, 2009. doi:10.1109/ICDE.2009.31.

38 Ohad Shamir. Fundamental limits of online and distributed algorithms for statistical learning and estimation. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 163–171, 2014.

39 J. Steinhardt, G. Valiant, and S. Wager. Memory, communication, and statistical queries. Manuscript, 2015.

40 TheJournal.ie. 50 billion instant messages expected to be sent each day in 2014. URL: http://businessetc.thejournal.ie/sms-messaging-falls-1262017-Jan2014/.

41 David P. Woodruff. The average-case complexity of counting distinct elements. In *ICDT*, pages 284–295, 2009. doi:10.1145/1514894.1514928.

Counting Matchings with k Unmatched Vertices in Planar Graphs

Radu Curticapean[*][†]

Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), Budapest, Hungary
radu.curticapean@gmail.com

Abstract

We consider the problem of counting matchings in planar graphs. While *perfect* matchings in planar graphs can be counted by a classical polynomial-time algorithm [26, 33, 27], the problem of counting all matchings (possibly containing unmatched vertices, also known as *defects*) is known to be #P-complete on planar graphs [23].

To interpolate between matchings and perfect matchings, we study the parameterized problem of counting matchings with k unmatched vertices in a planar graph G, on input G and k. This setting has a natural interpretation in statistical physics, and it is a special case of counting perfect matchings in k-apex graphs (graphs that become planar after removing k vertices). Starting from a recent #W[1]-hardness proof for counting perfect matchings on k-apex graphs [12], we obtain:

- Counting matchings with k unmatched vertices in planar graphs is #W[1]-hard.
- In contrast, given a plane graph G with s distinguished faces, there is an $\mathcal{O}(2^s \cdot n^3)$ time algorithm for counting those matchings with k unmatched vertices such that all unmatched vertices lie on the distinguished faces. This implies an $f(k,s) \cdot n^{O(1)}$ time algorithm for counting perfect matchings in k-apex graphs whose apex neighborhood is covered by s faces.

1998 ACM Subject Classification G.2.1 Combinatorics, G.2.2 Graph Theory, F.1.3 Complexity Measures and Classes

Keywords and phrases counting complexity, parameterized complexity, matchings, planar graphs

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.33

1 Introduction

The study of the computational complexity of counting problems was introduced in a seminal paper by Valiant [34] that established the class #P and proved that counting perfect matchings in an unweighted bipartite graph is #P-complete. In a companion paper [35], Valiant proved that counting all (not necessarily perfect) matchings in a graph is #P-complete as well. Even prior to these initial complexity-theoretic results, problems related to matchings and perfect matchings have played an important role in various scientific disciplines.

For instance, the number of perfect matchings in a bipartite graph G arises in enumerative combinatorics and algebraic complexity as the *permanent* of the bi-adjacency matrix associated with G [3, 1]. In statistical physics, counting perfect matchings amounts to evaluating the *partition function* of the *dimer model* [27, 26, 33]: The physical interpretation here is that vertices are discrete points that are occupied by atoms, while edges are interpreted as bonds

[*] Part of this work was carried out while the author was a PhD student at Saarland University in Saarbrücken, Germany, and while he was visiting the Simons Institute for the Theory of Computing in Berkeley, USA. The material also appears in his PhD thesis [10].
[†] The author is supported by the ERC grant PARAMTIGHT, no. 280152.

between the corresponding atoms. The partition function of G is then essentially defined as the number of perfect matchings in G, and it encodes thermodynamic properties of the associated system. Likewise, the problem of counting all matchings is known to statistical physicists as the *monomer-dimer model* [23]; in this setting, some points may be unoccupied by atoms. In the intersection of chemistry and computer science, the number of matchings of a graph (representing a molecule) is known as its *Hosoya index* [20].

In view of these applications and the #P-hardness of counting matchings and perfect matchings, several relaxations were considered to cope with these problems. Among these, *approximate* counting and the restriction to *planar* graphs proved most successful. However, once we start incorporating these relaxations, the seemingly very similar problems of counting matchings and counting perfect matchings exhibit stark differences:

- On planar graphs, perfect matchings can be counted in polynomial time by the classical and somewhat marvelous FKT method [27, 26, 33], which reduces this problem to the determinant. The problem of counting all matchings is however #P-complete on planar graphs [23]. In particular, the algebraic machinery in the FKT method breaks down for non-perfect matchings.
- It was shown that the number of matchings in a graph admits a polynomial-time randomized approximation scheme (FPRAS) on general graphs [24]. By a substantial extension of this approach, an FPRAS for counting perfect matchings in bipartite graphs was obtained [25] – but despite great efforts, no FPRAS is known for general graphs.

In the present paper, we focus on the differing behavior of matchings and perfect matchings on planar graphs. To this end, we study the problem #PlanarDefectMatch of counting matchings with k unmatched vertices (which we call k-defect matchings) in a planar graph G, on input G and k. This problem is clearly #P-hard under Turing reductions, as the #P-hard number of matchings in G can be obtained as the sum of numbers of k-defect matchings in G for $k = 0, \dots, |V(G)|$. On the other hand, #PlanarDefectMatch can easily be solved in time $|V(G)|^{\mathcal{O}(k)}$, as we can simply enumerate all k-subsets $X \subseteq V(G)$ that represent potential defects, count perfect matchings in the planar graph $G - X$ by the FKT method, and sum up these numbers.

1.1 Parameterized counting problems

The fact that #PlanarDefectMatch is #P-hard and polynomial-time solvable for constant k suggests that this problem benefits from the framework of *parameterized counting complexity* [15]. This area is concerned with *parameterized* counting problems, whose instances x come with *parameters* k, such as #PlanarDefectMatch or the problem #Clique of counting k-cliques in an n-vertex graph. Intuitively, the parameterized problem #PlanarDefectMatch considers k-defect matchings in planar graphs with $k \ll n$, and the physical interpretation in terms of the monomer-dimer model is that each configuration of the system admits only a small number of "vacant" points that are not occupied by atoms.

Note that both #PlanarDefectMatch and #Clique can be solved in time $n^{\mathcal{O}(k)}$ and are hence in the so-called class XP. One important goal for such problems lies in finding algorithms with running times $f(k) \cdot |x|^{\mathcal{O}(1)}$ for computable functions f, which renders the problems *fixed-parameter tractable* (FPT) [15, 16]. If no FPT-algorithms can be found for a given problem, one can try to show its #W[1]-hardness. This essentially boils down to finding a parameterized reduction from #Clique, and it shows that FPT-algorithms for the problem would imply FPT-algorithms for #Clique, which is considered unlikely.

For instance, to prove #W[1]-hardness of #PlanarDefectMatch by reduction from #Clique, we would need to find an algorithm that counts k-cliques of an n-vertex graph in time

$f(k) \cdot n^{\mathcal{O}(1)}$ with an oracle for #PlanarDefectMatch. Additionally, the algorithm should only invoke the oracle for counting k'-defect matchings with $k' \leq g(k)$. Here, both the function f appearing in the running time and the blow-up function g are arbitrary computable functions.

Furthermore, parameterized reductions can also be used to obtain lower bounds under the exponential-time hypothesis #ETH, which postulates that the satisfying assignments to formulas φ in 3-CNF cannot be counted in time $2^{o(n)}$ [13, 21, 22]. For instance, it is known that #Clique cannot be solved in time $n^{o(k)}$ unless #ETH fails [5]. If we reduce from #Clique to a target problem by means of a reduction that invokes only blow-up $\mathcal{O}(k)$, then #ETH also rules out $n^{o(k)}$ time algorithms for the target problem [29].

1.2 Perfect matchings with planar-like parameters

To put #PlanarDefectMatch into context, let us survey some parameterizations for the problem #PerfMatch of counting perfect matchings and see how these connect to #PlanarDefectMatch.

- The FKT method for planar graphs was extended [18, 30, 12] from planar graphs to graphs of fixed genus g, resulting in $\mathcal{O}(4^g \cdot n^3)$ time algorithms for #PerfMatch.
- Polynomial-time algorithms for #PerfMatch were obtained for $K_{3,3}$-free graphs [28, 38] and K_5-free graphs [32]. More generally, for every class of graphs excluding a fixed single-crossing minor H (that is, H can be drawn in the plane with at most one crossing), an $f(H) \cdot n^4$ time algorithm is known [7].
- A simple dynamic programming algorithm yields a running time of $3^t \cdot n^{\mathcal{O}(1)}$ for #PerfMatch on graphs of treewidth t. By using fast subset convolution [37], the running time can be improved to $2^t \cdot n^{\mathcal{O}(1)}$.

Since all of the tractable classes above exclude fixed minors for fixed parameter values, one is tempted to believe that #PerfMatch could be polynomial-time solvable on *each* class of graphs excluding a fixed minor H, and possibly even admit an FPT-algorithm when parameterized by the minimum size of an excluded minor. This last possibility was however ruled out by the following result:[1]

- #PerfMatch is #W[1]-hard on k-apex graphs [12]. For $k \in \mathbb{N}$, a graph G is k-apex if there is a set $A \subseteq V(G)$ of size k such that $G - A$ is planar. The vertices in A are called *apices*. Since k-apex graphs exclude minors on $\mathcal{O}(k)$ vertices, the #W[1]-hardness result for #PerfMatch on k-apex graphs implies #W[1]-hardness of #PerfMatch on graphs excluding fixed minors H (when parameterized by the minimum size of such an H).

Note that #PerfMatch can be solved in time $n^{\mathcal{O}(k)}$ on k-apex graphs by brute-force in a similar way as #PlanarDefectMatch. To cope with the #W[1]-hardness of #PerfMatch in k-apex graphs and potentially obtain faster algorithms, we study two special cases:

1. We consider #PlanarDefectMatch, which is indeed a special case, as discussed below.
2. We consider #PerfMatch in k-apex graphs whose apices are adjacent with only a bounded number of faces in the underlying planar graph. More in Section 1.4 of the introduction.

1.3 From k apices to k defects

To count the k-defect matchings in a planar graph G, we can equivalently count perfect matchings in the k-apex graph G' obtained from G by adding k independent apex vertices adjacent to all vertices of G: Every perfect matching of G' then corresponds to a k-defect matching of G, and likewise, every k-defect matching of G corresponds to precisely $k!$ perfect

[1] In fact, recent unpublished work suggests the existence of constant-sized minors H such that #PerfMatch is #P-hard on H-minor free graphs.

Table 1 Counting matchings under different parameterizations and input restrictions

counting matchings	on planar inputs	on general inputs
with k edges	FPT by [17]	#W[1]-complete by [6, 11]
with k defects	#W[1]-hard by Thm. 1	#P-complete for $k = 0$ by [34]

matchings of G'. This shows that #PlanarDefectMatch reduces to #PerfMatch on k-apex graphs, even when the apices in these latter graphs form an independent set and each apex is adjacent with all non-apex vertices. Note that the #W[1]-hardness for the general problem of #PerfMatch on k-apex graphs does a priori not carry over to the special case #PlanarDefectMatch, as the edges between apices and the planar graph cannot be assumed to be complete bipartite graphs in the general problem.

Nevertheless, we show in Section 3 that #PlanarDefectMatch is #W[1]-hard. To this end, we reduce from #PerfMatch on k-apex graphs by means of a "truncated" polynomial interpolation where we wish to recover only the first k coefficients from a polynomial of degree n. The technique is comparable to that used in the first #W[1]-hardness proofs for counting matchings with k edges [2, 6]. Interestingly enough, our reduction maps k-apex graphs to instances of counting k-defect matchings without incurring any parameter blowup at all. In particular, we obtain the same almost-tight lower bound under #ETH that was known for #PerfMatch on k-apex graphs [12].

▶ **Theorem 1.** #PlanarDefectMatch *is* #W[1]*-hard and admits no* $n^{o(k/\log k)}$ *time algorithm unless* #ETH *fails.*

It should be noted that the "primal" problem of counting matchings with k edges is #W[1]-hard on general graphs [6, 11], but becomes FPT on planar graphs [17]. Furthermore, recall that counting matchings with 0 defects (that is, perfect matchings) in general graphs is #P-hard. See also Table 1 for the complexity of counting matchings in various settings.

1.4 Few apices that also see few faces

In Section 4, we show that #PerfMatch becomes easier in k-apex graphs G when the apex neighborhoods can all be covered by s faces of the underlying planar graph. This setting is motivated by a structural decomposition theorem for graphs G excluding a fixed 1-apex minor H: As shown in [14], based on [31], if G excludes a fixed 1-apex minor H, then there is a constant $c_H \in \mathbb{N}$ such that G can be obtained by gluing together (in a formalized way) graphs that have genus $\leq c_H$ after removing "vortices" from $\leq c_H$ faces and a set A of $\leq c_H$ apex vertices, whose neighborhood in $G - A$ is however covered by $\leq c_H$ faces. Our setting is a simplification of this general situation as we forbid vortices, gluing, and restrict the genus to 0. We obtain an FPT-algorithm for this restricted case:

▶ **Theorem 2.** *Given as input a graph G, a set $A \subseteq V(G)$ of size k and a drawing of $G - A$ in the plane with s distinguished faces $F_1, \ldots, F_s$ such that the neighborhood of A is contained in the union of $F_1, \ldots, F_s$, we can count the perfect matchings of G in time* $2^{\mathcal{O}(2^k \cdot \log(k)+s)} \cdot n^4$.

Note that even with $k = 3$ and $s = 1$, such graphs can have unbounded genus, as witnessed by the graphs $K_{3,n}$ for $n \in \mathbb{N}$: Each graph $K_{3,n}$ is a 3-apex graph whose underlying planar graph (which is an independent set) can be drawn on one single face. However, the genus of $K_{3,n}$ is known to be $\Omega(n)$ [19].

To prove Theorem 2, we first consider a variant of #PlanarDefectMatch where the input graph G is given as a planar drawing with s distinguished faces. The task in this variant is to count k-defect matchings such that all defects are contained in the distinguished faces. This problem is FPT, even when k is not part of the parameter.

▶ **Theorem 3.** *Given as input a planar drawing of a graph G with s distinguished faces $F_1, \ldots, F_s$, the following problem can be solved in time $\mathcal{O}(2^s \cdot n^3)$: Count the matchings in G for which every defect is contained in $V(F_1) \cup \ldots \cup V(F_s)$.*

To prove Theorem 3, we implicitly use the technique of combined signatures [12]: Using a linear combination of two planar gadgets from [36], we show that counting the particular matchings needed in Theorem 3 can be reduced to 2^s instances of #PerfMatch in planar graphs. We can phrase this result in a self-contained way that does not require the general machinery of combined signatures. It should be noted that the case $s = 1$ was already solved by Valiant [36] and that our proof of Theorem 3 is a rather simple generalization of his construction. In a different context, this idea is also used in [9].

More effort is then required to prove Theorem 2, and we do so by reduction to Theorem 3. To this end, we label each vertex in the planar graph $G - A$ with its neighborhood in the apex set A. Each k-defect matching in $G - A$ then has a *type*, which is the k-element multiset of A-neighborhoods of its k defects.[2] We will be able to count k-defect matchings M of any specified type among the $(2^k)^k$ possible types, and we observe that the number of extensions from M to a perfect matching in G depends only on its type. This will allow us to recover the number of perfect matchings in G.

2 Preliminaries

For $n \in \mathbb{N}$, write $[n] = \{1, \ldots, n\}$. Graphs G are undirected and simple. They are unweighted unless specified otherwise. We write $N_G(v)$ for the neighborhood of $v \in V(G)$ in G.

2.1 Polynomials

We denote the degree of a polynomial $p \in \mathbb{Q}[x]$ by $\deg(p)$. If $\mathbf{x} = (x_1, \ldots, x_t)$ is a list of indeterminates, then we write $\mathbb{N}^{\mathbf{x}}$ for the set of all monomials over $\mathbf{x}$. A *multivariate polynomial* $p \in \mathbb{Q}[\mathbf{x}]$ is a polynomial $p = \sum_{\theta \in \mathbb{N}^{\mathbf{x}}} a(\theta) \cdot \theta$ with $a(\theta) \in \mathbb{Q}$ for all $\theta \in \mathbb{N}^{\mathbf{x}}$, where a has finite support. The polynomial p *contains* a given monomial $\theta \in \mathbb{N}^{\mathbf{x}}$ if $a(\theta) \neq 0$ holds. If x is an indeterminate from $\mathbf{x}$, then we write $\deg_x(p)$ for the *degree of* x in p. This is the maximum number $k \in \mathbb{N}$ such that p contains a monomial θ with factor x^k. If $\mathbf{y}$ is a list of indeterminates, then we denote the *total degree of* $\mathbf{y}$ in p as the maximum degree of any monomial $\mathbb{N}^{\mathbf{y}}$ that is contained as a factor of a monomial in p.

Furthermore, if $p \in \mathbb{Q}[x, y]$ is a bivariate polynomial and $\xi \in \mathbb{Q}$ is some arbitrary fixed value, we write $p(\cdot, \xi)$ for the result of the substitution $y \leftarrow \xi$ in p, and we observe that $p(\cdot, \xi) \in \mathbb{Q}[x]$. Likewise, we write $p(\xi, \cdot)$ for the result of substituting $x \leftarrow \xi$.

2.2 (Perfect) matching polynomials

If G is a graph, then a set $M \subseteq E(G)$ of vertex-disjoint edges is called a matching. We write $\mathcal{M}[G]$ for the set of all matchings of G. For $M \in \mathcal{M}[G]$, we write $\mathrm{usat}(M)$ for the set of unmatched vertices in M. If $|\mathrm{usat}(M)| = k$ for $k \in \mathbb{N}$, we say that M is a k-defect matching, and we write $\mathcal{DM}_k[G]$ for the set of k-defect matchings of G. We also write $\mathcal{PM}[G] = \mathcal{DM}_0[G]$ for the set of perfect matchings of G.

[2] This resembles an idea from an algorithm for counting subgraphs of bounded vertex-cover number [11].

If G is an edge-weighted graph with edge-weights $w : E(G) \to \mathbb{Q}$, then we define

$$\#\mathsf{PerfMatch}(G) = \sum_{M \in \mathcal{PM}[G]} \prod_{e \in M} w(e). \tag{1}$$

On planar graphs G, we can efficiently compute $\#\mathsf{PerfMatch}(G)$.

▶ **Theorem 4** ([26, 33, 27])**.** *For planar edge-weighted graphs G, the value $\#\mathsf{PerfMatch}(G)$ can be computed in time $\mathcal{O}(n^3)$.*

If G is a vertex-weighted graph with vertex-weights $w : V(G) \to \mathbb{Q}$, we define

$$\#\mathsf{MatchSum}(G) = \sum_{M \in \mathcal{M}[G]} \prod_{v \in \mathrm{usat}(M)} w(v). \tag{2}$$

Both $\#\mathsf{PerfMatch}$ and $\#\mathsf{MatchSum}$ are also used in [36]. Note that zero-weights have different semantics in the two expressions: A vertex $v \in V(G)$ with $w(v) = 0$ is required to be matched in all matchings $M \in \mathcal{M}[G]$ that contribute a non-zero term to $\#\mathsf{MatchSum}$. An edge $e \in E(G)$ with $w(e) = 0$ can simply be deleted from G without affecting $\#\mathsf{PerfMatch}(G)$.

Finally, if X is a formal indeterminate, we define the defect-generating matching polynomial of unweighted graphs G as

$$\mu(G) := \sum_{M \in \mathcal{M}[G]} X^{|\mathrm{usat}(M)|} = \sum_{k=0}^{n} \#\mathcal{DM}_k[G] \cdot X^k. \tag{3}$$

Note that $\mu(G) = \#\mathsf{MatchSum}(G')$ when G' is obtained from G by assigning weight X to every vertex of G. In this paper, we will be interested in the first k coefficients of $\mu(G)$.

▶ Remark. It is known [4] that for every fixed $\xi \in \mathbb{Q} \setminus \{0\}$, the problem of evaluating $\mu(G; \xi)$ on input G is #P-complete, even on planar bipartite graphs G of maximum degree 3. Note that the evaluation $\mu(G; 0)$ counts the perfect matchings of G.

2.3 Techniques from parameterized counting

Please consider Section 1.1 for an introduction to parameterized counting complexity, and [15] for a more formal treatment. We write $\leq^T_{fpt}$ for parameterized (Turing) reductions between problems (as introduced in Section 1.1). Furthermore, we write $\leq^{lin}_{fpt}$ for such parameterized reductions that incur only linear parameter blowup, i.e., on instances x with parameter k, they only issue queries with parameter $\mathcal{O}(k)$.

Given a universe Ω and several "bad" subsets of Ω, the inclusion-exclusion principle allows us to count those elements of Ω that avoid all bad subsets, provided that we know the sizes of intersections of bad subsets.

▶ **Lemma 5.** *Let Ω be a set and let $A_1, \ldots, A_t \subseteq \Omega$. For $\emptyset \subset S \subseteq [t]$, let $A_S := \bigcap_{i \in S} A_i$ and define $A_\emptyset := \Omega$. Then we have $\left|\Omega \setminus \bigcup_{i \in [t]} A_i\right| = \sum_{S \subseteq [t]} (-1)^{|S|} |A_S|$.*

In applications of Lemma 5, the left-hand side of the equation corresponds to a quantity we wish to determine, while the numbers $|A_S|$ for $S \subseteq [t]$ are computed by oracle calls.

We will also generously use the technique of polynomial interpolation: if a univariate polynomial p has degree n and we can evaluate $p(\xi)$ at $n + 1$ distinct values ξ, then we can recover the coefficients of p. This can be generalized to multivariate polynomials: If p has n variables, all of maximum degree d, and we are given sets $\Xi_1, \ldots, \Xi_n$, all of size $d + 1$, along with evaluations of $p(\xi)$ on all grid points $\xi \in \Xi_1 \times \ldots \times \Xi_n$, then we can determine the coefficients of p in time $\mathcal{O}((d + 1)^{3n})$.

▶ **Lemma 6** ([8]). *Let $p \in \mathbb{Z}[x_1, \ldots, x_n]$ be a multivariate polynomial, and for $i \in [n]$, let the degree of x_i in p be bounded by $d_i \in \mathbb{N}$. Let $\Xi = \Xi_1 \times \ldots \times \Xi_n \subseteq \mathbb{Q}^n$ with $|\Xi_i| = d_i + 1$* for all *$i \in [n]$. Then we can compute the coefficients of p with $\mathcal{O}(|\Xi|^3)$ arithmetic operations when given as input the set $\{(\xi, p(\xi)) \mid \xi \in \Xi\}$.*

3 Hardness of #PlanarDefectMatch

We now prove Theorem 1: Given a *planar* graph G and $k \in \mathbb{N}$, it is #W[1]-hard to count the k-defect matchings of G. This amounts to computing the coefficient of X^k in the matching-defect polynomial $\mu(G)$. We start from the #W[1]-hardness for the following problem #ApexPerfMatch, which follows from Theorem 1.2 and Remark 5.6 in [12]:

▶ **Theorem 7** ([12]). *The following problem #ApexPerfMatch is #W[1]-hard: Compute the value of #PerfMatch(G), when given as input an unweighted graph G and an independent set $A \subseteq V(G)$ of size k such that $G - A$ is planar and each vertex $v \in V(G) \setminus A$ satisfies $|N_G(v) \cap A| \leq 1$. The parameter in this problem is k. Furthermore, assuming #ETH, the problem cannot be solved in time $n^{o(k/\log k)}$.*

In the proof of Theorem 1, we introduce an intermediate problem #RestrDefectMatch:

▶ **Problem 8.** The problem #RestrDefectMatch is defined as follows: Given as input a triple (G, S, k) where G is a planar graph, $S \subseteq V(G)$ is a set of vertices, and $k \in \mathbb{N}$ is an integer, count those k-defect matchings of G whose defects all avoid S, i.e., those k-defect matchings M with $S \cap \mathrm{usat}(M) = \emptyset$. The parameter is k.

The problem #RestrDefectMatch is equivalent (up to multiplication by a simple factor) to the problem #ApexPerfMatch on graphs G whose apices A are all adjacent to a common subset S of the planar graph $G - A$, and to no other vertices. Our overall reduction then proceeds along the chain

$$\#\mathsf{ApexPerfMatch} \leq_{fpt}^{lin} \#\mathsf{RestrDefectMatch} \leq_{fpt}^{lin} \#\mathsf{PlanarDefectMatch}. \tag{4}$$

3.1 From #ApexPerfMatch to #RestrDefectMatch

The first reduction in (4) follows from an application of the inclusion-exclusion principle.

▶ **Lemma 9.** *We have #ApexPerfMatch $\leq_{fpt}^{lin}$ #RestrDefectMatch.*

Proof of Lemma 9. We reduce from #ApexPerfMatch and wish to count perfect matchings in an unweighted graph G with apex set $A = \{a_1, \ldots, a_k\}$ and planar base graph $H = G - A$. Note that A is given as part of the input, and it is an independent set. Furthermore, by definition of #ApexPerfMatch, the set $V(H)$ admits a partition into $V_1 \cup \ldots \cup V_k \cup W$ such that all vertices $v \in V_i$ for $i \in [k]$ are adjacent to the apex a_i and to no other apices, while no vertex $v \in W$ is adjacent to any apex. In other words, each vertex $v \in V(H)$ can be colored by its unique adjacent apex, or by a neutral color if $v \in W$.

Recall that $\mathcal{DM}_k[H]$ denotes the set of k-defect matchings in H. We call a k-defect matching $M \in \mathcal{DM}_k[H]$ *colorful* if $|\mathrm{usat}(M) \cap V_i| = 1$ holds for all $i \in [k]$, and we write $\mathcal{C}$ for the set of all such M. Note that $\mathrm{usat}(M) \cap W = \emptyset$ for $M \in \mathcal{C}$, since none of its k defects are left over for W.

We claim that $\mathcal{PM}[G] \simeq \mathcal{C}$: If $M \in \mathcal{PM}[G]$, then $N = M - A$ satisfies $N \in \mathcal{C}$. Conversely, every $N \in \mathcal{C}$ can be extended to a unique $M \in \mathcal{PM}[G]$ by matching the unique i-colored defect to its unique adjacent apex a_i.

Given oracle access to #RestrDefectMatch, we can determine $\#\mathcal{C}$ by the inclusion-exclusion principle from Lemma 5: For $i \in [k]$, let $\mathcal{A}_i$ denote the set of those $M \in \mathcal{DM}_k[H]$ whose defects avoid color i, i.e., they satisfy $\mathrm{usat}(H, M) \cap V_i = \emptyset$. Then $\mathcal{C} = \mathcal{DM}_k[H] \setminus \bigcup_{i \in [k]} \mathcal{A}_i$.

For $S \subseteq [k]$, write $\mathcal{A}_S = \bigcap_{i \in S} \mathcal{A}_i$ and note that we can compute $\#\mathcal{A}_S$ by an oracle call to #RestrDefectMatch on the instance $(H, \bigcup_{i \in S} V_i, k)$. We can hence compute $\#\mathcal{C} = \#\mathcal{PM}[G]$ via inclusion-exclusion (Lemma 5) and 2^k oracle calls to #RestrDefectMatch. ◀

3.2 From #RestrDefectMatch to #PlanarDefectMatch

For the second reduction in (4), we wish to solve instances (G, S, k) to #RestrDefectMatch when given only an oracle for counting k-defect matchings in planar graphs, *without* the ability of specifying the set S. Let G, S and k be fixed in the following. Our reduction involves manipulations on polynomials, such as a truncated version of polynomial division:

▶ **Lemma 10.** *Let X be an indeterminate, and let $p, q \in \mathbb{Z}[X]$ be polynomials $p = \sum_{i=0}^{m} b_i X^i$ and $q = \sum_{i=0}^{n} a_i X^i$ with $a_0 \neq 0$. For all $t \in \mathbb{N}$, we can compute $b_0, \dots, b_t$ with $\mathcal{O}(t^2)$ arithmetic operations from $a_0, \dots, a_t$ and the first $t+1$ coefficients of the product pq.*

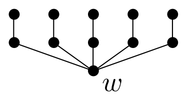
Proof. Let $c_0, \dots, c_{n+m}$ enumerate the coefficients of the product pq. By elementary algebra, we have $c_i = \sum_{\kappa=0}^{i} a_\kappa b_{i-\kappa}$, which implies the linear system

$$\begin{pmatrix} a_0 & & \\ \vdots & \ddots & \\ a_t & \dots & a_0 \end{pmatrix} \begin{pmatrix} b_0 \\ \vdots \\ b_t \end{pmatrix} = \begin{pmatrix} c_0 \\ \vdots \\ c_t \end{pmatrix}. \tag{5}$$

As this system is triangular with $a_0 \neq 0$ on its main diagonal, it has full rank and can be solved uniquely for $b_0, \dots, b_t$ with $\mathcal{O}(t^2)$ arithmetic operations. ◀

Our proof also relies upon a gadget which will allow to distinguish S from $V(G) \setminus S$.

▶ **Definition 11.** For $\ell \in \mathbb{N}$, an ℓ-rake R_ℓ is a matching M of size ℓ, together with an additional vertex w adjacent to one vertex of each edge in M:



Let $G_{S,\ell}$ be the graph obtained from attaching R_ℓ to each $v \in S$. This means adding a local copy of R_ℓ to v and identifying the copy of w with v. Please note that vertices $v \in V(G) \setminus S$ receive no attachments in $G_{S,\ell}$.

It is obvious that $G_{S,\ell}$ is planar if G is. Recall the defect-generating matching polynomial μ from (3). We first show that, for fixed $\ell \in \mathbb{N}$, the polynomial $\mu(G_{S,\ell})$ can be written as a weighted sum over matchings $M \in \mathcal{M}[G]$, where each M is weighted by an expression that depends on the number $|\mathrm{usat}(M) \cap S|$. Ultimately, we want to tweak these weights in such a way that only matchings with $|\mathrm{usat}(M) \cap S| = 0$ are counted.

▶ **Lemma 12.** *Define polynomials $r, f_\ell \in \mathbb{Z}[X]$ and $s \in \mathbb{Z}[X, \ell]$ by*

$$r(X) = 1 + X^2, \qquad s(X, \ell) = \ell + 1 + X^2, \qquad f_\ell(X) = (1 + X^2)^{|S|(\ell-1)}.$$

Then it holds that

$$\mu(G_{S,\ell}, X) = f_\ell \cdot \sum_{M \in \mathcal{M}[G]} X^{|\mathrm{usat}(M)|} \cdot r^{|S \setminus \mathrm{usat}(M)|} \cdot s^{|S \cap \mathrm{usat}(M)|}. \tag{6}$$

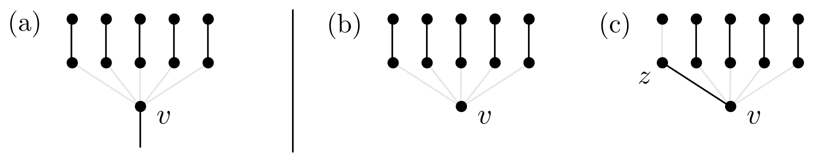


Figure 1 Possible types of extensions of the rake at v. The left case corresponds to $v \notin \mathrm{usat}(M)$, and the two right cases correspond to $v \in \mathrm{usat}(M)$.

Proof. Every matching $M \in \mathcal{M}[G]$ induces a certain set $\mathcal{C}_M \subseteq \mathcal{M}[G_{S,\ell}]$ of matchings in $G_{S,\ell}$, where each matching $N \in \mathcal{C}_M$ consists of M together with an extension by rake edges. The family $\{\mathcal{C}_M\}_{M\in\mathcal{M}[G]}$ is easily seen to partition $\mathcal{M}[G_{S,\ell}]$, and we obtain

$$\mu(G_{S,\ell}, X) = \sum_{M\in\mathcal{M}[G]} \underbrace{\sum_{N\in\mathcal{C}_M} X^{|\mathrm{usat}(N)|}}_{=:e(M)}. \tag{7}$$

Every matching $N \in \mathcal{C}_M$ consists of M and rake edges, which are added independently at each vertex $v \in S$. Hence, the expression $e(M)$ in (6) can be computed from the product of the individual extensions at each $v \in S$. To calculate the factor obtained by such an extension, we have to distinguish whether v is unmatched in M or not. The possible extensions at v are also shown in Figure 1.

$v \notin \mathrm{usat}(M)$: We can extend M at v by any subset of the ℓ rake edges not adjacent to v, as shown in Figure 1.a. In total, these 2^ℓ extensions contribute the factor $(1 + X^2)^\ell = (1 + X^2)^{\ell-1}r$.

$v \in \mathrm{usat}(M)$: We have two choices for extending, shown in the right part of Figure 1: Firstly, we can extend as in the case $v \notin \mathrm{usat}(M)$, and then we obtain the factor $X(1 + X^2)^\ell$. Here, the additional factor X corresponds to the unmatched vertex v. This situation is shown in Figure 1.b. Secondly, we can match v to one of its ℓ incident rake edges, say to $e = vz$ for a rake vertex z, as in Figure 1.c. Then we can choose a matching among the $\ell - 1$ rake edges not incident with z. This gives a factor of $\ell X(1 + X^2)^{\ell-1}$. Note that v is matched, but the vertex adjacent to z is not, yielding a factor of X.
In total, if $v \in \mathrm{usat}(M)$, we obtain the factor $X(1+X^2)^\ell+\ell X(1+X^2)^{\ell-1} = X(1+X^2)^{\ell-1}s$.

In each matching $N \in \mathcal{C}_M$, every unmatched vertex in $\bar{S} = V(G) \setminus S$ contributes a factor X. By multiplying the contributions of all $v \in V(G)$, we have thus shown that

$$\begin{aligned} e(M) &= f_\ell(X) \cdot X^{|\bar{S}\cap\mathrm{usat}(M)|} \cdot r^{|S\setminus\mathrm{usat}(M)|} \cdot (Xs)^{|S\cap\mathrm{usat}(M)|} \\ &= f_\ell(X) \cdot X^{|\mathrm{usat}(M)|} \cdot r^{|S\setminus\mathrm{usat}(M)|} \cdot s^{|S\cap\mathrm{usat}(M)|} \end{aligned}$$

and together with (7), this proves the claim. ◀

Due to the factor f_ℓ, the expression $\mu(G_{S,\ell})$ is not a polynomial in the indeterminates X and ℓ. We define a polynomial $p \in \mathbb{Z}[X, \ell]$ by removing this factor.

$$p(X, \ell) := \sum_{M\in\mathcal{M}[G]} X^{|\mathrm{usat}(M)|} \cdot r^{|S\setminus\mathrm{usat}(M)|} \cdot s^{|S\cap\mathrm{usat}(M)|}. \tag{8}$$

Depending upon the concrete application, we will consider $p \in \mathbb{Z}[X, \ell]$ as a polynomial in the indeterminates ℓ and X, or as a polynomial $p \in (\mathbb{Z}[\ell])[X]$ in the indeterminate X with coefficients from $\mathbb{Z}[\ell]$. In this last case, we write $p = \sum_{i=0}^{n} a_i X^i$ with coefficients $a_i \in \mathbb{Z}[\ell]$ for $i \in \mathbb{N}$ that are in turn polynomials. Then we define

$$[p]_k := \sum_{i=0}^{k} a_i X^i \tag{9}$$

as the restriction of p to its first $k+1$ coefficients. For later use, let us observe the following simple fact about $[p]_k$, considered as a polynomial $[p]_k \in \mathbb{Z}[X, \ell]$.

▶ **Fact 13.** *For $i, j \in \mathbb{N}$, every monomial $\ell^i X^j$ appearing in $[p]_k$ satisfies $i \le j \le k$.*

Proof. Recall r and s from Lemma 12. The indeterminate ℓ appears in s with degree 1, but it does not appear in r. In the right-hand side of (8), every term containing a factor s^t, for $t \in \mathbb{N}$, also contains the factor X^t, because $|S \cap \mathrm{usat}(M)| \le |\mathrm{usat}(M)|$ trivially holds. Hence, whenever $\ell^i X^j$ is a monomial in p, then $i \le j$. Since the maximum degree of X in $[p]_k$ is k by definition, the claim follows. ◀

In the next lemma, we show that knowing the coefficients of $[p]_k$ allows to solve the instance (G, S, k) to #RestrDefectMatch from the beginning of this subsection. After that, we will show how to compute $[p]_k$ with an oracle for #PlanarDefectMatch.

▶ **Lemma 14.** *Let $\mathcal{N}$ denote the set of (not necessarily k-defect) matchings in G with $\mathrm{usat}(M) \cap S = \emptyset$. For all $k \in \mathbb{N}$, we can compute the number of k-defect matchings in $\mathcal{N}$ in polynomial time when given the coefficients of $[p]_k$.*

Proof. For ease of presentation, assume first we knew *all* coefficients of p rather than only those of $[p]_k$. We will later show how to solve the problem when given only $[p]_k$.

Starting from p, we perform the substitution

$$\ell \leftarrow -(1 + X^2) \tag{10}$$

to obtain a new polynomial $q \in \mathbb{Z}[X]$ from p. By definition of s (see Lemma 12), we have

$$s(X, -(1 + X^2)) = 0, \tag{11}$$

so every matching $M \notin \mathcal{N}$ has zero weight in q. To see this, note that by (8), the weight of each matching $M \in \mathcal{M}[G]$ in p contains a factor $s^{|S \cap \mathrm{usat}(M)|}$. But due to (11), the corresponding term in q is non-zero only if $|S \cap \mathrm{usat}(M)| = 0$. We obtain

$$q = \sum_{M \in \mathcal{N}} X^{|\mathrm{usat}(M)|} \cdot (1 + X^2)^{|S \setminus \mathrm{usat}(M)|}.$$

Since every $M \in \mathcal{N}$ satisfies $|S \setminus \mathrm{usat}(M)| = |S|$, this simplifies to

$$q = (1 + X^2)^{|S|} \cdot \underbrace{\sum_{M \in \mathcal{N}} X^{|\mathrm{usat}(M)|}}_{=:q'} \tag{12}$$

and we can use standard polynomial division by $(1 + X^2)^{|S|}$ to obtain

$$q' = q/(1 + X^2)^{|S|}. \tag{13}$$

By (12), for all $k \in \mathbb{N}$, the coefficient of X^k in q' counts precisely the k-defect matchings in $\mathcal{N}$. This finishes the discussion of the idealized setting when all coefficients of p are known.

Recall the three steps involved: The substitution in (10), the polynomial division in (13), and the extraction of the coefficient X^k from q'.

The full claim, when only $[p]_k$ rather than p is given, can be shown similarly, but some additional care has to be taken. First, we perform the substitution (10) on $[p]_k$ rather than p. This results in a polynomial $b \in \mathbb{Z}[X]$, for which we claim the following:

▶ **Claim 15.** *We have* $[b]_k = [q]_k$.

Proof. Let $\Theta_{\leq i}$ for $i \in \mathbb{N}$ denote the set of monomials in p with degree $\leq i$ in X. The substitution (10) maps every monomial θ in the indeterminates X and ℓ to some polynomial $g_\theta \in \mathbb{Z}[X]$. Writing $a(\theta) \in \mathbb{Z}$ for the coefficient of θ in p, we obtain $q, b \in \mathbb{Z}[X]$ with

$$q = \sum_{\theta \in \Theta_{\leq n}} a(\theta) \cdot g_\theta, \tag{14}$$

$$b = \sum_{\theta \in \Theta_{\leq k}} a(\theta) \cdot g_\theta. \tag{15}$$

We can conclude that

$$[q]_k \underset{(14)}{=} \left[\sum_{\theta \in \Theta_{\leq n}} a(\theta) \cdot g_\theta \right]_k = \left[\sum_{\theta \in \Theta_{\leq k}} a(\theta) \cdot g_\theta \right]_k \underset{(15)}{=} [b]_k \,, \tag{16}$$

where the second identity holds since, whenever θ has degree i in X, for $i \in \mathbb{N}$, then g_θ contains a factor X^i. Hence, for $\theta \in \Theta_{\leq n} \setminus \Theta_{\leq k}$, no terms of the polynomial g_θ appear in $\left[\sum_{\theta \in \Theta_{\leq n}} a(\theta) \cdot g_\theta\right]_k$. This proves the claim. ◀

Recall the polynomial q' from (13); it remains to apply polynomial division as in (13) to recover $[q']_k$ from $[b]_k$. To this end, we observe that the constant coefficient in $(1+X^2)^{|S|}$ is 1, and that all coefficients of $(1+X^2)^{|S|}$ can be computed by a closed formula. We can thus divide $[b]_k = [q]_k$ by $[(1+X^2)^{|S|}]_k$ via truncated polynomial division (Lemma 10) to obtain $[q']_k$, whose k-th coefficient counts the k-defect matchings in $\mathcal{N}$, as in the idealized setting discussed before. ◀

Using a combination of truncated polynomial division (Lemma 10) and interpolation, we compute the coefficients of $[p]_k$ with oracle access for #PlanarDefectMatch. This completes the reduction from #RestrDefectMatch to #PlanarDefectMatch.

▶ **Lemma 16.** *We can compute* $[p]_k$ *by a Turing fpt-reduction to* #PlanarDefectMatch *such that all queries have maximum parameter* k.

Proof. For ξ with $0 \leq \xi \leq k$, let $f_\xi \in \mathbb{Z}[X]$ be the evaluation of the expression f_ℓ defined in Lemma 12 at $\ell = \xi$. Define $p_\xi^{(k)} \in \mathbb{Z}[X]$ by

$$p_\xi^{(k)} := [\mu(G_{S,\xi})/f_\xi]_k \,. \tag{17}$$

▶ **Claim 17.** *We have* $p_\xi^{(k)} = [p(\cdot,\xi)]_k = [p]_k(\cdot,\xi)$.

Proof. The first identity holds by the definition of p in (8), and by the definition of $p_\xi^{(k)}$. The second identity holds because, for all $t \in \mathbb{N}$, the coefficient of X^t in p is a polynomial in ℓ and does not depend on X. Hence we may arbitrarily interchange (i) the operation of substituting ℓ by expressions not depending on X (and by numbers $\xi \in \mathbb{N}$ in particular), and (ii) the operation of truncating to the first k coefficients. ◀

Recall that $a_t \in \mathbb{Z}[\ell]$ for $t \in \mathbb{N}$ denotes the coefficient of X^t in p, which has degree at most k (in the indeterminate ℓ) by Fact 13. Hence, for fixed $t \in \mathbb{N}$, if we knew the values $a_t(0), \ldots, a_t(k)$, we could recover the coefficients of $a_t \in \mathbb{Z}[\ell]$ via univariate polynomial interpolation. But for $0 \leq \xi, t \leq k$, we can obtain the value $a_t(\xi)$ as the coefficient of X^t in $p_\xi^{(k)}$. This follows from Claim 17. It remains to compute the polynomials $p_0^{(k)}, \ldots, p_k^{(k)}$ with an oracle for #PlanarDefectMatch: First, we observe that the constant coefficient in f_ξ is 1 for all $0 \leq \xi \leq k$, so we can apply the definition of $p_\xi^{(k)}$ from (17) and truncated polynomial division (Lemma 10) to compute $p_\xi^{(k)}$ from $[\mu(G_{S,\xi})]_k$ and f_ξ.

It remains only to compute $[\mu(G_{S,\xi})]_k$ and f_ξ. Note that the coefficients of f_ξ admit a closed expression by definition, and that $[\mu(G_{S,\xi})]_k$ can be computed by querying the oracle for #PlanarDefectMatch to obtain the number of matchings in $G_{S,\xi}$ with $0, \ldots, k$ defects. ◀

We recapitulate the proof of Theorem 1 in the following.

Proof of Theorem 1. By Theorem 7, the problem #ApexPerfMatch is #W[1]-hard, and we have reduced it to #RestrDefectMatch in Lemma 9. By Lemma 16, we can use oracle calls to #PlanarDefectMatch with maximum parameter k to compute the polynomial $[p]_k$, and by Lemma 14, the coefficients of $[p]_k$ allow to recover the solution to #RestrDefectMatch in polynomial time. These two steps establish the second reduction in (4).

Note that both reductions incur only linear blowup on the parameter. Hence, the lower bound of $n^{\Omega(k/\log k)}$ for #ApexPerfMatch under #ETH from Theorem 7 carries over to #PlanarDefectMatch. ◀

4 Apices with few adjacent faces

We prove Theorem 2: We present an FPT-algorithm for a restricted version of the problem #PerfMatch on graphs G with an apex set A of size k such that every apex can see only a bounded number of faces. To this end, we first prove a stronger version of Theorem 3 that allows us to compute #MatchSum(G) rather than just count matchings in G.

▶ **Theorem 18.** *Assume we are given a drawing of a planar graph G with vertex-weights $w : V(G) \to \mathbb{Q}$ and faces $F_1, \ldots, F_s$ for $s \in \mathbb{N}$ such that all vertices $v \in V(G)$ with $w(v) \neq 0$ satisfy $v \in V(F_1) \cup \ldots \cup V(F_s)$. Then we can compute #MatchSum(G) in time $\mathcal{O}(2^s \cdot n^3)$.*

Proof. We first create a partition $B_1, \ldots, B_s$ of $\bigcup_{i \in [s]} V(F_i)$ such that $B_i \subseteq F_i$ for $i \in [s]$ and $B_i \cap B_j = \emptyset$ for $i \neq j$. This can be achieved trivially by assigning each vertex that occurs in several faces F_i to some arbitrarily chosen set B_i.

Now we define a type $\theta_M \in \{0,1\}^s$ for each $M \in \mathcal{M}[G]$. For $i \in [s]$, we define

$$\theta_M(i) := \begin{cases} 1 & |\mathrm{usat}(M) \cap B_i| \text{ odd}, \\ 0 & |\mathrm{usat}(M) \cap B_i| \text{ even}. \end{cases}$$

For $\theta \in \{0,1\}^s$, let $\mathcal{M}_\theta[G]$ denote the set of matchings $M \in \mathcal{M}[G]$ with $\theta_M = \theta$, and define

$$S_\theta = \sum_{M \in \mathcal{M}_\theta[G]} \prod_{v \in \mathrm{usat}(M)} w(v).$$

It is clear that #MatchSum$(G) = \sum_{\theta \in \{0,1\}^s} S_\theta$. We show how to compute S_θ for fixed θ in time $\mathcal{O}(n^3)$ by reduction to #PerfMatch in planar graphs. For this argument, we momentarily define $\#\mathsf{MatchSum}(G)$ on graphs that have vertex- and edge-weights $w : V(G) \cup E(G) \to \mathbb{Q}$:

$$\#\mathsf{MatchSum}(G) = \sum_{M \in \mathcal{M}[G]} \left(\prod_{v \in \mathrm{usat}(M)} w(v) \right) \left(\prod_{e \in M} w(e) \right).$$

As shown in the proof of Theorem 3.3 in [36], and in Example 15 in [9], for every $t \in \mathbb{N}$, there exist explicit planar graphs D_t^0 and D_t^1 with $\mathcal{O}(t)$ vertices, which contain special vertices $u_1, \dots, u_t$ such that all of the following holds:

1. The graphs D_t^0 and D_t^1 can be drawn in the plane with $u_1, \dots, u_t$ on their outer faces.
2. Let H be a vertex- and edge-weighted graph with distinct vertices $X = \{v_1, \dots, v_t\} \subseteq V(H)$ and let H' be obtained from H by placing a disjoint copy of D_t^0 into H and connecting v_i to u_i with an edge of weight $w(v_i)$ for all $i \in [t]$. Assign weight 0 to the vertices v_i and to all vertices of D_t^0. Then
$$\#\mathsf{MatchSum}(H') = \sum_{\substack{M \in \mathcal{M}[H] \\ |\mathrm{usat}(M) \cap X| \text{ even}}} \left(\prod_{v \in \mathrm{usat}(M)} w(v) \right) \left(\prod_{e \in M} w(e) \right) \tag{18}$$
3. The above statement also applies for D_t^1, but the corresponding sum in (18) ranges over those $M \in \mathcal{M}[H]$ where $|\mathrm{usat}(M) \cap X|$ is odd rather than even.

We observe that inserting D_t^0 or D_t^1 into the face of a planar graph preserves planarity. Hence, we can insert $D_{|B_i|}^{\theta(i)}$ at the vertices B_i along face F_i in G, for each $i \in [s]$, and obtain a planar graph G_θ. By construction, we have $\#\mathsf{MatchSum}(G_\theta) = S_\theta$. Furthermore, all vertex-weights in G_θ are 0 by construction, so we actually have $\#\mathsf{MatchSum}(G_\theta) = \#\mathsf{PerfMatch}(G_\theta)$. Since G_θ is planar, we can evaluate $\#\mathsf{PerfMatch}(G_\theta)$ in time $\mathcal{O}(n^3)$, thus concluding the proof. ◀

Note that the above theorem allows us to recover the number of k-defect matchings in G that have all defects on fixed distinguished faces, for any $k \in \mathbb{N}$: Let G_X be obtained from G by assigning weight X to each vertex. Then $p := \#\mathsf{MatchSum}(G_X)$ is a polynomial of degree at most n and can be interpolated from evaluations $p(0), \dots p(n)$, but each of these evaluations can be computed in time $\mathcal{O}(2^s \cdot n^3)$ by Theorem 18. As we know, the k-th coefficient of $p(X)$ is equal to the number of k-defect matchings in G.

In the following, we extend this argument by using a variant of multivariate polynomial interpolation (Lemma 6) that applies when we do not require the values of *all* coefficients, but rather only those in a "slice" of total degree k, for fixed $k \in \mathbb{N}$. Here, the polynomial p to be interpolated features a distinguished indeterminate X, and we wish to extract the coefficient a_k of X^k, which is in turn a polynomial. Under certain restrictions, this can be achieved with $f(k) \cdot n$ evaluations, where n denotes the degree of X in p.

▶ **Lemma 19.** *Let $p \in \mathbb{Z}[X, \lambda]$ be a multivariate polynomial in the indeterminates X and $\lambda = (\lambda_1, \dots, \lambda_t)$. Consider $p \in (\mathbb{Z}[\lambda])[X]$ and assume that p has degree n in X, and that for all $s \in \mathbb{N}$, the coefficient $a_s \in \mathbb{Z}[\lambda]$ of X^s in p has total degree at most s. Let $k \in \mathbb{N}$ be a given parameter, and let $\Xi = \Xi_0 \times \ldots \times \Xi_t \subseteq \mathbb{Q}^{t+1}$ with $|\Xi_0| = n + 1$ and $|\Xi_i| = k + 1$ for all $i > 0$. Then we can compute the coefficients of the polynomial $a_k \in \mathbb{Z}[\lambda]$ with $\mathcal{O}(|\Xi|^3)$ arithmetic operations when given as input the set $\{(\xi, p(\xi)) \mid \xi \in \Xi\}$.*

Proof. We consider the grid Ξ' defined by removing the first component from Ξ, that is, $\Xi' = \Xi_1 \times \ldots \times \Xi_t$. Observe that $p(\cdot, \xi') \in \mathbb{Z}[X]$ holds for $\xi' \in \Xi'$. Write $\Xi_0 = \{c_0, \dots, c_n\}$ and note that, for fixed $\xi' \in \Xi'$, our input contains all evaluations

$$p(c_0, \xi'), \dots, p(c_n, \xi'),$$

so we can use univariate interpolation to determine the coefficient of X^k in $p(\cdot, \xi')$. This coefficient is equal to $a_k(\xi')$ by definition. By performing this process for all $\xi' \in \Xi'$, we can evaluate $a_k(\xi')$ on all $\xi' \in \Xi'$, and hence interpolate the polynomial $a_k \in \mathbb{Z}[\lambda]$ via grid interpolation (Lemma 6). ◀

This brings us closer to the proof of Theorem 2. To proceed, we first consider the case that A is an independent set; the full algorithm is obtained by reduction to this case.

▶ **Lemma 20.** *Let G be an edge-weighted graph, given as input together with an independent set $A \subseteq V(G)$ of size k, a planar drawing of $H = G - A$, and faces $F_1, \dots, F_s$ that contain all neighbors of A. Then we can compute $\#\mathsf{PerfMatch}(G)$ in time $k^{\mathcal{O}(2^k)} \cdot 2^{\mathcal{O}(s)} \cdot n^4$.*

▶ Remark. We may assume that every edge $av \in E(G)$ with $a \in A$ and $v \in V(G) \setminus A$ has weight 1: Otherwise, replace av by a path ar_1r_2v with fresh vertices r_1, r_2, together with edges ar_1 and r_1r_2 of unit weight, and an edge r_2v of weight $w(e)$. This clearly preserves the apex number, the value of $\#\mathsf{PerfMatch}$, and ensures that every apex is only incident with unweighted edges.

Proof. Recall that $\mathcal{DM}_k[H]$ denotes the set of k-defect matchings in H. By Remark 4, we can assume that all edges incident with A have unit weight. Let

$$\mathcal{C} = \{M \in \mathcal{DM}_k[H] \mid \mathrm{usat}(M) \subseteq N_G(A)\}.$$

Given any matching $M \in \mathcal{C}$, let $t(M)$ denote its *type*[3], which is defined as the following *multiset* with precisely k elements from 2^A:

$$t(M) = \{N_G(v) \cap A \mid v \in \mathrm{usat}(M)\}.$$

For the set of all such types, we write $\mathcal{T} = \{t(M) \mid M \in \mathcal{C}\}$ and observe that $|\mathcal{T}| \leq (2^k)^k = 2^{k^2}$. For $t \in \mathcal{T}$, define a graph S_t as follows: Create an independent set $[k]$, corresponding to A. Then, for each $N \in t$, create a vertex v_N that is adjacent to all of $N \subseteq [k]$. We note that every *perfect* matching $M \in \mathcal{PM}[G]$ can be decomposed uniquely as $M = B(M) \dot\cup I(M)$ with a k-defect matching $B(M) \in \mathcal{C}$ and a perfect matching $I(M) \in \mathcal{PM}[S_{t(B(M))}]$. That is, $B(M) = M - A$ and $I(M) = M[A \cup \mathrm{usat}(B(M))]$. For $t \in \mathcal{T}$, let

$$\begin{aligned} \mathcal{C}_t &= \{M \in \mathcal{C} \mid t(M) = t\}, \\ P_t &:= \sum_{N \in \mathcal{C}_t} \prod_{e \in N} w(e). \end{aligned}$$

It is clear that $\{\mathcal{C}_t\}_{t \in \mathcal{T}}$ partitions $\mathcal{C}$, and this implies

$$\#\mathsf{PerfMatch}(G) = \sum_{t \in \mathcal{T}} P_t \cdot \#\mathsf{PerfMatch}(S_t). \tag{19}$$

To see this, note that each perfect matching of type t can be obtained by extending some matching $M \in \mathcal{C}_t$ (all of which have k defects) by a perfect matching from $\mathrm{usat}(M)$ to A, which is precisely a perfect matching of S_t. Note that we require here that edges between $\mathrm{usat}(M)$ and A have unit weight, otherwise the graphs S_t would have to be edge-weighted as well and might no longer depend on t only, but would also have to incorporate the edge-weights of G.

[3] Please note that these types have no connection to those used in the proof of Theorem 18.

Since $|E(S_t)| \leq k^2$, we can compute $\#\mathsf{PerfMatch}(S_t)$ in time $2^{\mathcal{O}(k^2)}$ by brute force for each $t \in \mathcal{T}$. Hence, we can use (19) to determine $\#\mathsf{PerfMatch}(G)$ in time $|\mathcal{T}| \cdot 2^{\mathcal{O}(k^2)}$ if we know P_t for all $t \in \mathcal{T}$. In the remainder of this proof, we show how to compute P_t by using multivariate polynomial interpolation and the algorithm for $\#\mathsf{MatchSum}$ presented in Theorem 18. To this end, define indeterminates $\lambda = \{\lambda_R \mid R \subseteq A\}$ corresponding to subsets of the apices. Let X denote an additional distinguished indeterminate, and define the following polynomial $p \in \mathbb{Z}[X, \lambda]$. In this definition, we abbreviate $w(M) := \prod_{e \in M} w(e)$.

$$p(X, \lambda) := \sum_{M \in \mathcal{C}} w(M) \cdot X^{|\mathrm{usat}(M)|} \cdot \prod_{v \in \mathrm{usat}(M)} \lambda_{N_G(v) \cap A}. \tag{20}$$

For each type $t \in \mathcal{T}$, say $t = \{N_1, \ldots, N_k\}$, the coefficient of $X^k \cdot \lambda_{N_1} \cdot \ldots \cdot \lambda_{N_k}$ in p is equal to P_t. Hence, we can extract P_t for all $t \in \mathcal{T}$ from the coefficients of the monomials in p that have degree exactly k in X. Let us denote these monomials by $\mathfrak{N}$, and observe that each monomial $\nu \in \mathfrak{N}$ has total degree k in λ by the definition of p in (20).

If we can evaluate p on the elements (r, ξ) from the grid $\Xi = [n+1] \times [k+1]^{2^{|A|}}$, then we can compute the coefficients of all $\nu \in \mathfrak{N}$ in p, and thus P_t for all $t \in \mathcal{T}$, by sliced grid interpolation (Lemma 19). Note that $|\Xi| \leq \mathcal{O}(n \cdot k^{2^k})$. We compute these evaluations $p(r, \xi)$ as $p(r, \xi) = \#\mathsf{MatchSum}(H')$, where the vertex-weighted graph $H' = H'(r, \xi)$ is obtained from H via the weight function

$$w(v) := \begin{cases} 0 & \text{if } v \notin N_G(A), \\ r \cdot \xi_{N_G(v) \cap A} & \text{otherwise.} \end{cases}$$

Since all vertices with non-zero weight in H' are contained in the faces $F_1, \ldots, F_s$, we can compute $\#\mathsf{MatchSum}(H')$ in time $\mathcal{O}(2^s \cdot n^3)$ with Theorem 18. We obtain the values P_t for all $t \in \mathcal{T}$, so we obtain $\#\mathsf{PerfMatch}(G)$ via (19) in the required time. ◀

It remains to lift Lemma 20 to the case that A is not an independent set. This follows easily from the fact that, whenever $E(G) = E \dot\cup E'$, then every perfect matching $M \in \mathcal{PM}[G]$ must match every vertex $v \in V(G)$ into exactly one of the sets E or E'.

Proof of Theorem 2. Let $\mathcal{A} = \mathcal{M}[G[A]]$ denote the set of (not necessarily perfect) matchings of the induced subgraph $G[A]$, and note that $|\mathcal{A}| \leq 2^{k^2}$. For $M \in \mathcal{A}$, let $a_M = \#\mathsf{PerfMatch}(G_M)$, where G_M is defined by keeping from A only $\mathrm{usat}(M)$, and then deleting all edges between the remaining vertices of A. We can compute a_M by Lemma 20, since the remaining part of A in G_M is an independent set. It is also easily verified that $\#\mathsf{PerfMatch}(G) = \sum_{M \in \mathcal{A}} a_M \cdot \prod_{e \in M} w(e)$, so we can compute $\#\mathsf{PerfMatch}$ as a linear combination of 2^{k^2} values, each of which can be computed by Lemma 20. ◀

Acknowledgments. The author wishes to thank Dániel Marx and Holger Dell for pointing out the connection between perfect matchings in k-apex graphs and $\#\mathsf{PlanarDefectMatch}$ during the Dagstuhl Seminar 10481 on Computational Counting in 2010. Furthermore, thanks to Mingji Xia for interesting discussions about this topic. In particular, Theorem 3 was found in joint work on combined signatures back in 2013. Thanks also to Markus Bläser for reading earlier drafts of this material as it appeared in my PhD thesis, and thanks to the reviewers of this version for providing helpful comments.

References

1 Manindra Agrawal. Determinant versus permanent. In *Proceedings of the 25th International Congress of Mathematicians, ICM 2006*, volume 3, pages 985–997, 2006.

2 Markus Bläser and Radu Curticapean. Weighted counting of k-matchings is #W[1]-hard. In *IPEC*, pages 171–181, 2012. doi:10.1007/978-3-642-33293-7_17.

3 P. Bürgisser. *Completeness and Reduction in Algebraic Complexity Theory.* Number 7 in Algorithms and Computation in Mathematics. Springer Verlag, 2000. 168 + xii pp.

4 Jin-Yi Cai, Pinyan Lu, and Mingji Xia. A computational proof of complexity of some restricted counting problems. In *TAMC 2009*, pages 138–149, 2009.

5 Jianer Chen, Benny Chor, Mike Fellows, Xiuzhen Huang, David W. Juedes, Iyad A. Kanj, and Ge Xia. Tight lower bounds for certain parameterized NP-hard problems. *Inf. Comput.*, 201(2):216–231, 2005.

6 Radu Curticapean. Counting matchings of size k is #W[1]-hard. In *ICALP 2013*, pages 352–363, 2013. doi:10.1007/978-3-642-39206-1_30.

7 Radu Curticapean. Counting perfect matchings in graphs that exclude a single-crossing minor. *CoRR*, abs/1406.4056, 2014.

8 Radu Curticapean. Block interpolation: A framework for tight exponential-time counting complexity. In *ICALP 2015*, pages 380–392, 2015.

9 Radu Curticapean. Parity separation: A scientifically proven method for permanent weight loss. *CoRR*, abs/1511.07480, 2015.

10 Radu Curticapean. *The simple, little and slow things count: on parameterized counting complexity.* PhD thesis, Saarland University, 2015.

11 Radu Curticapean and Dániel Marx. Complexity of counting subgraphs: Only the boundedness of the vertex-cover number counts. In *FOCS 2014*, pages 130–139, 2014.

12 Radu Curticapean and Mingji Xia. Parameterizing the permanent: Genus, apices, minors, evaluation mod 2^k. In *FOCS 2015*, pages 994–1009, 2015.

13 Holger Dell, Thore Husfeldt, Dániel Marx, Nina Taslaman, and Martin Wahlen. Exponential time complexity of the permanent and the Tutte polynomial. *ACM Transactions on Algorithms*, 10(4):21, 2014.

14 Erik D. Demaine, MohammadTaghi Hajiaghayi, and Ken-ichi Kawarabayashi. Approximation algorithms via structural results for apex-minor-free graphs. In *ICALP 2009*, pages 316–327, 2009.

15 Jörg Flum and Martin Grohe. The parameterized complexity of counting problems. *SIAM Journal on Computing*, 33(4):892–922, 2004.

16 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory.* Springer, 2006.

17 Markus Frick. Generalized model-checking over locally tree-decomposable classes. *Theory Comput. Syst.*, 37(1):157–191, 2004.

18 Anna Galluccio and Martin Loebl. On the theory of Pfaffian orientations. I. Perfect matchings and permanents. *Electronic Journal of Combinatorics*, 6, 1998.

19 Frank Harary. *Graph theory.* Addison-Wesley, 1991.

20 Haruo Hosoya. Topological index. A newly proposed quantity characterizing the topological nature of structural isomers of saturated hydrocarbons. *Bulletin of the Chemical Society of Japan*, 44(9):2332–2339, 1971.

21 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

22 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. System Sci.*, 63(4):512–530, 2001.

23 Mark Jerrum. Two-dimensional monomer-dimer systems are computationally intractable. *Journal of Statistical Physics*, 48(1-2):121–134, 1987.

24 Mark Jerrum and Alistair Sinclair. Approximating the permanent. *SIAM J. Comput.*, 18(6):1149–1178, 1989.

25 Mark Jerrum, Alistair Sinclair, and Eric Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *J. ACM*, 51(4):671–697, 2004.

26 Pieter W. Kasteleyn. The statistics of dimers on a lattice: I. The number of dimer arrangements on a quadratic lattice. *Physica*, 27(12):1209–1225, 1961.

27 Pieter W. Kasteleyn. Graph Theory and Crystal Physics. In *Graph Theory and Theoretical Physics*, pages 43–110. Academic Press, 1967.

28 Charles Little. An extension of Kasteleyn's method of enumerating the 1-factors of planar graphs. In *Combinatorial Mathematics*, LNCS, pages 63–72. Springer, 1974.

29 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the Exponential Time Hypothesis. *Bulletin of the EATCS*, 84:41–71, 2011.

30 Tullio Regge and Riccardo Zecchina. Combinatorial and topological approach to the 3d ising model. *Journal of Physics A: Mathematical and General*, 33(4):741, 2000.

31 Neil Robertson and Paul D. Seymour. Graph minors. XVI. Excluding a non-planar graph. *J. Comb. Theory, Ser. B*, 89(1):43–76, 2003.

32 Simon Straub, Thomas Thierauf, and Fabian Wagner. Counting the number of perfect matchings in K_5-free graphs. In *CCC 2014*, pages 66–77, 2014.

33 H. N. V. Temperley and Michael E. Fisher. Dimer problem in statistical mechanics – an exact result. *Philosophical Magazine*, 6(68):1478–6435, 1961.

34 Leslie G. Valiant. The complexity of computing the permanent. *Theoret. Comput. Sci.*, 8(2):189–201, 1979.

35 Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.

36 Leslie G. Valiant. Holographic algorithms. *SIAM J. Comput.*, 37(5):1565–1594, 2008. doi:10.1137/070682575.

37 Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In *ESA 2009*, pages 566–577, 2009.

38 Vijay V. Vazirani. NC algorithms for computing the number of perfect matchings in $K_{3,3}$-free graphs and related problems. *Inf. Comput.*, 80(2):152–164, 1989.

On Interference Among Moving Sensors and Related Problems

Jean-Lou De Carufel[1], Matthew J. Katz[2], Matias Korman[*3], André van Renssen[4], Marcel Roeloffzen[5], and Shakhar Smorodinsky[†6]

1 **University of Ottawa, Ottawa, Canada**
`jdecaruf@uottawa.ca`
2 **Ben-Gurion University of the Negev, Beer-Sheva, Israel**
`matya@cs.bgu.ac.il`
3 **Tohoku University, Sendai, Japan**
`mati@dais.is.tohoku.ac.jp`
4 **National Institute of Informatics, Tokyo, Japan, and JST, ERATO, Kawarabayashi Large Graph Project**
`andre@nii.ac.jp`
5 **National Institute of Informatics, Tokyo, Japan, and JST, ERATO, Kawarabayashi Large Graph Project**
`marcel@nii.ac.jp`
6 **Ben-Gurion University of the Negev, Beer-Sheva, Israel, and École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland**
`shakhar@math.bgu.ac.il`

Abstract

We show that for any set of n moving points in $\mathbb{R}^d$ and any parameter $2 \leq k \leq n$, one can select a fixed non-empty subset of the points of size $O(k \log k)$, such that the Voronoi diagram of this subset is "balanced" at any given time (i.e., it contains $O(n/k)$ points per cell). We also show that the bound $O(k \log k)$ is near optimal even for the one dimensional case in which points move linearly in time. As an application, we show that one can assign communication radii to the sensors of a network of n moving sensors so that at any given time, their interference is $O(\sqrt{n \log n})$. This is optimal up to an $O(\sqrt{\log n})$ factor.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

Keywords and phrases Range spaces, Voronoi diagrams, moving points, facility location, interference minimization

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.34

1 Introduction

We consider the following kinetic facility location problem: given n clients (i.e., points) that are moving in $\mathbb{R}^d$ along simple trajectories and a parameter $k \leq n$, we wish to select few of them to become facilities to serve the remaining clients. We follow the usual assumption

[*] M. K. was supported in part by the ELC project (MEXT KAKENHI Nos. 12H00855 and 15H02665).
[†] S. S. was partially supported by Grant 1136/12 from the Israel Science Foundation and by the Swiss National Science Foundation Grants 200020144531 and 200021-137574.

that at any instant of time a client is served by its nearest facility. Our aim is to select the facilities so that none serves too many customers. Specifically, we wish to maintain the invariant that at any given time the number of clients served by each of the chosen facilities is bounded by n/k.

The pigeon-hole principle directly implies that we cannot select fewer than k facilities. Our main result is that a subset of size $O(k \log k)$ will suffice. We also show that one cannot improve this bound to $O(k)$, even for $d = 1$. As an application, we show how to construct a communication graph among a set of n moving sensors such that at any given time, the interference of the communication graph is bounded by $O(\sqrt{n \log n})$ (and its hop-diameter is three). Intuitively speaking, the interference of a sensor is the in-degree (i.e., the number of sensors that can communicate to him directly, see more details in Section 5). This bound is near optimal as already, in the static case, there are examples where the interference is at least $\Omega(\sqrt{n})$ [6].

In order to obtain our results we use the machinery of geometric hypergraphs and the theory of VC-dimension and ε-nets. By a geometric hypergraph (also called a range-space) we mean the following: suppose we are given a finite set P of points in $\mathbb{R}^d$ and a family of simple geometric regions, such as the family of all halfspaces in $\mathbb{R}^d$. Then we consider the combinatorial structure of the set system $(P, \{h \cap P\})$ where h is any halfspace. A key property of such hypergraphs is bounded VC-dimension (see Section 2 for exact definitions). In this paper we study a more general structure by allowing the underlying set of points to move along some "reasonable" trajectories (i.e., the coordinates of each point can be described with a polynomial function of bounded degree). Even though the static case is well-known, little research has been done for the case in which the points move. We show that those more complex hypergraphs, defined as the union of all hypergraphs obtained at all possible times, still have a bounded VC-dimension.

In addition to the above mentioned applications, we believe that the bounded VC-dimension of such hypergraphs is of independent interest and to the best of our knowledge has not been observed before. We hope that this paper will have many follow-up applications, since bounded VC-dimension has applications in many other areas of mathematics and computer science.

The paper is organized as follows: in Section 2 we introduce several key concepts as well as review known results that hold for static range spaces. In Section 3 we extend these results to the kinetic case. In Sections 4 and 5 we prove our main results concerning Voronoi diagrams for moving points and the interference problem mentioned above.

2 Preliminaries and Previous Work

A hypergraph $H = (V, \mathcal{E})$ is a pair of sets such that $\mathcal{E} \subseteq 2^V$. A geometric hypergraph is one that can be realized in a geometric way. For example, consider the hypergraph $H = (V, \mathcal{E})$, where V is a finite subset of $\mathbb{R}^d$ and $\mathcal{E}$ consists of all subsets of V that can be cut-off from V by intersecting it with a shape belonging to some family of "nice" geometric shapes, such as the family of all halfspaces. The elements of V are called *vertices*, and the elements of $\mathcal{E}$ are called *hyperedges*. For a subset $V' \subseteq V$, the hypergraph $H[V'] = (V', \{V' \cap S \colon S \in \mathcal{E}\})$ is the *sub-hypergraph* induced by V'.

We consider the following families of geometric hypergraphs: Let P be a set of points in $\mathbb{R}^2$ (or, in general, in $\mathbb{R}^d$) and let $\mathcal{R}$ be a family of regions in the same space. We refer to the hypergraph $H = (P, \{P \cap r \colon r \in \mathcal{R}\})$ as the hypergraph induced by P with respect to $\mathcal{R}$. When $\mathcal{R}$ is clear from the context, we sometimes refer to it as the hypergraph induced by P. In the literature, hypergraphs that are induced by points with respect to geometric regions of some specific kind are also referred to as *range spaces*. We sometimes abuse the notation and write $(P, \mathcal{R})$, instead of $H = (P, E)$, where $E = \{P \cap r : r \in \mathcal{R}\}$.

ε-nets and VC-dimension

A subset $T \subset V$ is called a *transversal* (or a *hitting set*) of a hypergraph $H = (V, \mathcal{E})$, if it intersects all sets of $\mathcal{E}$. The *transversal number* of H, denoted by $\tau(H)$, is the smallest possible cardinality of a transversal of H. The fundamental notion of a transversal of a hypergraph is central in many areas of combinatorics and its relatives. In computational geometry, there is a particular interest in transversals, since many geometric problems can be rephrased as questions on the transversal number of certain hypergraphs. An important special case arises when we are interested in finding a small size set $N \subset V$ that intersects all "relatively large" sets of $\mathcal{E}$. This is captured in the notion of an ε-net for a hypergraph:

▶ **Definition 1** (ε-net)**.** Let $H = (V, \mathcal{E})$ be a hypergraph with V finite. Let $\varepsilon \in [0, 1]$ be a real number. A set $N \subseteq V$ (not necessarily in $\mathcal{E}$) is called an *ε-net* for H if for every hyperedge $S \in \mathcal{E}$ with $|S| \geq \varepsilon|V|$ we have $S \cap N \neq \emptyset$.[1]

In other words, a set N is an ε-net for a hypergraph $H = (V, \mathcal{E})$ if it stabs all "large" hyperedges (i.e., those of cardinality at least $\varepsilon|V|$). The well-known result of Haussler and Welzl [7] provides a combinatorial condition on hypergraphs that guarantees the existence of small ε-nets (see below). This requires the following well-studied notion of the Vapnik-Chervonenkis dimension [16]:

▶ **Definition 2** (VC-dimension)**.** Let $H = (V, \mathcal{E})$ be a hypergraph. A subset $X \subset V$ (not necessarily in $\mathcal{E}$) is said to be *shattered* by H if $\{X \cap S : S \in \mathcal{E}\} = 2^X$. The *Vapnik-Chervonenkis dimension*, also denoted the VC-*dimension* of H, is the maximum size of a subset of V shattered by H.

Relation between ε-nets and the VC-dimension

Haussler and Welzl [7] proved the following fundamental theorem regarding the existence of small ε-nets for hypergraphs with small VC-dimension.

▶ **Theorem 3** (ε-net theorem)**.** *Let $H = (V, \mathcal{E})$ be a hypergraph with* VC*-dimension d. For every $\varepsilon \in (0, 1)$, there exists an ε-net $N \subset V$ with cardinality at most* $O\left(\frac{d}{\varepsilon} \log \frac{1}{\varepsilon}\right)$.

In fact, it can be shown that a random sample of vertices of size $O(\frac{d}{\varepsilon} \log \frac{1}{\varepsilon})$ is an ε-net for H with a positive constant probability (see [10] for details on how to compute such nets).

Many hypergraphs studied in computational geometry and learning theory have a "small" VC-dimension, where by "small" we mean a constant independent of the number of vertices of the underlying hypergraph. It is known that whenever range spaces are defined through semi-algebraic sets of constant description complexity (i.e., sets defined as a Boolean combination of a constant number of polynomial equations and inequalities of constant maximum degree), the resulting hypergraph has finite VC-dimension. Halfspaces, balls, boxes, etc. are examples of such sets; see, e.g., [11, 13] for more details.

[1] An analogous definition applies when V is not necessarily finite and H is endowed with a probability measure.

Thus, by Theorem 3, these hypergraphs admit "small" size ε-nets. Kómlos *et al.* [8] proved that the bound $O(\frac{d}{\varepsilon}\log\frac{1}{\varepsilon})$ on the size of an ε-net for hypergraphs with VC-dimension d is best possible. Namely, for a constant d, they construct a hypergraph H with VC-dimension d such that any ε-net for H must have size of at least $\Omega(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$. Recently, several breakthrough results provided better lower and upper bounds on the size of ε-nets in several special cases [1, 2, 14].

3 Kinetic hypergraphs

We start by extending the concept of geometric hypergraphs to the kinetic model. Let $P = \{p_1, \ldots, p_n\}$ denote a set of n moving points in $\mathbb{R}^d$, where each point is moving along some "simple" trajectory. That is, each p_i is a function $p_i : [0, \infty) \to \mathbb{R}^d$ of the form $p_i(t) = (x_1^i(t), \ldots, x_d^i(t))$, where $x_j^i(t)$ is a univariate polynomial $(1 \leq j \leq d)$. For a given real number $t \geq 0$ and a subset $P' \subset P$, we denote by $P'(t)$ the fixed set of points $\{p(t) \colon p \in P'\}$.

Let $\mathcal{R}$ be a (not necessarily finite) family of ranges; for example, the family of all half-spaces in $\mathbb{R}^d$. We define the *kinetic hypergraph* induced by $\mathcal{R}$:

▶ **Definition 4** (kinetic hypergraph). Let P be a set of moving points in $\mathbb{R}^d$ and let $\mathcal{R}$ be a family of ranges. Let $(P, \mathcal{E})$ denote the hypergraph where $\mathcal{E}$ consists of all subsets $P' \subseteq P$ for which there exists a time t and a range $r \in \mathcal{R}$ such that $P'(t) = P(t) \cap r$. We call $(P, \mathcal{E})$ the *kinetic hypergraph* induced by $\mathcal{R}$.

As in the static case we abuse the notation and denote the kinetic hypergraph by $(P, \mathcal{R})$. In order to apply our techniques, we need the following "bounded description complexity" assumption concerning the movement of the points of P. We say that a point $p_i = p_i(t) = (x_1^i(t), \ldots, x_d^i(t)) \in P$ moves with *description complexity* $s > 0$ if for each $1 \leq j \leq d$, the univariate polynomial $x_j^i(t)$ has degree at most s. In the remainder of this paper, we assume that $P(0)$ is in "general position". That is, at time $t = 0$ no $d+1$ points of $P(0)$ are on a common hyperplane. This assumption can be removed through usual symbolic perturbation techniques.

3.1 VC-Dimension of kinetic hypergraphs

In this section we prove that for many of the static range spaces that have small VC-dimension, their kinetic counterparts also have small VC-dimension. We start with the family $\mathcal{H}_d$ of all halfspaces in $\mathbb{R}^d$.

▶ **Theorem 5.** *Let $P \subset \mathbb{R}^d$ be a set of moving points with bounded description complexity s. Then, the kinetic-range space $(P, \mathcal{H}_d)$ has VC-dimension bounded by $O(d \log d)$.*

To prove Theorem 5, we need the following known definition and lemma (see, e.g., [11]). The *primal shatter function* of a hypergraph $H = (V, \mathcal{E})$ denoted by π_H is a function:

$$\pi_H : \{1, \ldots, |V|\} \to \mathbb{N}$$

defined by $\pi_H(i) = max_{V' \subseteq V, |V'|=i} |H[V']|$, where $|H[V']|$ denotes the number of hyperedges in the sub-hypergraph $H[V']$.

▶ **Lemma 6.** *Let $H = (V, \mathcal{E})$ be a hypergraph whose primal shatter function π_H satisfies $\pi_H(m) = O(m^c)$ for some constant $c \geq 2$. Then the VC-dimension of H is $O(c \log c)$.*

We provide a sketch of the proof of Lemma 6 for the sake of completeness.

Proof. Let d denote the VC-dimension of H, and let $V' \subseteq V$ be a shattered subset of cardinality d. On one hand it means that the number of possible subsets of V' that can be realized as the intersection of V' and a hyperedge in $\mathcal{E}$ is 2^d. On the other hand, by our assumption on π_H, for a subset of size d, there can be at most Ad^c hyperedges in the sub-hypergraph induced by it, for some appropriate constant A. In other words we have $2^d \leq \pi_H(d) \leq Ad^c$. This is easily seen to imply that $d = O(c \log c)$. Indeed, by choosing, say, $d = 10Ac \log c$, the above inequality does not hold, a contradiction. This completes the proof of the lemma. ◀

Proof of Theorem 5. By Lemma 6 it suffices to bound the primal shatter function $\pi_{\mathcal{H}_d}(m)$ by a polynomial of constant degree. It is a well known fact that the number of combinatorially distinct half-spaces determined by n (static) points in $\mathbb{R}^d$ is $O(n^d)$. This can be easily seen by charging hyperplanes to d-tuples of points (using rotations and translations) and observing that each tuple can be charged at most a constant (depending on the dimension d) number of times. Thus, at any given time, the number of hyperedges is bounded by $O(n^d)$. Next, note that as t varies, a combinatorial change in the hypergraph $(P(t), \mathcal{R})$ can occur only when $d+1$ points $p_1(t), \ldots, p_{d+1}(t)$ become affinely dependent. Indeed, a hyperedge is defined by a hyperplane that contains d points of $P(t)$, and that hyperedge changes when an additional point of $P(t)$ crosses the hyperplane (and thus $d+1$ points become affinely dependent). This happens if and only if the following determinant condition holds:

$$\begin{vmatrix} x_1^1(t) & x_2^1(t) & \cdots & x_d^1(t) & 1 \\ x_1^2(t) & x_2^2(t) & \cdots & x_d^2(t) & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_1^{d+1}(t) & x_2^{d+1}(t) & \cdots & x_d^{d+1}(t) & 1 \end{vmatrix} = 0 \tag{1}$$

where $x_i^j(t)$ denotes the i'th coordinate of $p_j(t)$. The left side of the equation is a univariate polynomial of degree at most ds. By our general position assumption this polynomial is not identically zero and thus can have at most ds solutions.

That is, a tuple of $d+1$ points of $P(t)$ generates at most ds events. Hence, the total number of such events is bounded by $O(\binom{n}{d+1}) = O(n^{d+1})$. Between any two events we have a fixed set of at most $O(n^d)$ distinct hyperedges, thus we can have $O(n^{2d+1})$ distinct hyperedges along all instants of time.

Since each hyperedge is defined by the points on its boundary, this property is hereditary. That is, for any subset $P' \subseteq P$ the hypergraph $H[P']$ has at most $O(|P'|^{2d+1})$ hyperedges. Thus, the shatter function satisfies $\pi_H(m) = O(m^{2d+1})$. Then by Lemma 6, $(P, \mathcal{H}_d)$ has bounded constant VC-dimension, where the constant depends only on d and s. ◀

Theorem 5 can be further generalized to arbitrary ranges with so-called bounded description complexity as defined below:

▶ **Theorem 7.** *Let $\mathcal{R}$ be a collection of semi-algebraic subsets of $\mathbb{R}^d$, each of which can be expressed as a Boolean combination of a constant number of polynomial equations and inequalities of maximum degree c (for some constant c). Let P be a set of moving points in $\mathbb{R}^d$ with bounded description complexity. Then the kinetic range-space $(P, \mathcal{R})$ has bounded* VC*-dimension.*

Proof. The proof combines Lemma 6 with Theorem 5 and the so-called Veronese lifting map from Algebraic Geometry. We omit the details as it is very similar to the proof for the static case. See, e.g., [11]. ◀

4 Balanced Voronoi cells for moving points

In this section we tackle the facility location problem for a set of moving clients, where the goal is to ensure a balanced division of the load among the facilities at any instance of time. Given a set P of moving points or *clients* in $\mathbb{R}^d$, locate a small number of the points to serve as *facilities* so that at every instance of time no facility is serving more than n/k clients. We make the usual assumption that each client goes to its nearest facility. In the following we show how to obtain an almost optimal balancing (up to a $\log k$ factor), even under the restriction that facilities may be located only at points of P.

▶ **Theorem 8.** *Let $P = \{p_1, \dots, p_n\}$ be any set of n moving points in $\mathbb{R}^d$ with bounded description complexity. For any integer $2 \le k \le n$, there exists a subset $N \subset P$ of cardinality $O(k \log k)$, such that for any time $t \ge 0$, each cell of the Voronoi diagram Vor$(N(t))$ contains at most $O(n/k)$ points of $P(t)$.*

Before proceeding with the proof of Theorem 8 we need the following result. An *infinite cone* with apex $a \in \mathbb{R}^d$ and angle $\theta \in \mathbb{R}$ is defined as the union of all halflines emanating from a whose orientations belong to some fixed cap of the sphere $\mathcal{S}^{d-1}$. Equivalently, it can be defined as the set:

$$\{x \in \mathbb{R}^d \colon (x-a) \cdot (b-a) \ge \|x-a\| \cos\theta\} \ ,$$

where "$\|\|$" denotes the Euclidean norm, " $\cdot$ " denotes the scalar product and b is a vector such that $\|b-a\| = 1$. A *bounded cone* is the intersection of an infinite cone with a ball centered at its apex.

▶ **Lemma 9.** *Let P be a set of moving points in $\mathbb{R}^d$ with bounded description complexity s, and let $\mathcal{R}$ be the family of all bounded cones. The kinetic hypergraph $(P, \mathcal{R})$ has bounded* VC*-dimension.*

Proof. As shown above, the boundary surface of an infinite cone is a quadric (i.e., a polynomial of degree 2). In particular, the ranges of $\mathcal{R}$ can be expressed as semi-algebraic sets of constant description complexity. Thus, by Theorem 7 the hypergraph $(P, \mathcal{R})$ has constant VC-dimension as claimed. ◀

Proof of Theorem 8. Let $\mathcal{W}$ be the family of all bounded cones in $\mathbb{R}^d$. Let $H = (P, \mathcal{W})$ be the corresponding kinetic hypergraph. By Lemma 9, H has constant VC-dimension.

We fix $\varepsilon = \frac{1}{k}$ and let $N \subset P$ be an ε-net for H of size $O(k \log k)$ (refer to Theorem 3). We show that N satisfies the desired property. That is, for any time $t \ge 0$ and point $q \in N$, the Voronoi cell of $q(t)$ in the Voronoi diagram Vor$(N(t))$ contains at most $O(n/k)$ points of $P(t)$. Let C_d be the minimum number of sixty-degree caps that are needed to cover the unit sphere $\mathcal{S}^{d-1}$. Using packing arguments it is easily seen that C_d is a constant that depends only on d; see, e.g., [3].

Assume to the contrary that the Voronoi cell of $q(t)$ contains a subset $P'(t) \subset P(t)$ of more than $C_d n/k$ points. By definition, each of the points in $P'(t)$ is closer to $q(t)$ than to any other point in $N(t)$. By the pigeonhole principle, there is an infinite sixty-degree cone W which has $q(t)$ as its apex and that contains at least $n/k + 1$ of the points of $P'(t)$. Sort the points of $P'(t) \cap W$ in increasing distance from $q(t)$; let $p_1(t), \dots, p_j(t)$ be the obtained order (note that by assumption, we have $j \ge n/k + 1$). Slightly perturb the cone W and bound it to obtain a bounded cone W' that contains the points $p_1(t), \dots, p_j(t)$ but does not contain $q(t)$ (or any other point of $P(t)$). This can always be done by choosing a sufficiently large radius, doing an infinitesimally small translation of the apex and (if necessary) changing the angle of the cone. Since N is an ε-net with respect to bounded cones, W' must contain a point $q'(t) \in N(t)$ (other than $q(t)$).

Since any point in $P(t) \cap W'$ also belongs to W, which is a cone of sixty degrees, any point $p(t) \in P(t) \cap W'$ for which $d(p(t), q(t)) \geq d(q'(t), q(t))$ must be closer to $q'(t)$ than to $q(t)$. In particular, $p_j(t)$ satisfies this inequality and thus belongs to the Voronoi cell of $q'(t)$ (and not of $q(t)$), which is a contradiction. ◀

In fact, a careful look at the proof above shows that the following stronger result holds:

▶ **Corollary 10.** *Let $N \subset P$, $|N| = O(k \log k)$, as in Theorem 8. Then, for any finite point set $S \subset \mathbb{R}^d$, and for any $t \geq 0$, the cell of any $q \in S$ in the Voronoi diagram $Vor(N(t) \cup S)$ contains at most $O(n/k)$ points of $P(t)$.*

Remark

We note that the bound of $O(k \log k)$ in Theorem 8 is near optimal. Clearly, if there are only $o(k)$ points in N then by the pigeonhole principle one of the Voronoi cells must contain $\omega(n/k)$ points of P. We also note that reducing the size of the set N seems to be out of reach and maybe impossible, even for the one dimensional case where the points move with constant speed. This follows from a recent lower-bound construction of Alon [1] for ε-nets for static hypergraphs consisting of points with respect to strips in the plane.

Indeed, assume that $d = 1$ and each point $p \in P$ is described with a linear equation of the form $p(t) = at + b$ (i.e., a line). Assume that there exists a subset $N \subset P$ such that for any $t > 0$ and $q \in N$, the Voronoi cell of $q(t)$ contains at most n/k points of $P(t)$. In particular, this implies that there are at most $2n/k$ points of $P(t)$ between any pair of consecutive points of $N(t)$. If we view the moving points in $\mathbb{R}$ as lines in $\mathbb{R}^2$, this is equivalent to choosing a subset of the lines with the property that any vertical segment (i.e., a range of the form $t_0 \times [c, d]$ for constants $t_0 > 0$, $c, d \in \mathbb{R}$) that intersects more than $2n/k$ of the above lines will also intersect one of the chosen lines. By standard point-line duality in two dimensions, this is equivalent to the problem of finding an $\varepsilon = \frac{2}{k}$-net for points with respect to strips in the plane, which still remains an open problem. Recently, Alon [1] gave a construction showing that such hypergraphs cannot have linear (in $\frac{1}{\varepsilon}$) size ε-nets. Since their problem can be reduced to ours, the same lower bound holds for our problem.

5 Low interference for moving transmitters

Here we show how to tackle the problem of minimizing interference among a set of wireless moving transmitters while keeping the number of topological changes of the underlying network subquadratic. In the following we define the concept of (receiver-based) *interference* of a set of ad-hoc sensors [17] (see Figure 1).

▶ **Definition 11.** Let $P = \{p_1, \ldots, p_n\}$ be a set of n points in $\mathbb{R}^d$ and let $r_1, \ldots, r_n$ be n non-negative reals representing power levels (or transmission radii) assigned to the points $p_1, \ldots, p_n$, respectively. Let $G = (P, E)$ be the graph associated with this power assignment, where $E = \{\{p, q\} \colon d(p, q) \leq min\{r_p, r_q\}\}$. That is, points p, q are neighbors in G if and only if p is contained in the ball centered at q with radius r_q and *vice versa*. Let $D = \{d_1, \ldots, d_n\}$ denote the set of balls where d_i is the ball centered at p_i and having radius r_i.

Let $I(D)$ denote the maximum depth of the arrangement of the balls in D. That is $I(D) = \max_{q \in \mathbb{R}^d} \{|\{d \in D \colon q \in d\}|\}$. We call $I(D)$ the *interference* of D, which is also the

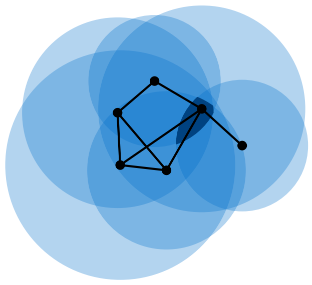


Figure 1 Given a set of fixed points in $\mathbb{R}^2$ and their power assignments represented by disks, the interference is the deepest point in the arrangement of the disks (the highlighted region in the figure). The underlying communication graph is shown with solid edges.

interference of the network. Note that both G and $I(D)$ are determined by P and $r_1, \ldots, r_n$. Given a set P of points in $\mathbb{R}^d$, the *interference* of P (denoted $I(P)$) is the smallest possible interference $I(D)$, where D corresponds to a power assignment whose associated graph is connected. The *interference minimization problem* asks for the power assignment for which $I(P) = I(D)$.

Empirically, (in dimension two) it has been observed that networks with high interference have high rates of message collision. This requires messages to be repeated often, which slows down the network and reduces battery life of the sensors [17]. Thus, a significant amount of research has focused in the creation of connected networks with low interference (see, e.g., [6, 9]). It is known that computing $I(P)$ (or even approximating it by a constant factor) is an NP-complete problem [4], but some worst-case bounds are known.

▶ **Theorem 12** ([6]). *Let P be a set of n points in the plane. Then $I(P) = O(\sqrt{n})$. Furthermore, this bound is asymptotically tight, in the sense that for any n there exists a set P of n points such that $I(P) = \Omega(\sqrt{n})$.*

Here, we turn our attention to the kinetic version of the interference problem in arbitrary but fixed dimension. We wish to maintain a connected graph of a set of moving points (representing moving sensors) that always has low interference. Unless the distances between sensors remain constant, no static radii assignment can work for a long period of time (since points will eventually be far from each other). Instead, we describe the network in a combinatorial way. That is, we look for a function $f : P \times [0, \infty) \to P$ that determines, for each sensor of P and instance of time, its furthest away sensor that must be reached. Then, at time t the communication radius of a sensor $p \in P$ is simply set equal to the distance between p and $f(p, t)$. Ideally, we would like to construct a network that not only has small interference at any instance of time, but also the underlying graph has a small amount of combinatorial changes along time.

Our algorithm to maintain a connected graph is based on the ideas used in [6] for the static case. We first pick a subset $N \subset P$ of "hubs". Those hubs will never change along time and will always have transmission radius big enough to cover all other points. Each other point in $P \setminus N$ will be assigned at every instance of time to its nearest hub. In the following we show that a careful choice of hubs will ensure a small interference, and overall small number of combinatorial changes in the radii assignment protocol. To bound the number of combinatorial changes, we need to use the machinery of Davenport-Schinzel sequences: A finite sequence $\sigma = (e_1, \ldots, e_m)$ over an alphabet of n symbols is called a *Davenport-Schinzel sequence* of order t when no two consecutive elements of σ are equal, and for any two distinct symbols x, y, there does not exist a subsequence where x and y alternate $t + 2$

times. Several bounds are known on the maximum length of Davenport-Schinzel sequences of a given order. In particular, we are interested in upper bounds. See [15] for more details on Davenport-Schinzel sequences.

▶ **Theorem 13** (Upper bound on Davenport-Schinzel sequences [12])**.** *A Davenport-Schinzel sequence of order t on n symbols has length at most $O(n2^{O(\alpha(n)^{\lfloor(t-2)/2\rfloor})})$, where $\alpha(n)$ is the inverse of the Ackermann function.*

The Ackermann function is a function that grows very rapidly, hence its inverse is usually regarded as a small constant (indeed, it is known that $\alpha(n) \leq 5$ for any input that can be stored explicitly in current computers). Davenport-Schinzel sequences are often used to bound the complexity of upper (or lower) envelopes of polynomial functions. Whenever we have a family of n functions such that no two graphs of those functions cross more than t times (for some bounded constant t), we can use Theorem 13 to bound the complexity of their upper and lower envelope.

▶ **Theorem 14.** *Let P be a set of n moving points in $\mathbb{R}^d$ with bounded description complexity s. Then, there is a power assignment with updates, such that at any given time t the interference of the network is at most $O(\sqrt{n \log n})$. Moreover, the total number of combinatorial changes in the network is at most $O^*(n^{1.5}\sqrt{\log n})$, where the O^* notation hides a term involving the inverse Ackermann function that depends on d and s.*

Proof. We use Theorem 8 for some value of k that will be determined later. We obtain a set N of size $O(k \log k)$ with the properties guaranteed by Theorem 8. The elements of N are called *hubs*, and we assign to each of them the largest possible radius. That is, at any instance of time $t \geq 0$, a point $p \in N$ is assigned the distance to its furthest point in P. In other words, $f(p,t)$ is equal to the point $q \in P$ that maximizes the distance $d(p(t), q(t))$. Other points of P are assigned the distance to their nearest hub. More formally, $f(p,t)$, for a point $p \in P \setminus N$, is equal to the point $q \in N$ that minimizes the distance $d(p(t), q(t))$. Equivalently, if we consider the Voronoi diagram with sites $N(t)$, the function $f(p,t)$ will match $p(t)$ with the site associated to the Voronoi cell that contains $p(t)$ at time t.

First observe that the network is connected: indeed, all hubs are connected to each other forming a clique. Moreover, each point of $P \setminus N$ has radius large enough to reach one point of N. In particular, any two points of P can reach each other after hopping through at most two intermediate sensors of N (thus, the constructed network has diameter 3).

We now pick the correct value of k so that the interference of this protocol is minimized. Since N has $O(k \log k)$ points, the overall interference contribution by hubs is bounded by the same amount. By Corollary 10, we also know that no point $q \in \mathbb{R}^d$ can be reached by more than $O(n/k)$ points of $P \setminus N$ at any instance of time. That is, the total interference of any point $q \in \mathbb{R}^d$ is at most $O(k \log k)$ from hubs, and at most $O(n/k)$ from non-hubs. Thus, by setting $k = \sqrt{n/\log n}$ we obtain the claimed bound.

We now bound the total number of combinatorial changes that will happen to the network along time. Let $p \in P$, we will show that the number of combinatorial changes of p is bounded. Recall that, if p is a hub it will connect to its furthest away point of P. Otherwise, p will connect to its nearest hub. In either case, it suffices to bound the number of combinatorial changes of the nearest/furthest point within a group of moving points with respect to the moving point p. Equivalently, we are looking at the number of combinatorial changes of the upper envelope of the family of functions $\mathcal{F}_1 = \{d(p(t), p'(t)) \colon p' \in P\}$ for points $p \in N$, or the lower envelope of the family of functions $\mathcal{F}_2 = \{d(p(t), p'(t)) \colon p' \in N\}$ for points $p \notin N$. By the bounded description complexity assumption, functions of $\mathcal{F}_1$ and

$\mathcal{F}_2$ are such that the graphs of any pair of them cross $O(s)$ times. Thus, by the Davenport-Schinzel Theorem we can bound the number of combinatorial changes of the upper envelope of $\mathcal{F}_1$ by $O(\lambda_{O(s)}(n))$, where $\lambda_t(m)$ denotes the maximum length of a Davenport-Schinzel sequence of order t on m symbols. Similarly, the number of changes of the lower envelope of $\mathcal{F}_2$ is bounded by $O(\lambda_{O(s)}(\sqrt{n \log n}))$.

Ignoring the terms that depend on the inverse of the Ackermann function, we have that for any fixed constant s, $\lambda_t(m) = O^*(m)$. Combining this with the fact that we have $O(\sqrt{n \log n})$ hubs and at most n non-hub points, the overall number of combinatorial changes is bounded by $O^*(n \times \sqrt{n \log n} + \sqrt{n \log n} \times n) = O^*(n^{1.5}\sqrt{\log n})$ as claimed. ◀

6 Conclusion

Using the the machinery of VC-dimension we have shown that the difference between static and kinetic environments for our facility location problem is small. We believe that a similar approach can be used for other problems. Some directly follow from Theorem 7 (such as kinetic range counting or discrepancy, see details in the extended version of this paper [5]). We hope that future research will show other interesting applications.

In Section 4 we argued that it is unlikely that the "balanced" property can be significantly improved. Similarly, it seems unlikely that the "reasonable" constraint can be removed, even in one dimension. Indeed, if points are allowed to move arbitrarily, they can create all $n!$ orderings along time. In particular, for any set $N \subset P$ we can always find a time and range that contains all points of $P \setminus N$. Thus, no subset $N \subset P$ can act as an ε-net for all instances of time. Further note that, since the alternation in orderings can be repeated arbitrarily many times, the number of times that we need to change the set N can also be unbounded. This behaviour can be created with trigonometric functions of low description complexity.

Acknowledgements. This work was initiated during the Second Sendai Winter Workshop on Discrete and Computational Geometry. The authors would like to thank the other participants for interesting discussions during the workshop, as well as Alexandre Rok for helpful discussions.

References

1 N. Alon. A non-linear lower bound for planar epsilon-nets. *Discrete & Computational Geometry*, 47(2):235–244, 2012. doi:10.1007/s00454-010-9323-7.

2 B. Aronov, E. Ezra, and M. Sharir. Small-size epsilon-nets for axis-parallel rectangles and boxes. *SIAM Journal on Computing*, 39(7):3248–3282, 2010.

3 K. Böröczky. Packing of spheres in spaces of constant curvature. *Acta Mathematica Academiae Scientiarum Hungarica*, 32(3-4):243–261, 1978. doi:10.1007/BF01902361.

4 Y. Brise, K. Buchin, D. Eversmann, M. Hoffmann, and W. Mulzer. Interference minimization in asymmetric sensor networks. In *ALGOSENSORS 2014*, pages 136–151, 2014. doi:10.1007/978-3-662-46018-4_9.

5 J.-L. De Carufel, M. Katz, M. Korman, A. van Renssen, M. Roeloffzen, and S. Smorodinsky. On kinetic range spaces and their applications. *CoRR*, abs/1507.02130, 2015. URL: http://arxiv.org/abs/1507.02130.

6 M.M. Halldórsson and T. Tokuyama. Minimizing interference of a wireless ad-hoc network in a plane. *Theoretical Computer Science*, 402(1):29–42, 2008. doi:10.1016/j.tcs.2008.03.003.

7 D. Haussler and E. Welzl. Epsilon-nets and simplex range queries. *Discrete & Computational Geometry*, 2:127–151, 1987.

8 J. Komlós, J. Pach, and G.J. Woeginger. Almost tight bounds for epsilon-nets. *Discrete & Computational Geometry*, 7:163–173, 1992.

9 M. Korman. Minimizing interference in ad-hoc networks with bounded communication radius. *Information Processing Letters*, 112(19):748–752, 2012. `doi:10.1016/j.ipl.2012.06.021`.

10 J. Matoušek. Approximations and optimal geometric divide-and-conquer. In *Symposium on the Theory of Computing*, pages 505–511, 1991.

11 J. Matoušek. *Lectures on Discrete Geometry.* Springer-Verlag New York, Inc., 2002.

12 G. Nivasch. Improved bounds and new techniques for Davenport–Schinzel sequences and their generalizations. *Journal of the ACM*, 57(3), 2010.

13 J. Pach and P. K. Agarwal. *Combinatorial Geometry.* Wiley Interscience, 1995.

14 J. Pach and G. Tardos. Tight lower bounds for the size of epsilon-nets. In *Symposium on Computational Geometry*, pages 458–463, 2011.

15 M. Sharir and P. K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications.* Cambridge University Press, New York, 1995.

16 V. N. Vapnik and A. Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971.

17 P. von Rickenbach, R. Wattenhofer, and A. Zollinger. Algorithmic models of interference in wireless ad hoc and sensor networks. *IEEE/ACM transactions on sensor networks*, 17(1):172–185, 2009. `doi:10.1109/TNET.2008.926506`.

SimBa: An Efficient Tool for Approximating Rips-Filtration Persistence via *Sim*plicial *Ba*tch-Collapse[*]

Tamal K. Dey[1], Dayu Shi[2], and Yusu Wang[3]

1 **Dept. of Computer Science and Engineering, and Dept. of Mathematics, The Ohio State University, Columbus, OH, USA**
`tamaldey@cse.ohio-state.edu`

2 **Dept. of Computer Science and Engineering, and Dept. of Mathematics, The Ohio State University, Columbus, OH, USA**
`shiday@cse.ohio-state.edu`

3 **Dept. of Computer Science and Engineering, and Dept. of Mathematics, The Ohio State University, Columbus, OH, USA**
`wang.1016@osu.edu`

Abstract

In topological data analysis, a point cloud data P extracted from a metric space is often analyzed by computing the persistence diagram or barcodes of a sequence of Rips complexes built on P indexed by a scale parameter. Unfortunately, even for input of moderate size, the size of the Rips complex may become prohibitively large as the scale parameter increases. Starting with the *Sparse Rips filtration* introduced by Sheehy, some existing methods aim to reduce the size of the complex so as to improve the time efficiency as well. However, as we demonstrate, existing approaches still fall short of scaling well, especially for high dimensional data. In this paper, we investigate the advantages and limitations of existing approaches. Based on insights gained from the experiments, we propose an efficient new algorithm, called *SimBa*, for approximating the persistent homology of Rips filtrations with quality guarantees. Our new algorithm leverages a batch collapse strategy as well as a new sparse Rips-like filtration. We experiment on a variety of low and high dimensional data sets. We show that our strategy presents a significant size reduction, and our algorithm for approximating Rips filtration persistence is order of magnitude faster than existing methods in practice.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Rips filtration, Homology groups, Persistence, Topological data analysis

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.35

1 Introduction

In recent years, topological ideas and methods have emerged as a new paradigm for analyzing complex data [8, 24]. An important line of work in this direction is the theory and applications of persistent homology. It provides a powerful and flexible framework to inspect data for characterizing and summarizing important features that persist across different scales. Since its introduction [25, 26, 34], there have been many fundamental developments [7, 9, 10, 12, 14, 17, 18, 19, 39] both to generalize the framework and to provide theoretical understanding

[*] This work has been supported by NSF grants CCF-1318595 and CCF-1526513.

for various aspects of it (such as its stability). These developments help to provide foundation and justification of the practical usage of persistent homology; see e.g, [13, 15, 22, 37, 33].

A determining factor in applying persistent homology to a broad range of data analysis problems is the availability of efficient and scalable software. Given the rapidly increasing size of modern data, the "efficiency" necessarily concerns both time and space complexities. The original algorithm to compute persistent homology takes $O(n^3)$ time and $O(n^2)$ space for a filtration involving n number of total simplices [25]. Various practical improvements have been suggested [16, 20]. An early software widely used for computing persistence is Morozov's Dionysus [23]. Later, Bauer et al. developed the PHAT toolbox [3], based on several efficient matrix reduction strategies (mostly focusing on time efficiency) as described in [2]. A more recently developed library called GUDHI [38] considers the improvement both in time and space efficiencies. In particular, it uses an efficient data structure, called the simplex tree [5], to encode input simplicial complexes, and uses the *compressed annotation matrix* [4] to implement the persistent cohomology algorithm. Dionysus, PHAT, and GUDHI offer efficient software for computing persistence induced by inclusions. For our algorithm, we need persistence induced by more general simplicial maps for which we use Simpers [36] developed on the basis of the algorithm in [21].

The above results and software cater to general persistence computations. In practice, often the persistence needs to be computed for a particular filtration called the *Vietoris-Rips* or *Rips filtration* in short. Given a set of points P embedded in $\mathbb{R}^d$ (or in more general metric spaces), the Rips complex $\mathcal{R}^\alpha(P)$ with radius or scale α is the clique complex induced by the set of edges $\{(p,p') \mid d(p,p') \le \alpha, p, p' \in P\}$. One is interested in the persistent homology induced by the sequence of Rips complexes $\mathcal{R}^{\alpha_1} \subseteq \mathcal{R}^{\alpha_2} \subseteq \cdots \subseteq \mathcal{R}^{\alpha_m}$ for a growing sequence of radii $\alpha_1 \le \alpha_2 \le \ldots \le \alpha_m$. Intuitively, the Rips complex at a specific scale α approximates the union of radius-α balls around sample points in P. Thus, it captures the structure formed by input points P at different scales.

Unfortunately, even for a modest size of n (in the range of thousands), the size of Rips complex (as well as the slightly more economical Čech complex) becomes prohibitively large as the radius α increases. In [35], Sheehy proposed an elegant solution for this problem by introducing a *sparse Rips filtration* to approximate the persistent homology of the Rips filtration for a set of points P. An alternative approach of collapsing input points in batches with increasing radius α was proposed in [21], which leveraged the persistence algorithm proposed in the same paper for filtrations arising out of simplicial maps.

New work

Given the importance of the Rips filtration in practice, our goal is to investigate the practical performance of the existing proposed methods, understand their advantages and limitations, and develop an efficient implementation for approximating the persistent homology of Rips filtrations. To this end, we make the following contributions.

1. We investigate the advantages and limitations of three existing methods, two based on Sparse Rips [35, 11], and another on Batch-collapse [21]. Specifically, experiments show that while the sparse Rips algorithm by Sheehy [35] has a theoretical guarantee on the size of the filtration and gives good approximation of the persistence diagrams for the Rips filtration in practice, it generates simplicial complexes of large size even for input of moderate size. This problem becomes more severe as the dimension of the input data increases. The algorithm fails to finish for several high dimensional data sets of rather moderate size. See Table 1 for some examples. The batch-collapse approach is much more space efficient (which leads to time efficiency as well). Nevertheless, we find that its size still becomes prohibitive for high dimensional data.

2. Based on the insights gained from experimenting with the existing approaches, we propose a new algorithm called *SimBa* that approximates a Rips filtration persistence via simplicial batch-collapses. Our algorithm is a modification of the previous batch-collapse of Rips filtration proposed in [21]. While theoretically, the modification may not seem major, empirically, it reduces the size of the filtration significantly and thus leads to a much more efficient approximation of the Rips filtration persistence. Furthermore, we show that this modification maintains a similar approximation guarantee as the batch-collapse of Rips filtration proposed in [21]. We describe the details of an efficient and practical implementation of SimBa, the software for which has been made publicly available from [36].

Two concepts, homology groups of a simplicial complex, and simplicial maps between two complexes are used throughout this paper. We refer the reader to any standard text such as [32] for details. We denote the p-dimensional homology group of a simplicial complex $\mathcal{K}$ under $\mathbb{Z}_2$ coefficients by $H_p(\mathcal{K})$.

2 Rips filtration and its approximation

Given a set of points $P \subset \mathbb{R}^d$, let $\langle p_0, \ldots, p_s \rangle$ denote the s-dimensional simplex spanned by vertices $p_0, \ldots, p_s \in P$. The Rips complex at scalar α is defined as $\mathcal{R}^\alpha(P) = \{\langle p_0, \ldots, p_s \rangle \mid \|p_i - p_j\| \leq \alpha, \text{ for any } i, j \in [0, s]\}$. Now consider the following *Rips filtration*:

$$\{\mathcal{R}^\alpha(P)\}_{\alpha>0} \;:=\; \mathcal{R}^{\alpha_1}(P) \hookrightarrow \mathcal{R}^{\alpha_2}(P) \cdots \hookrightarrow \mathcal{R}^{\alpha_n}(P) \cdots \tag{1}$$

The inclusion maps between consecutive complexes above induce homomorphisms between respective homology groups, giving rise to a so called *persistence module* for dimension p:

$$H_p(\mathcal{R}^{\alpha_1}(P)) \to H_p(\mathcal{R}^{\alpha_2}(P)) \to \ldots \to H_p(\mathcal{R}^{\alpha_n}(P)) \cdots \tag{2}$$

If a homology class is created at $\mathcal{R}^{\alpha_i}(P)$ (i.e, does not have pre-image under homomorphism $H_p(\mathcal{R}^{\alpha_{i-1}}) \to H_p(\mathcal{R}^{\alpha_i})$) and dies entering $\mathcal{R}^{\alpha_j}(P)$ (i.e, its image vanishes under homomorphism $H_p(\mathcal{R}^{\alpha_{j-1}}(P)) \to H_p(\mathcal{R}^{\alpha_j}(P))$, then α_i is its *birth time*, α_j is its *death time*, and the difference $\alpha_j - \alpha_i$ is called the *persistence* of the class. In each dimension, the persistence barcodes capture the persistence of such homology classes by using a horizontal bar with left and right end points at α_i and α_j respectively. These *persistence barcodes* of the above Rips filtration are often the target summary of P and/or of the space P samples, which one wishes to compute in topological data analysis.

The main bottleneck for computing the barcodes of a Rips filtration stems from its size blowup. As the parameter α grows, the Rips complex $\mathcal{R}^\alpha(P)$ can become huge very quickly. To address this blowup in size, Sheehy [35] suggested a novel approach of sparsifying the point set P as one proceeds with increasing α in a way that does not alter the barcodes too much. The idea is to replace the original Rips filtration $\{\mathcal{R}^\alpha(P)\}_{\alpha>0}$ on P with a sequence of smaller complexes $\{\mathcal{S}^\alpha\}_{\alpha>0}$ and show that the two sequences *interleave* at the homology level. Then, appealing to the results of interleaving persistence modules [12], one can show that the barcodes of $\{\mathcal{S}^\alpha\}_{\alpha>0}$ approximate those of $\{\mathcal{R}^\alpha(P)\}_{\alpha>0}$ reasonably. The complexes $\mathcal{S}^\alpha$ are constructed as the *union* of Rips-like complexes built on a sequence of subsets of P rather than the entire set P.

The union allows the complexes in $\{\mathcal{S}^\alpha\}_{\alpha>0}$ to be connected with inclusions and hence permits using efficient algorithms and software designed for inclusion induced persistence. However, the size of $\mathcal{S}^\alpha$ may still be large due to the union operation. An alternative is to avoid the union operation but allow deletion or collapse of vertices (and simplices) at larger scale α [11, 35] resulting into a sequence of Rips-like complexes connected with simplicial maps instead of inclusions. This approach, which we refer to as *Sparse Rips with collapse*, however achieves only moderate improvements in size reduction. We find that much more aggressive size reduction can be achieved by considering the collapse in a batched fashion that gives rise to the approach of *Batch-collapsed Rips* [21].

Finally, building on the batch-collapse idea, we propose a new approach, called *SimBa* that significantly reduces the size of Rips-like complexes and their computations. This is achieved primarily by replacing inter-point distances with *set distances* while computing the complexes. We prove that this approach still provably approximates the barcodes of the original Rips filtration in sequence (1).

In what follows, we provide more details about each existing method along with its performance in practice, which explains the motivation behind the new tool SimBa.

2.1 Sparse Rips filtration (inclusions)

Let P be a set of points in a metric space $(\mathcal{M}, d)$. A *greedy permutation* $\{p_1, .., p_n\}$ of P is defined recursively as follows: Let $p_1 \in P$ be any point and define p_i recursively as $p_i = \text{argmax}_{p \in P \setminus P_{i-1}} d(p, P_{i-1})$, where $P_{i-1} = \{p_1, ..., p_{i-1}\}$. This gives rise to a nested sequence of subsets $P_1 \subset P_2 \subset \cdots P_n = P$. Furthermore, each subset P_i is locally dense and uniform (net) in the following sense. Define the insertion radius λ_{p_i} of a point p_i as $\lambda_{p_i} = d(p_i, P_{i-1})$. Each P_i is a λ_{p_i}-net of P, meaning that $d(p, P_i) \leq \lambda_{p_i}$ for every $p \in P$ and $d(p, q) \geq \lambda_{p_i}$ for every distinct pairs $p, q \in P_i$. These nets can be extended to a single-parameter family of nets as $\{N_\gamma\}$ where $N_\gamma = \{p \in P | \lambda_p > \gamma\}$ is a γ-net of P.

Using the idea of Sheehy [35], Buchet et al. [6] define a Rips-like filtration using the above specific nets and assigning weights to points whose geometric interpretation is explained in [11]. Each point $p \in P$ is associated with a weight $w_p(\alpha)$ at scale α as

$$w_p(\alpha) = \begin{cases} 0 & \text{if } \alpha \leq \frac{\lambda_p}{\varepsilon} \\ \alpha - \frac{\lambda_p}{\varepsilon} & \text{if } \frac{\lambda_p}{\varepsilon} < \alpha \leq \frac{\lambda_p}{\varepsilon(1-\varepsilon)} \\ \varepsilon\alpha & \text{if } \frac{\lambda_p}{\varepsilon(1-\varepsilon)} \leq \alpha \end{cases}$$

where $0 < \varepsilon < 1$ is an input constant that controls the sparsity of the filtration. Then, the perturbed distance between pairs of points is defined as

$$\hat{d}_\alpha(p, q) = d(p, q) + w_p(\alpha) + w_q(\alpha).$$

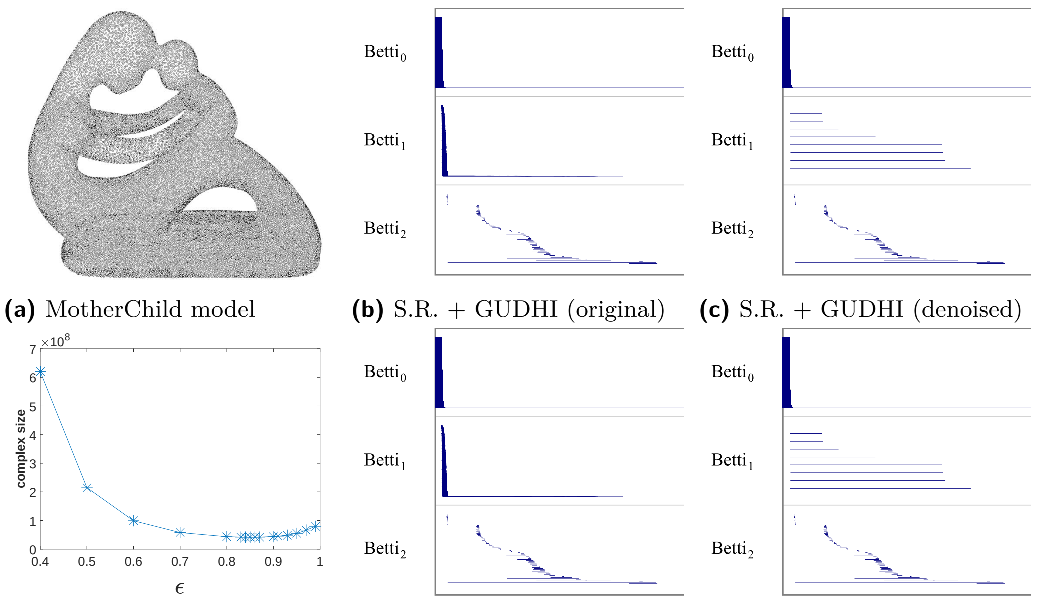
Using the perturbed distance $\hat{d}_\alpha$, the Sparse Rips complex at scale α is defined as

$$\mathcal{Q}^\alpha = \{\sigma \subset N_{\varepsilon(1-\varepsilon)\alpha} \mid \forall p, q \in \sigma,\ \hat{d}_\alpha(p, q) \leq 2\alpha\}.$$

The sequence of $\{\mathcal{Q}^\alpha\}_{\alpha>0}$ does not form a nested sequence of spaces because the vertex set of each $\mathcal{Q}^\alpha$ comes from the net $N_{\varepsilon(1-\varepsilon)\alpha}$ and may decrease as α increases. However, one can take $\mathcal{S}^\alpha = \bigcup_{\alpha' \leq \alpha} \mathcal{Q}^{\alpha'}$ and build a natural filtration $\{\mathcal{S}^\alpha \hookrightarrow \mathcal{S}^{\alpha'}\}_{\alpha < \alpha'}$ connected by inclusions. It is shown that the persistence barcodes of the filtration $\{\mathcal{S}^\alpha\}_{\alpha>0}$ approximate those of the Rips filtration $\{\mathcal{R}^\alpha\}_{\alpha>0}$ [35].

We use the code from [6] to compute this sparse Rips filtration $\{\mathcal{S}^\alpha\}_{\alpha>0}$. We then use GUDHI [38] to compute its persistent barcodes as GUDHI has the state-of-the-art performance for handling large complexes due to a compression technique [4] for inclusion-based filtrations.

As a common test case to illustrate the performance of various existing methods, we use a 3-dimensional point set sampled from a surface model called MotherChild; see Figure



(a) MotherChild model **(b)** S.R. + GUDHI (original) **(c)** S.R. + GUDHI (denoised)

(d) cumulative size **(e)** S.R. + Simpers (original) **(f)** S.R. + Simpers (denoised)

Figure 1 MotherChild surface model and its persistence barcodes computed by Sparse Rips (S.R.) based approaches. Since the surface has genus 4, its barcodes contain long bars: 1 for H_0, 8 for H_1, and 1 for H_2. The minimum cumulative size for complexes, which is about 43 million, is achieved around $\varepsilon = 0.8$.

1a. We choose this model because the ground truth is available and also because existing methods have trouble (to different degrees) handling high-dimensional data. The size of the point set is 23075. For indicating memory consumption, we refer to *cumulative size* which is the total number of simplices arising in the filtration, and also to *maximum size* which is the maximum over all complexes arising in the filtration. For Sparse Rips filtrations two sizes coincide at the last complex due to inclusions. Figure 1d shows the cumulative size with different Sparse Rips parameter ε. It is minimum when ε is between 0.8 and 0.9. So, we choose $\varepsilon = 0.8$ to achieve the best performance while observing that the approximation quality does not suffer much as predicted by the theory.

The original persistence barcodes are shown in Figure 1b. Since it becomes hard to see the main (long) bars in presence of all spurious ones creating excessive overlaps, we remove all short bars whose ratio between death and birth time is smaller than a threshold for 1-dimensional homology group H_1. The bars for H_0 and H_2 are not denoised. Unless specified otherwise, all barcodes are denoised in the same way. The denoised barcodes are shown in Figure 1c, one for H_0, four short and four long for H_1 (MotherChild has genus 4), and one for H_2. The cumulative size of the Sparse Rips complexes in the filtration is 43.5 million and the total time cost is about 350 seconds.

2.2 Sparse Rips with collapse

The persistence barcodes for the inclusion-based filtration $\{\mathcal{S}^\alpha\}_{\alpha>0}$ are the same as the barcodes of the filtration $\{\mathcal{Q}^\alpha\}_{\alpha>0}$ connected by simplicial maps $\mathcal{Q}^\alpha \to \mathcal{Q}^{\alpha'}$ for $\alpha < \alpha'$. Specifically, these simplicial maps originate from vertex collapses defined by the following projection map:

$$\mu_\alpha(p) = \begin{cases} p & \text{if } p \in N_{\varepsilon(1-\varepsilon)\alpha} \\ \operatorname*{argmin}_{q \in N_{\varepsilon\alpha}} d(p,q) & \text{otherwise} \end{cases}$$

For any scale α, the projection μ_α maps the points of P to the net $N_{\varepsilon(1-\varepsilon)\alpha} \supseteq N_{\varepsilon\alpha}$. One can view it as p being deleted at time (scale) $\alpha_p = \frac{\lambda_p}{\varepsilon(1-\varepsilon)}$. We can construct the sequence of Sparse Rips complexes $\{\mathcal{Q}^\alpha\}_{\alpha>0}$ connected with simplicial maps induced by insertions and vertex collapses as α increases: Specifically, we delete the vertex p and all its incident simplices by collapsing it to its projection $\mu_{\alpha_p}(p)$ where $\alpha_p = \frac{\lambda_p}{\varepsilon(1-\varepsilon)}$, when entering complex $\mathcal{Q}^{\alpha_p}$. See [11] for more details.

In this approach we need to compute the persistence induced with simplicial maps. For this, we use the only available software Simpers [36] based on the algorithm presented in [21]. Our experimental results on the MotherChild model of Figure 1a with $\varepsilon = 0.8$ are given in Figure 1e and 1f. The barcodes are exactly the same as those in Figure 1b and 1c. The cumulative size of the entire sequence is the same, 43.5 million, because the final complex in $\mathcal{S}^\alpha$ is the union of all complexes in $\{\mathcal{Q}^\alpha\}_{\alpha>0}$. However, the maximum size in the sequence is 24.9 million due to vertex collapses in contrast to the maximum size for $\{\mathcal{S}^\alpha\}_{\alpha>0}$ which equals the cumulative size. The time cost for this approach is 463.7 seconds which is larger than that for Sparse Rips with GUDHI. So compared to Sparse Rips with GUDHI, this approach has smaller maximum size due to collapse but costs more time for computing persistence since Simpers computes persistence over collapses which are slower operations than inclusions.

While the size of these Sparse Rips complexes is linear in the number of input points, the hidden constant factor depends exponentially on the doubling dimension of the metric space where points are sampled from. Empirically, we note that the size is still large, and becomes much worse as the dimension of data increases. For example, for the Gesture Phase data in Table 1 which has only 1747 points in $\mathbb{R}^{18}$, the cumulative size of the Sparse Rips filtration is 45.6 million, which approaches the limit GUDHI or Simpers can handle. For other larger data sets such as Primary Circle or Survivin, the complex reaches a size beyond this limit. Moreover, one has to pre-compute a greedy permutation of the input point set before constructing the Sparse Rips filtration. This computation is usually costly requiring furthest point computations for which software as efficient as ANN (for nearest neighbors) is not available. This motivates us to consider the batched approach considered next.

2.3 Batch-collapsed Rips

For handling large and high dimensional data, we need a more aggressive sparsification than the Sparse Rips filtration. We consider the Batch-collapsed Rips filtration, which has been proposed previously in [21] (section 6.1).

Given a set of points P, first set $V_0 := P$ and compute the shortest pairwise distance α. We next construct a sequence of vertex sets $V_k, k \in [0, m]$ such that V_{k+1} is an αc^{k+1}-net of V_k where $c > 1$ is a parameter that controls the rate of the scale increase. Consider the vertex map $\pi_k : V_k \to V_{k+1}$, for $k \in [0, m-1]$, such that for any $v \in V_k$, $\pi_k(v)$ is v's nearest neighbor in V_{k+1}. It can be shown that each vertex map π_k induces a well-defined simplicial map $s_k : \mathcal{R}^{\alpha c^k \frac{3c-1}{c-1}}(V_k) \to \mathcal{R}^{\alpha c^{k+1} \frac{3c-1}{c-1}}(V_{k+1})$. The *Batch-collapsed Rips filtration* is:

$$\mathcal{R}^0(V_0) \xrightarrow{s_0} \mathcal{R}^{\alpha c \frac{3c-1}{c-1}}(V_1) \cdots \xrightarrow{s_{m-1}} \mathcal{R}^{\alpha c^m \frac{3c-1}{c-1}}(V_m). \tag{3}$$

Using the line of proof in [21], one can show that the persistence of this sequence is a $3\log(\frac{2}{c-1}+3)$-approximation of the persistence diagram of Rips filtration given below.

$$\mathcal{R}^0(V_0) \hookrightarrow \mathcal{R}^{\alpha c}(V_0) \cdots \hookrightarrow \mathcal{R}^{\alpha c^m}(V_0). \tag{4}$$

The blowup in scale by the factor of $\frac{3c-1}{c-1}$ results from the proof, which in practice causes some problems. We elaborate this further. To satisfy the approximation guarantee, one has to show that the persistence modules arising from Batch-collapsed Rips in sequence (3) and the standard Rips in sequence (4) interleave. In particular, this requires that we have well-defined simplicial maps from complexes in sequence (3) to those in sequence (4) and vice versa. The multiplicative factor $\frac{3c-1}{c-1}$ is needed to ensure that there is a well-defined simplicial map $\mathcal{R}^{\alpha c^k}(V_0) \to \mathcal{R}^{\alpha c^k \frac{3c-1}{c-1}}(V_k)$, as $\mathcal{R}^{\alpha c^k \frac{3c-1}{c-1}}(V_k)$ has to be sufficiently connected to include all the images of the simplices in the domain Rips complex $\mathcal{R}^{\alpha c^k}(V_0)$. The side effect of this is that the Batch-collapsed Rips complex has to be built at a much larger scale than the Rips complex, and it ends up with many unnecessary connections and thus more simplices in practice. This also causes a trade-off: Larger c reduces the over-connection but results in a worse approximation factor leading to a worse approximation quality. It is not clear how to set an increase rate that achieves both good approximation quality and efficiency for a specific data set.

We experimented Batch-collapsed Rips with Simpers on the same MotherChild model. Figure 2a, 2b and 2c show the persistence barcodes for different values of c. Observe that smaller values of c give better approximation. The barcode for $c = 1.3$ is the most similar among the three to that of Sparse Rips filtration in Figure 1b which is supposed to be more accurate theoretically. On the other hand, when c grows more than 1.8, it starts to lose some main bars in H_1 and noisy bars get longer in H_2. On the other hand, Figure 3 shows that, as c increases, both complex size and time cost decrease drastically. When $c = 2.0$, it only involves less than $216K$ simplices and takes time 9.4 seconds while, although $c = 1.3$ gives more accurate barcode, its size (22.5 million) and time (325s) approach those of the Sparse Rips. This demonstrates the dilemma that Batch-collapsed Rips faces in practice. We address this issue in our new approach SimBa. In particular, when $c \le 2$, SimBa performs better than Batch-collapsed Rips for both size and time as shown in Figure 3 while capturing all main bars correctly as shown in Figure 2.

3 SimBa

To tame the over-connection in Batch-collapsed Rips, we replace the sequence in (3) with the sequence below where the parameter does not incur the extra factor $\frac{3c-1}{c-1}$:

$$\mathcal{B}^0(V_0) \to \mathcal{B}^{\alpha c}(V_1) \to \cdots \mathcal{B}^{\alpha c^m}(V_m) \tag{5}$$

The complexes $\mathcal{B}^{\alpha c^k}(V_k)$ are built on the same vertex sequence $\{V_k\}$ as in Batch-collapsed Rips, but the distances among the vertices of V_k are replaced with a *set distance* which allows us to avoid the over-connection. For two sets of points (clusters) $A, B \subset P$, we define their set distance as $d(A,B) = \min_{a\in A, b\in B} d(a,b)$. The sets that we consider are the pre-images of the vertices in V_k under the composition of projections π_i's, namely, for a vertex $v \in V_k$, we consider the set

$$B_v^k = \{p \in V_0 \mid \hat{\pi}_k(p) = v\} \text{ where } \hat{\pi}_k : V_0 \to V_k \text{ is defined as } \hat{\pi}_k = \pi_{k-1} \circ \cdots \circ \pi_0.$$

The complex $\mathcal{B}^{\alpha c^k}(V_k)$ is simply the clique complex induced by edges $\{(u,v) \in V_k \mid d(B_u^k, B_v^k) \le \alpha c^k\}$. Observe that $d(u,v) \ge d(B_u^k, B_v^k)$ which ensures that the normal connection between u and v for a Rips filtration at the respective scale is not missed by considering the set distance while still avoiding the over-connection.

It turns out that each vertex map (nearest neighbor projection) $\pi_k : V_k \to V_{k+1}$ induces a simplicial map $h_k : \mathcal{B}^{\alpha c^k}(V_k) \to \mathcal{B}^{\alpha c^{k+1}}(V_{k+1})$. Instead of recomputing the simplicial complex each time, we generate elementary insertion and collapse operations incrementally for each h_k in three steps: (i) collapse each $v \in V_k \setminus V_{k+1}$ to its image $\pi_k(v)$ in V_{k+1} along with all incident simplices, (ii) insert new edges between two vertices in V_{k+1} if the distance between the two sets they represent are smaller than or equal to the current scale, and (iii) insert all new clique simplices containing new edges generated by (i) and (ii). Each h_k is processed in one batch, starting from a simplicial complex on vertices in V_k and resulting in a simplicial complex on vertices in V_{k+1}. The collapse and insertions of new simplices are exactly what Simpers need for computing the persistence.

3.1 Implementation Details

The advantage of SimBa (and Batch-collapsed Rips) over Sparse Rips filtrations is mainly due to the batched approach, which requires us to compute δ-nets of a point set for some δ repeatedly. Its advantage over the Batch-collapsed Rips is credited to the use of set distances. These computations require k-nearest neighbor search and fixed radius search for which efficient library like ANN [31] exists. We take advantage of this available software.

To compute a δ-net of a given point set (to obtain V_{k+1} from V_k), we randomly pick an untouched point, say p, use fixed-radius search to find all points in the ball of radius δ around p, map them to it, and mark them processed. We do this repeatedly until there is no untouched point left. We observe that this sub-sampling procedure can be carried out faster at early stage when δ is small because those points whose nearest neighbor distances are larger than the current δ can be taken directly into the net–they are all mapped to themselves and no other points are mapped to them. So, we maintain a list L of the points ordered by their nearest neighbor distances in increasing order and process them sequentially for δ-net computations. To compute the net points V_{k+1} from V_k, we carry out the full sub-sampling process only on the points in V_k that are already known to have nearest neighbor distances below δ and the new ones that qualify from L for increased δ. After δ becomes more than the largest nearest neighbor distance, we convert to the usual net computation.

Next, we describe an efficient implementation of the set distance computation, which being a basic operation in SimBa, speeds it up significantly. A straightforward implementation requires quadratic time, but we can make it more efficient in practice with the help of the ANN library. We use a hybrid strategy as follows. The sets B_u^k for vertices $u \in V_k$ are maintained by a union-find data structure. As vertices are collapsed while going from V_k to V_{k+1}, the sets of the collapsed vertices are merged to that of the target vertex. At early stages, when the number of sets (i.e, the size of V_k) is large and the diameter of each set is potentially small, we avoid computing set distances for all pairs. For each processing set B_u^k, we only need to find all the sets B_v^k whose distances to B_u^k are smaller than the current scale $\alpha' = \alpha c^k$. If so, we add an edge between u and v. To find all these nearby sets, we can do a fixed-radius search in $V_0 = P$ around each point in B_u^k within α' distance. For each point v returned by the search, we find v in the union-find data structure to identify its image $\hat{\pi}_k(v) \in V_k$. If the representing set of v is different from that of u, we add the edge $\hat{\pi}_k(u)\hat{\pi}_k(v)$.

Later when α' becomes large, it may not be efficient to continue this fixed-radius search, as the number of candidate points from P may be too large (can be n in the worst case). So we fall back on pairwise set distance computation. In particular, when the cardinality of V_k becomes lower than a threshold, say 1/10 of the number of input points, we compute a pairwise set distance matrix (of size $|V_k| \times |V_k|$) among the surviving sets once and then keep updating the matrix with batch collapse thereafter. In particular, note that given sets A, B, and C, the set distance $d(A \cup B, C) = \min\{d(A, C), d(B, C)\}$.

3.2 SimBa on MotherChild model

We compare SimBa with other approaches on the same MotherChild model. Figure 2d, 2e and 2f show the persistence barcodes computed by SimBa with different values of c. We see that SimBa captures all the main 0, 1, 2-dimensional bars for all values of c in the range from 1.3 to 2.0 as opposed to Batch-collapsed Rips which fails to capture the main H_1 bars for $c > 1.8$. It tolerates larger range of c and thus is more robust than Batch-collapsed Rips. As expected, larger values of c produce less bars since there are less batches. So, in practice, we should choose smaller c, say less than 1.5. More importantly, as Figure 3 shows, the size and time for SimBa are also stable against different values of c, all less than $100K$ simplices and 10 seconds respectively for $c \leq 2$. These are less than those for Batch-collapsed Rips and significantly less than those for Sparse Rips: In particular, when $c = 1.3$, the maximum size for SimBa is $100K$, similar to when $c = 2$. However, for Batch-collapsed Rips, the maximum size is closer to that of SimBa when $c = 2$, and is 22.5 and 1.4 million when $c = 1.3$ and $c = 1.5$ respectively. This size difference becomes even more prominent for high dimensional data, as Table 1 shows. Although the approximation quality of SimBa is slightly worse than that of Sparse Rips based approaches, it does capture all the main bars, and more importantly, costs significantly less time. This advantage allows SimBa to process much larger high dimensional data sets which no previous approaches can handle, as we illustrate in section 5.

4 Approximation guarantee of SimBa

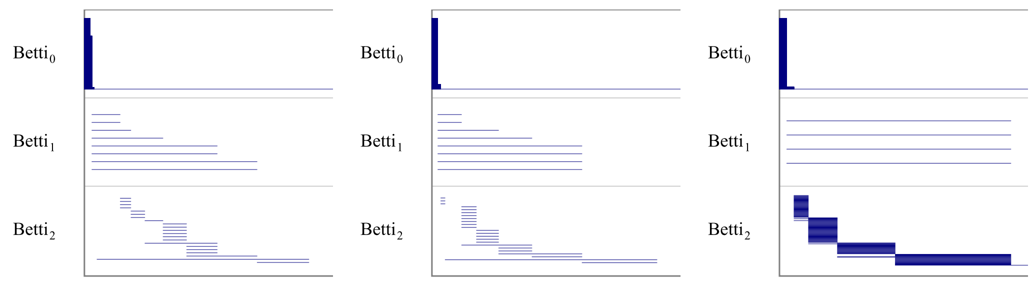
Recall that the simplicial complex $\mathcal{B}^\alpha(V_k)$ appearing in SimBa's filtration is defined as:

$$\mathcal{B}^\alpha(V_k) = \{\sigma \subset V_k \mid \forall u, v \in \sigma, d(B_u^k, B_v^k) \leq \alpha\}.$$

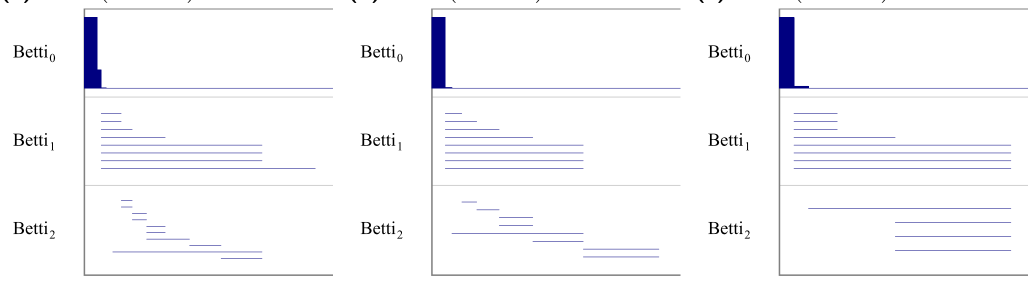
We prove that the persistence barcodes of SimBa's filtration in sequence (5) approximates those of the Rips filtration in (4) by showing that the persistence modules induced by these two sequences interleave.

First, observe that each vertex map π_k induces a well-defined simplicial map $h_k : \mathcal{B}^{\alpha c^k}(V_k) \to \mathcal{B}^{\alpha c^{k+1}}(V_{k+1})$. Indeed, for any edge $\{u, v\}$ in $\mathcal{B}^{\alpha c^k}(V_k)$, suppose $u' = \pi_k(u), v' = \pi_k(v)$, then $B_u^k \subset B_{u'}^{k+1}$ and $B_v^k \subset B_{v'}^{k+1}$. So we have $d(B_{u'}^{k+1}, B_{v'}^{k+1}) \leq d(B_u^k, B_v^k) \leq \alpha c^k < \alpha c^{k+1}$. Therefore $\{u', v'\}$ must be an edge in $\mathcal{B}^{\alpha c^{k+1}}(V_{k+1})$ as well. Since each complex in SimBa's filtration is a clique complex determined by edges, every simplex in $\mathcal{B}^{\alpha c^k}(V_k)$ has a well-defined image in $\mathcal{B}^{\alpha c^{k+1}}(V_{k+1})$. Thus, each h_k is well-defined.

Recall that the map $\hat{\pi}_k : V_0 \to V_{k+1}$ is defined as $\hat{\pi}_k(v) = \pi_k \circ \cdots \circ \pi_0(v)$, which tracks the image of any point in $V_0 = P$ during the batch collapse process. Observe that the vertex map $\hat{\pi}_k$ also induces a simplicial map $\hat{h}_k : \mathcal{R}^{\alpha c^k}(V_0) \to \mathcal{B}^{\alpha c^{k+1}}(V_{k+1})$: specifically, for any edge $(u, v) \in \mathcal{R}^{\alpha c^k}(V_0)$ with $d(u, v) \leq \alpha c^k$, it is easy to see that $d(B_{\hat{\pi}_k(u)}^k, B_{\hat{\pi}_k(v)}^k) \leq d(u, v) \leq \alpha c^k < \alpha c^{k+1}$, implying that $(\hat{\pi}_k(u), \hat{\pi}_k(v))$ is an edge in $\mathcal{B}^{\alpha c^{k+1}}(V_{k+1})$. The key observation is the following lemma.



(a) B.R. (c = 1.3) **(b)** B.R. (c = 1.5) **(c)** B.R. (c = 2.0)



(d) SimBa (c = 1.3) **(e)** SimBa (c = 1.5) **(f)** SimBa (c = 2.0)

Figure 2 Persistence barcodes computed by Batch-collapsed Rips plus Simpes (B.R.) and SimBa on the same MotherChild model. B.R. captures main bars for H_1 correctly for smaller values of c as shown in Figure (a) and (b) and loses some for $c = 2.0$ as shown in Figure (c), while SimBa works for $c = 2.0$.

▶ **Lemma 1.** *Each triangle in the diagram below commutes at homology level, where i_k and j_k are induced by inclusions, $h_{k,t} := h_{k+t-1} \circ \cdots \circ h_k$, $c > 1$, $t \geq \log_c(\frac{2}{c-1} + 3)$ and $t \in Z$.*

$$\begin{tikzcd}
\mathcal{R}^{\alpha c^k}(V_0) \arrow[r, hook, "i_k"] \arrow[d, "\hat{h}_k"] & \mathcal{R}^{\alpha c^{k+t}}(V_0) \arrow[d, "\hat{h}_{k+t}"] \\
\mathcal{B}^{\alpha c^k}(V_k) \arrow[ur, hook, "j_k"] \arrow[r, "h_{k,t}"] & \mathcal{B}^{\alpha c^{k+t}}(V_{k+t})
\end{tikzcd}$$

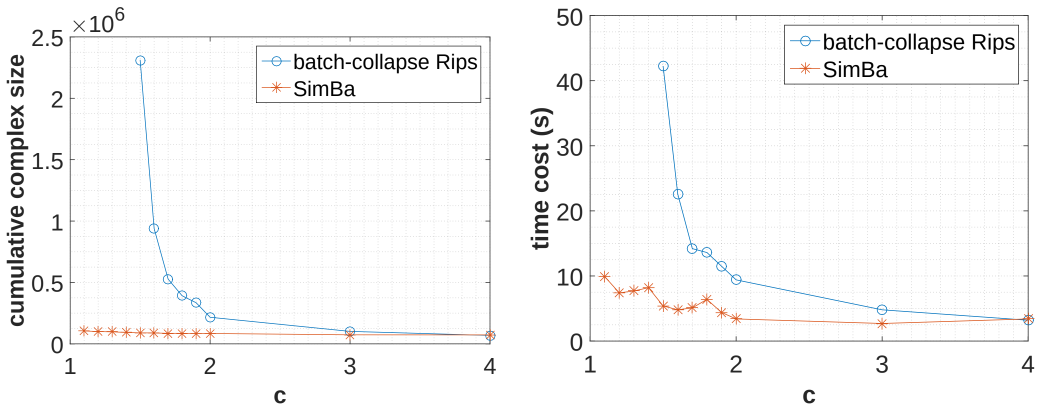
Proof. First, we prove that there is indeed an inclusion map $j_k : \mathcal{B}^{\alpha c^k}(V_k) \hookrightarrow \mathcal{R}^{\alpha c^{k+t}}(V_0)$. In particular, we show for each edge (u, v) in $\mathcal{B}^{\alpha c^k}(V_k)$, it's also an edge in $\mathcal{R}^{\alpha c^{k+t}}(V_0)$. Suppose the set distance $d(B_u^k, B_v^k)$ is achieved by the closest pair (u_0, v_0) between the two sets where $u_0 \in B_u^k, v_0 \in B_v^k$. Then $d(B_u^k, B_v^k) = d(u_0, v_0) \leq \alpha c^k$. Since V_{i+1} is an αc^{i+1}-net of V_i for each $i \in [0, k-1]$, it follows that $d(u, u_0) \leq \alpha c^k \sum_{i=0}^{k-1} \frac{1}{c^i} < \alpha c^k \frac{c}{c-1}$. Similar bound holds for $d(v, v_0)$. Thus:

$$d(u, v) \leq d(u, u_0) + d(v, v_0) + d(u_0, v_0) \leq \alpha c^k (\frac{2c}{c-1} + 1) = \alpha c^k (\frac{2}{c-1} + 3) \leq \alpha c^{k+t}.$$

Hence u, v is an edge in $\mathcal{R}^{\alpha c^{k+t}}(V_0)$.

Next, observe that the vertex map $\hat{\pi}_{k+t}$ restricted on the set of vertices V_k is exactly the same as the vertex map $\pi_{k,t} := \pi_{k+t-1} \circ \cdots \circ \pi_k$ (this vertex map induces the simplicial map $h_{k,t}$ in the diagram). Namely, for a vertex $u \in V_k \subseteq V_0$, $h_{k,t}(u) = \hat{h}_{k+t}(u)$. Thus $h_{k,t} = \hat{h}_{k+t} \circ j_k$. Hence the bottom triangle commutes both at the complex and the homology level.

We now consider the top triangle. Specifically, we prove that the map $j_k \circ \hat{h}_k$ is contiguous to the inclusion map i_k. Since two contiguous maps induce the same homomorphisms at the homology level, the top triangle commutes at the homology level.



(a) cumulative size **(b)** time cost

Figure 3 Complex size and time cost comparison between Batch-collapsed Rips and SimBa. SimBa beats Batch-collapsed Rips for both size and time when $c \leq 2$. For $c > 2$, the barcodes of both batch-based approaches become too coarse to be useful in practice.

Indeed, given a simplex $\sigma \in \mathcal{R}^{\alpha c^k}(V_0)$, we need to show that vertices from $i_k(\sigma) \cup j_k \circ \hat{h}_k(\sigma)$ span a simplex in $\mathcal{R}^{\alpha c^{k+t}}(V_0)$. Since both are Rips complexes and i_k and j_k are inclusion maps, we only need to prove that for any two vertices u and v from $\sigma \cup \hat{h}_k(\sigma)$, $d(u,v) \leq \alpha c^{k+t}$ (namely, (u,v) is an edge in $\mathcal{R}^{\alpha c^{k+t}}(V_0)$). If u and v are both from σ or both from $\hat{h}_k(\sigma)$, then $d(u,v) \leq \alpha c^{k+t}$ trivially. Otherwise, assume without loss of generality that $v \in \sigma$ and $u \in \hat{h}_k(\sigma)$, where $u = \hat{\pi}_k(u')$ for some $u' \in \sigma$. Since V_{i+1} is an is an αc^{i+1}-net of V_i for each $i \in [0, k-1]$, it follows that $d(u,u') \leq \alpha c^k \sum_{i=0}^{k-1} \frac{1}{c^i} < \alpha c^k \frac{c}{c-1}$. One then has

$$d(u,v) \quad \leq \quad d(u,u') + d(u',v) \leq \alpha c^k \frac{c}{c-1} + \alpha c^k = \alpha c^k \frac{2c-1}{c-1} < \alpha c^k (\frac{2}{c-1} + 3) \leq \alpha c^{k+t}.$$

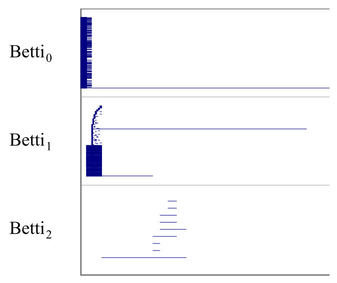
Thus i_k is contiguous to $j_k \circ \hat{h}_k$ and the lemma follows. ◀

The above result implies that the persistence modules induced by sequences (5) and (4) are weakly $\log c^t$-interleaved at the log-scale. Since $t \geq \log_c(\frac{2}{c-1} + 3)$, we have $c^t \geq \frac{2}{c-1} + 3$. By Theorem 4.3 of [12], we conclude with the following:

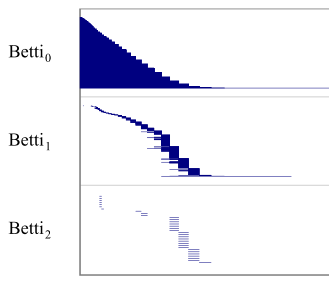
▶ **Theorem 2.** *The persistence diagram of the sequence (5) provides a $3\log(\frac{2}{c-1} + 3)$-approximation of the persistence diagram of the sequence (4) at the log-scale for $c > 1$.*

5 Experiments

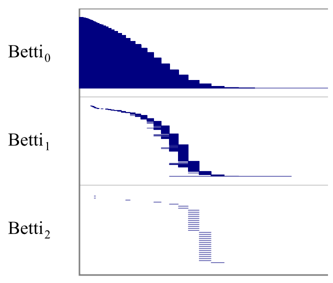
In this section, we report some experimental results of SimBa on large high dimensional data sets from other fields such as image processing, machine learning, and computational biology. For most of the data sets, previous approaches are not efficient enough to finish processing. They either ran out of memory ('∞' in size) or ran more than one day ('∞' in time). Table 1 at the end of this section provides the cumulative size and time cost for all four approaches mentioned in this paper. All approaches are implemented in C++. Note that we only compute persistences up to dimension 2 (which means we build simplicial complexes up to dimension 3). For Sparse Rips with GUDHI and Sparse Rips with Simpers, we choose parameter $\varepsilon = 0.8$ which gives the best performance while not sacrificing much of the approximation quality. For Batch-collapsed Rips with Simpers, we choose $c = 1.5$



(a) Klein Bottle in $\mathbb{R}^4$



(b) Primary Circle in $\mathbb{R}^{25}$



(c) Primary Circle in $\mathbb{R}^{49}$

Figure 4 Original persistence barcodes computed by SimBa on data sets with ground truth.

which appears to reach a good trade-off between efficiency and quality. For SimBa, we choose $c = 1.1$ which in practice appears to have best quality – note that the choice of c does not seem to change the empirical efficiency much as Figure 3 illustrates. All experiments were run on a 64-bit Windows machine with a 3.50GHz Intel processor and 16GB RAM.

Data with ground truth

We first test with two data sets whose ground truth persistences are known. They help demonstrate that SimBa works properly and efficiently in practice. All persistence barcodes shown in Figure 4 are original and not cleaned up.

We first consider a uniform sample of 22500 points from a Klein bottle in $\mathbb{R}^4$, and use SimBa to compute its barcode which is shown in Figure 4a. There are two main bars for H_1 and one for H_2 as expected.

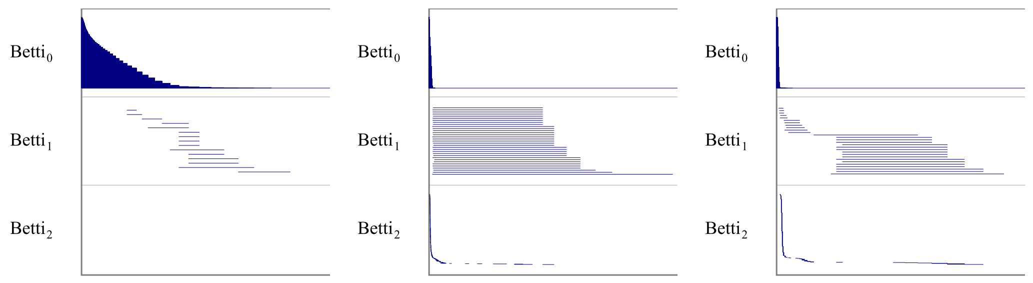
Next, we consider the primary circle of natural image data in [1], which has 15000 points. Each point is a 5×5 or 7×7 image patch, thus considered as a point in $\mathbb{R}^{25}$ or $\mathbb{R}^{49}$. From Figures 4b and 4c, we can see the primary circle bar for H_1 for data both in $\mathbb{R}^{25}$ and $\mathbb{R}^{49}$. All short bars for H_2 persist for only one batch step and thus can be regarded as noise.

Data without ground truth

Next, we provide some experiments on the data sets whose ground truth persistences are not known. We used SimBa to compute their persistences and found some relatively long bars which are likely to be features and may worth further investigation by domain experts. The persistence barcodes shown in Figure 5 and 6d are denoised for H_1. The rest of Figure 6 are original.

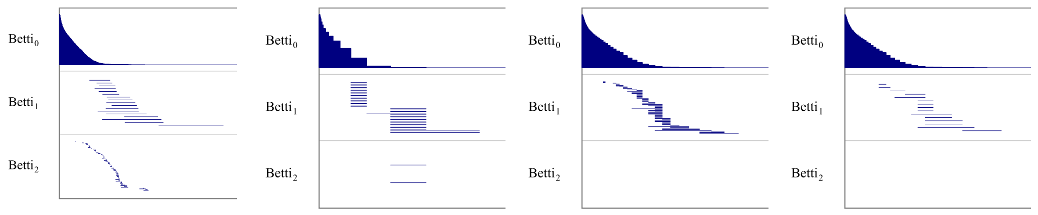
We first take the Gesture Phase Segmentation data set [30] from UCI machine learning repository [28]. This data set was used in [29]. It comprises of features extracted from 7 videos with people gesticulating. Each video is represented by a raw file that contains the positions of hands, wrists, head, and spine of the user in each frame. We took the raw file from video A1 of 1747 frames. Since there are six sensors each with x, y, z coordinates, we have in total 1747 points in $\mathbb{R}^{18}$. There are five gesture phases in the videos: rest, preparation, stroke, hold, and retraction. Indeed, there are five long bars for H_0 in 5a (although they overlap and do not stand out in the picture), which seems to match the five clusters of different phases. We see some long bars for H_1, which could be created due to periodic patterns in these gesture movements.

Another data set is the Survivin protein data from [27]. There are totally 252996 protein conformations and each conformation is considered as a point in $\mathbb{R}^{150}$. We used PCA to



(a) Gesture Phase data in $\mathbb{R}^{18}$ **(b)** Survivin data in $\mathbb{R}^3$ **(c)** Survivin data in $\mathbb{R}^{150}$

Figure 5 Denoised persistence barcodes computed by SimBa on data sets without ground truth.



(a) S.R.+GUDHI **(b)** B.R.+Simpers **(c)** SimBa **(d)** SimBa (denoised)

Figure 6 Persistence barcodes computed by different approaches on Gesture Phase Segmentation data in $\mathbb{R}^{18}$.

Table 1 cumulative size and time cost.

				S.R.+GUDHI		**S.R.+Simpers**		**B.R.+Simpers**		**SimBa**	
Data	n	D	d	size	time(s)	size	time(s)	size	time(s)	size	time(s)
Mother	23075	3	2	$43.5 \cdot 10^6$	350	$43.5 \cdot 10^6$	463.7	$2.3 \cdot 10^6$	42.3	104701	8.8
KlBt	22500	4	2	$20.9 \cdot 10^6$	205.3	$20.9 \cdot 10^6$	303.5	440049	8	78064	6.6
PrCi25	15000	25	?	∞	–	∞	–	–	∞	$4.8 \cdot 10^6$	216
PrCi49	15000	49	?	∞	–	∞	–	–	∞	$10.2 \cdot 10^6$	585
GePh	1747	18	?	$45.6 \cdot 10^6$	282.5	$45.6 \cdot 10^6$	432.8	$1.4 \cdot 10^6$	29	7145	0.83
Sur3	252996	3	?	∞	–	∞	–	$15.7 \cdot 10^6$	1056.4	915110	1079.6
Sur150	252996	150	?	∞	–	∞	–	–	∞	$3.1 \cdot 10^6$	5089.7

reduce the data dimension to 3. We ran SimBa on both data sets and show the barcodes in Figure 5c and 5b. We can see that there are some long bars for H_1.

Performance results

We provide the performance results for all data sets mentioned in Table 1, which includes cumulative size and time cost of each approach. The time is obtained by adding the time to construct the complexes and the time to compute persistence. **S.R.+GUDHI**, **S.R.+Simpers**, **B.R.+Simpers** and **SimBa** stand for Sparse Rips plus GUDHI, Sparse Rips plus Simpers, Batch-collapsed Rips plus Simpers, and SimBa respectively. **Mother**, **KlBt**, **PrCi25**, **PrCi49**, **GePh**, **Sur3** and **Sur150** stand for MotherChild model, Klein Bottle, Primary Circle in $\mathbb{R}^{25}$, Primary Circle in $\mathbb{R}^{25}$, Gesture Phase Segmentation data, Survivin protein data in $\mathbb{R}^3$ and in $\mathbb{R}^{150}$ respectively. Each data set has size n, ambient dimension D, and intrinsic dimension d. The symbol ∞ means that the program either ran out of memory or did not finish after a day. From the table, we can see that SimBa out-performed the other three approaches significantly. Notice that for those larger cases of SimBa, the nearest neighbor search operations (ANN) usually take most of time and become the bottleneck. This is why **Sur150** costs much more time than **PrCi49** while its cumulative size is smaller. It would be an interesting future work to make nearest neighbor search more efficient so that SimBa performs better even for such cases.

References

1 H. Adams and G. Carlsson. On the nonlinear statistics of range image patches. *SIAM J. Img. Sci.*, 2(1):110–117, 2009. doi:10.1137/070711669.

2 U. Bauer, M. Kerber, and J. Reininghaus. *Topological Methods in Data Analysis and Visualization III: Theory, Algorithms, and Applications*, chapter Clear and Compress: Computing Persistent Homology in Chunks, pages 103–117. Springer International Publishing, Cham, 2014. doi:10.1007/978-3-319-04099-8_7.

3 U. Bauer, M. Kerber, J. Reininghaus, and H. Wagner. *Mathematical Software – ICMS 2014: 4th International Congress, Seoul, South Korea, August 5-9, 2014. Proceedings*, chapter PHAT – Persistent Homology Algorithms Toolbox, pages 137–143. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014. Project URL: https://bitbucket.org/phat-code/phat.

4 J.-D. Boissonnat, T. K. Dey, and C. Maria. *Algorithms – ESA 2013: 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings*, chapter The Compressed Annotation Matrix: An Efficient Data Structure for Computing Persistent Cohomology, pages 695–706. Springer, Berlin, Heidelberg, 2013. doi:10.1007/978-3-642-40450-4_59.

5 J.-D. Boissonnat and C. Maria. *Algorithms – ESA 2012: 20th Annual European Symposium, Ljubljana, Slovenia, September 10-12, 2012. Proceedings*, chapter The Simplex Tree: An Efficient Data Structure for General Simplicial Complexes, pages 731–742. Springer, 2012.

6 M. Buchet, F. Chazal, S. Y. Oudot, and D. R. Sheehy. Efficient and robust persistent homology for measures. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 168–180, 2015. doi:10.1137/1.9781611973730.13.

7 D. Burghelea and T. K. Dey. Topological persistence for circle-valued maps. *Discrete Comput. Geom.*, 50:69–98, 2013.

8 G. Carlsson. Topology and data. *Bull. Amer. Math. Soc.*, 46:255–308, 2009.

9 G. Carlsson and V. de Silva. Zigzag persistence. *Foundations of computational mathematics*, 10(4):367–405, 2010.

10 G. Carlsson and A. Zomorodian. The theory of multidimensional persistence. *Discrete & Computational Geometry*, 42(1):71–93, 2009. doi:10.1007/s00454-009-9176-0.

11 N. J. Cavanna, M. Jahanseir, and D. R. Sheehy. A geometric perspective on sparse filtrations. In *Canadian Conf. Comput. Geom. (CCCG)*, 2015. URL: http://dblp.uni-trier.de/db/conf/cccg/cccg2015.html#CavannaJS15.

12 F. Chazal, D. Cohen-Steiner, M. Glisse, L. J. Guibas, and S. Y. Oudot. Proximity of persistence modules and their diagrams. In *Proceedings of the Twenty-fifth Annual Symposium on Computational Geometry*, SCG'09, pages 237–246, New York, NY, USA, 2009. ACM. doi:10.1145/1542362.1542407.

13 F. Chazal, D. Cohen-Steiner, L. Guibas, F. Mémoli, and S. Y. Oudot. Gromov-Hausdorff stable signatures for shapes using persistence. In *Proc. of SGP*, 2009.

14 F. Chazal, V. de Silva, M. Glisse, and S. Oudot. The structure and stability of persistence modules. *CoRR*, abs/1207.3674, 2012.

15 F. Chazal, L. J. Guibas, S. Y. Oudot, and P. Skraba. Persistence-based clustering in Riemannian manifolds. In *Proc. 27th Annu. ACM Sympos. Comput. Geom.*, pages 97–106, 2011.

16 C. Chen and M. Kerber. An output-sensitive algorithm for persistent homology. *Comput. Geom. Theory Appl.*, 46(4):435–447, May 2013. doi:10.1016/j.comgeo.2012.02.010.

17 D. Cohen-Steiner, H. Edelsbrunner, and J. Harer. Stability of persistence diagrams. *Discrete & Computational Geometry*, 37(1):103–120, 2007.

18 D. Cohen-Steiner, H. Edelsbrunner, and J. Harer. Extending persistence using Poincaré and Lefschetz duality. *Foundations of Computational Mathematics*, 9(1):79–103, 2009.

19 D. Cohen-Steiner, H. Edelsbrunner, and D. Morozov. Vines and vineyards by updating persistence in linear time. In *Proceedings of the twenty-second annual symposium on Computational geometry*, pages 119–126. ACM, 2006.

20 V. de Silva, D. Morozov, and M. Vejdemo-Johansson. Persistent cohomology and circular coordinates. *Discrete Comput. Geom.*, 45(4):737–759, 2011.

21 T. K. Dey, F. Fan, and Y. Wang. Computing topological persistence for simplicial maps. In *Proceedings of the Thirtieth Annual Symposium on Computational Geometry*, SOCG'14, pages 345–354. ACM, 2014. doi:10.1145/2582112.2582165.

22 T. K. Dey, K. Li, C. Luo, P. Ranjan, I. Safa, and Y. Wang. Persistent heat signature for pose-oblivious matching of incomplete models. *Comput. Graph. Forum. (special issue from* Sympos. Geom. Process.*)*, 29(5):1545–1554, 2010.

23 Dmitriy Morozov. Dionysus Software. http://mrzv.org/software/dionysus/, 2012.

24 H. Edelsbrunner and J. Harer. *Computational Topology – an Introduction.* American Mathematical Society, 2010.

25 H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. *Discrete Comput. Geom.*, 28:511–533, 2002.

26 P. Frosini. A distance for similarity classes of submanifolds of a euclidean space. *Bulletin of the Australian Mathematical Society*, 42(3):407–416, 1990.

27 W. Harvey, I.-H. Park, O. Rübel, V. Pascucci, P.-T. Bremer, C. Li, and Y. Wang. A collaborative visual analytics suite for protein folding research. *Journal of Molecular Graphics and Modelling*, 53:59–71, 2014. doi:10.1016/j.jmgm.2014.06.003.

28 M. Lichman. UCI machine learning repository, 2013. Project URL: http://archive.ics.uci.edu/ml.

29 R. C. B. Madeo, S. M. Peres, and C. A. de M. Lima. Gesture phase segmentation using support vector machines. *Expert Systems with Applications*, 56:100–115, 2016. doi:10.1016/j.eswa.2016.02.021.

30 R. C. B. Madeo, P. K. Wagner, and S. M. Peres. Gesture Phase Segmentation Data Set, 2014. Project URL: http://archive.ics.uci.edu/ml/datasets/Gesture+Phase+Segmentation.

31 D. M. Mount and S. Arya. ANN: A Library for Approximate Nearest Neighbor Searching, 2010. Project URL: https://www.cs.umd.edu/~mount/ANN/.

32 James R. Munkres. *Elements of Algebraic Topology.* Addison-Wesley, 1993.

33 J. Reininghaus, S. Huber, U. Bauer, and R. Kwitt. A stable multi-scale kernel for topological machine learning. In *Proc. IEEE Conf. Comp. Vision & Pat. Rec. (CVPR)*, pages 4741–4748, 2015.

34 V. Robins. Towards computing homology from finite approximations. *Topology Proceedings*, 24(1):503–532, 1999.

35 D. R. Sheehy. Linear-size approximations to the vietoris-rips filtration. In *Proceedings of the Twenty-eighth Annual Symposium on Computational Geometry*, SoCG'12, pages 239–248. ACM, 2012. doi:10.1145/2261250.2261286.

36 Simpers Software, 2015. Project URL: http://web.cse.ohio-state.edu/~tamaldey/SimPers/Simpers.html.

37 G. Singh, F. Mémoli, T. Ishkhanov, G. Sapiro, G. Carlsson, and D. L Ringach. Topological analysis of population activity in visual cortex. *Journal of vision*, 8(8):11, 2008.

38 The GUDHI Project. GUDHI user and reference manual, 2015. Project URL: `http://gudhi.gforge.inria.fr/doc/latest/`.

39 A. Zomorodian and G. Carlsson. Computing persistent homology. *Discrete Comput. Geom.*, 33(2):249–274, 2005.

Exponential Time Paradigms Through the Polynomial Time Lens[*]

Andrew Drucker[1], Jesper Nederlof[†2], and Rahul Santhanam[‡3]

1 **Computer Science Department, University of Chicago, Chicago, USA**
`andy.drucker@gmail.com`

2 **Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands**
`J.Nederlof@tue.nl`

3 **Department of Computer Science, University of Oxford, Oxford, UK**
`rahul.santhanam@cs.ox.ac.uk`

Abstract

We propose a general approach to modelling algorithmic paradigms for the exact solution of NP-hard problems. Our approach is based on polynomial time reductions to succinct versions of problems solvable in polynomial time. We use this viewpoint to explore and compare the power of paradigms such as branching and dynamic programming, and to shed light on the true complexity of various problems.

As one instantiation, we model branching using the notion of witness compression, i.e., reducibility to the circuit satisfiability problem parameterized by the number of variables of the circuit. We show this is equivalent to the previously studied notion of 'OPP-algorithms', and provide a technique for proving conditional lower bounds for witness compressions via a constructive variant of AND-composition, which is a notion previously studied in theory of preprocessing. In the context of parameterized complexity we use this to show that problems such as Pathwidth and Treewidth and Independent Set parameterized by pathwidth do not have witness compression, assuming $NP \not\subseteq coNP/poly$. Since these problems admit fast fixed parameter tractable algorithms via dynamic programming, this shows that dynamic programming can be stronger than branching, under a standard complexity hypothesis. Our approach has applications outside parameterized complexity as well: for example, we show if a polynomial time algorithm outputs a maximum independent set of a given planar graph on n vertices with probability $\exp(-n^{1-\epsilon})$ for some $\epsilon > 0$, then $NP \subseteq coNP/poly$. This negative result dims the prospects for one very natural approach to sub-exponential time algorithms for problems on planar graphs.

As two other illustrations (more exploratory) of our approach, we model algorithms based on inclusion-exclusion or group algebras via the notion of "parity compression", and we model a subclass of dynamic programming algorithms with the notion of "disjunctive dynamic programming". These models give us a way to naturally classify various parameterized problems with FPT algorithms. In the case of the dynamic programming model, we show that Independent Set parameterized by pathwidth is complete for this model.

1998 ACM Subject Classification F.2.0. Analysis of Algorithms and Problem Complexity

Keywords and phrases exponential time paradigms, branching, dynamic programming, lower bounds

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.36

[*] This work was partly done while the authors were visiting the Simons Institute for the Theory of Computing during the program 'Fine-Grained Complexity and Algorithm Design' in the fall of 2015.

[†] Funded by the NWO VENI project 639.021.438.

[‡] Supported by the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013)/ERC Grant Agreement no. 615075

1 Introduction

The successes of theoretical computer science have often been driven by simple but general algorithmic approaches, or *paradigms*, leading to efficient algorithms in many different application domains. Indeed, paradigms such as divide-and-conquer, branching, dynamic programming and linear programming have been applied over and over to design algorithms.

A natural question that arises is to *quantify* the power and limitations of a given algorithmic paradigm. Doing so may have several benefits. It can help us understand what makes the paradigm effective. It can make algorithm design and analysis less ad hoc, with greater clarity about when and for which problems the paradigm is relevant. It can also enable us to compare various algorithmic paradigms with each other in terms of their power and usefulness. A crucial challenge in studying the power of algorithmic paradigms is the *modelling* question. We need a modelling framework which is rich enough to capture existing, successful algorithms within the paradigm. On the other hand, we need the modelling framework to be meaningfully restricted, so that we can prove interesting things about these models and the limits of their power. These goals are often in tension.

We aim to model exponential time algorithms. Understanding what can be computed in exponential time seems to be harder than understanding what can be computed in polynomial time, and less is known. In particular, showing general exponential-time lower bounds based on standard hypotheses about *polynomial-time* computation (for example, the hypotheses that $P \neq NP$ or that the Polynomial Hierarchy is infinite) seems out of reach. We propose to bypass this issue by arguing that several specific, established algorithmic paradigms can be modelled as polynomial-time reductions to (succinct) problems, so that limitations to their power may follow from these kinds of traditional hypotheses.

Approaches to algorithmic modelling can be broadly classified into syntactic and semantic approaches. Syntactic approaches attempt to faithfully represent the step-by-step operation of algorithms conforming to the method. Examples include the modelling of 1. DPLL algorithms by proof systems such as Resolution, 2. backtracking and dynamic programming by certain kinds of branching programs [1], 3. dynamic programming by feasible dominance relations [16], 4. linear programming by extended formulations [5]. These approaches, though natural, suffer from some drawbacks. The first is their lack of flexibility—they can fail to capture simple-looking variants of the method, e.g., the failure of proof systems to capture randomization. Second, in the search for accuracy, the models produced by such approaches can get quite complicated, which makes them hard to analyze.

Our models, in contrast, are semantic—we try to capture broad features of the algorithmic method rather than trying to model it in a step-by-step fashion. In particular, we allow arbitrary polynomial-time computations as constituent subroutines. This allows the model to flexibly accommodate preprocessing and natural variants of the method, and makes sense for the intended applications to exponential time algorithms. Our use of parameterization enables us to distinguish between algorithmic methods in a way that a traditional complexity-theoretic approach cannot. Although our approach is coarser than most syntactic approaches, it is more uniform, applying to a variety of algorithmic methods at once, and enables us to get useful information about the relative power of these methods.

Related Previous Work

A large number of problems have been shown to be *Fixed Parameter Tractable (FPT)*, i.e., solvable in time $O^*(f(k))$, where k is a parameter provided with each input, $O^*(\cdot)$ suppresses factors polynomial in the input size, and $f(\cdot)$ is some computable function. For many problems we now know essentially the optimal running time: there is an $O^*(f(k))$ time algorithm and an $O^*(g(k))$ time algorithm for any $g(k) < f(o(k))$ contradicts the Exponential Time Hypothesis (ETH). For a few problems we even know that $O^*(f(k))$ time algorithms cannot be improved to $O^*(f(k)^{1-\Omega(1)})$ time algorithm under stronger hypotheses as the Strong ETH. In this work we are mostly interested in problems for which $f(k) = 2^{\mathrm{poly}(k)}$ - this is the case for most natural FPT parameterizations of NP-complete problems.

Kernelization. A natural paradigm to prove a problem is solvable in $O^*(2^{\mathrm{poly}(k)})$ time is preprocessing plus brute force: given an instance (x, k) of a parameterized problem, transform it in polynomial time to an instance (x', k') of the same problem where $|x'|, k'$ are polynomial in k (this part is called the *polynomial kernel*), and then solve the smaller instance using brute-force search.[1] The power of polynomial kernelization has been extensively investigated, and is by now fairly well understood. For many parameterized problems, we have either found a polynomial kernel, or showed they do not exist unless $NP \subseteq coNP/poly$; the latter is proved by providing an *(OR* or *AND)-composition*, and appealing to results in [3, 13, 11] and related works. This fits as an excellent starting point for our study since it gives a lower bound for a class of exponential-time algorithms modelled via polynomial-time reductions, and is conditional on an hypothesis concerning polynomial-time computation.

Branching. Another heavily used paradigm to solve a problem in $O^*(2^{\mathrm{poly}(k)})$ time is that of *branching*, or *bounded search trees*. A natural model for this paradigm is the model of *One-sided Probabilistic Polynomial (OPP) algorithms* proposed by Paturi and Pudlak [24] in their study of algorithms for satisfiability. OPP algorithms are polynomial-time algorithms with one-sided error which never accept no-instances but only detect yes-instances with small but non-trivial probability (called the *success probability*). An OPP algorithm with success probability $f(n)$ can be converted to a bounded-error randomized algorithm running in time $\mathrm{poly}(n)/f(n)$ just by taking the OR of $f(n)$ independent trials. On the other hand if an exponential-time algorithm can be thought of as traversing an exponential-size recursion tree which performs polynomial-time checks at leaves and returns true if at some leaf true is returned, then we can cast this as an OPP algorithm provided we are able to sample leaves of the branching tree in an efficient, nearly uniform way (in [24], this observation was attributed to Eppstein [12]). We would like to remark that OPP is more powerful than one might think at first sight as it also directly captures, for example, Schöning's algorithm [27].

Concerning lower bounds, Paturi and Pudlak [24] showed that OPP algorithms with success probability significantly better than 2^{-n} for circuit satisfiability on n variables would have unlikely consequences. Particularly relevant for our work is work by Drucker [10] showing a $2^{-n^{1-\epsilon}}$ upper bound of OPP algorithms' success probability for 3-CNF-SAT (for any $\epsilon > 0$), assuming $NP \nsubseteq coNP/poly$.

Several closely-related formalisms of branching algorithms have been proposed in the literature [4, 26, 31]. In the context of parameterized complexity, Marx proposed a study of branching [20, 21] using a model 'BFPT' of branching FPT algorithms.[2] Also relevant is work of Dantsin and Hirsch [8], which discusses a notion closely related to our notion of witness compression in the context of exact algorithms for Satisfiability, and provides lower bounds conditioned on ETH.

[1] That is, try all bit-strings and see if a certificate arises.
[2] That turns out to be equivalent to OPP algorithms with success probability $2^{-O(k)}$.

Our Contribution

In this work, we argue that many contemporary exponential-time algorithms can be rewritten as polynomial-time reductions to succinct version of problems in P, and we also give several concrete results on the applicability of specific algorithmic paradigms to different problems. We outline these results next.

Branching. Our main technical contributions address branching algorithms as modelled by OPP algorithms or equivalently witness compressions (defined below). Building on machinery developed by Drucker [10] we give lower bounds for constructive OPP algorithms. For instance:

▶ **Theorem 1.1.** *If there is a polynomial time algorithm that given a planar graph outputs a maximum independent set of n vertices with probability* $\exp(-O(n^{1-\epsilon}))$ *for some* $\epsilon > 0$*, then* $NP \subseteq coNP/poly$.

Note that $\exp(O(\sqrt{n}))$ time algorithms are known (e.g. [18]), so this indicates that a rich class of branching algorithms is incapable of exploiting planarity for solving independent set. We also give a simple OPP algorithm that actually establishes success probability $\exp(-O(n/\sqrt{\log(n)}))$.

Following a hashing lemma from [24], we observe that having an OPP algorithm with success probability $f(k)$ is equivalent to having a polynomial-time Monte Carlo reduction from the problem at hand to CKT-SAT[3] with $1/\log(f(k))$ input gates. Thus in the generic context sketched in this paper, the succinct problem corresponding to our model of branching is CKT-SAT. If $f(k) = 2^{-\text{poly}(k)}$, there are witnesses for the problem of size poly(k) and we will refer to the polynomial time Monte Carlo reduction as a *polynomial witness compression* since a satisfying solution of the circuit that the reduction outputs can be seen as a witness for the original instance to be a yes-instance. We call a witness compression *Levin* or *constructive* if we can determine a solution of the original problem given a satisfying assignment of the circuit.

We define a type of reduction we call 'constructive AND-composition' that is closely related to AND-compositions from kernelization theory, and show that assuming $NP \not\subseteq coNP/poly$ no parameterized problem can both have a constructive AND-composition and a Levin polynomial witness compression. As one particular application, we use this to *separate* dynamic programming from branching (as modelled via OPP algorithms). Specifically, we show that Independent Set parameterized by pathwidth,[4] which is known to be FPT via a dynamic programming algorithm, does not have Levin polynomial witness compressions unless $NP \not\subseteq coNP/poly$. An important question[5] is how fast this problem can be solved using only polynomial space. In [19], the authors provide an $O^*(2^{O(\mathtt{pw}^2)})$-time and polynomial-space algorithm based on a tradeoff between dynamic programming and Savitch's theorem, but the folklore dynamic programming algorithm uses $O^*(2^{\mathtt{pw}})$ time and space. Our results thus indicate that branching algorithms of the OPP type, a very natural class of polynomial space algorithms, will not be useful here.

We emphasize that the model of OPP algorithms and witness compressions are powerful by observing that problems such as Steiner Tree, Long Path and Directed Feedback

[3] Refer to Section 2 for a definition.
[4] That is, we assume a path decomposition of width `pw` is given as input.
[5] This question first appeared in print in [19], but was explicitly asked before at least in [22].

Vertex Set (DFVS) do have polynomial witness compressions as a consequence of methods from previous works.

Kernelization. The above results on branching have a number of consequences for kernelization theory. To explain these, let us first stress that it seems that if a problem has an AND-composition it seems very likely it also has an constructive AND-composition since all known AND-compositions are known to be constructive.

There has been interest recently in relaxed versions of kernelization, such as OR-kernels, where rather than computing one small instance from the initial instance, we compute a list of instances, at least one of which is in the language if and only if the original instance was. It is easy to see that a polynomial witness compression is a far reaching generalization of OR-kernelization: if a problem has a OR-kernel the witness would indicate which output of the OR-kernel is a yes-instance along with a certificate of this instance being a YES. On the other hand, a problem as CKT-SAT with k input variables is known to not have polynomial kernelization assuming $NP \not\subseteq coNP/poly$ (see e.g., [9]) but trivially has a polynomial witness compression. Our observation thus implies that problems cannot have both constructive AND-compositions and OR-kernelizations simultaneously unless $NP \subseteq coNP/poly$.

Our connection between constructive AND-composition and witness compressions combined with the polynomial witness compressions for Steiner Tree, Long Path and DFVS implies that these problems do not have constructive AND-compositions, which is a clear indication that they do not admit AND-compositions as studied in kernelization theory. We feel this is a useful insight especially for DFVS because the existence of a polynomial compression for this is a major open problem [6], and since we currently only know how to exclude polynomial compressions via AND- and OR-compressions our connection indicates we probably should not look for AND-compressions.

Parity Compression. There are several other important paradigms that in many cases seem essential to known algorithms for various problems, especially to obtain the best known bounds on the function $f(k)$. In [24], the authors mention as examples the paradigms of exponential-time divide-and-conquer; inclusion-exclusion; dynamic programming; group algebra; and Voronoi cell decomposition; and they argue that 'OPP and its generalizations could serve as an excellent starting point for the study of exponential-time algorithms for NP-complete problems in general', although they leave such generalizations unspecified.

We further explore this direction, using our unifying perspective via succinct parameterized problems. Similar to witness compression, we define a notion of "parity compression" corresponding to reducibility to the problem $\oplus$CKT-SAT parameterized by the number of variables. The idea here is that algebraic and inclusion-exclusion based approaches to FPT algorithms often implicitly reduce the problem to a succinctly represented parity of exponentially many input bits, i.e, an instance of $\oplus$CKT-SAT. We illustrate this phenomenon by capturing the Long Path and K-Cycle problems in our model.

Disjunctive Dynamic Programming. We model a subclass of dynamic programming algorithms which we refer to as "disjunctive dynamic programming". Intuitively, this corresponds to dynamic programming tables whose entries are Boolean ORs of lexicographically-prior entries. We model this class via reducibility to the problem CNF-Reach, an instance of which is a directed graph succinctly encoded by a CNF, with the question being whether there is a source-sink path of a prescribed length in the graph. The parameter is the number of variables of the CNF. Essentially, the existence of a path corresponds to a "trace" of a disjunctive dynamic programming algorithm with a YES answer.

More generally, one could study succinctness implemented by circuits rather than CNFs; however, the choice of CNFs has a nice benefit: it allows us to find natural complete problems for our model. Specifically, we show that Independent Set parameterized by pathwidth is complete for this model, thus in some sense dynamic programming is the "right" algorithmic technique for this problem. The completeness of Independent Set parameterized by pathwidth may also be interpreted as another signal that polynomial space algorithms for problems parameterized by pathwidth might be hard to find as they need to exploit the succinctness given by the CNF representation or otherwise need to improve over Savitch's theorem for short reachability. Let us remark that related research has been proposed earlier: the reduction as outlined here has been conjectured in the second author's PhD-thesis [23], and the aforementioned signal was remarked in [23, 2, 25]

Organization. This work is organized as follows: in Section 2 we provide a few preliminaries. Note that due to space constraints we do not cover basic definitions from parameterized complexity such as definitions of fixed-parameter tractability that provide context for our work; we refer the reader to a recent textbook [7]. Section 3 presents our main technical results, which are on branching algorithms. Section 4 introduces the model of parity compression, Section 5 introduces the model of disjunctive dynamic programming and in Section 6 we list a number of interesting directions for further research.

2 Preliminaries and Notation

For an integer p, $[p] := \{1, \dots, p\}$, and $\binom{X}{p}$ denotes the family of size-p subsets of a set X.

Probabilistic Circuits. A *probabilistic circuit* is a (De Morgan) Boolean circuit $C(x, r)$ which, in addition to its input gates $x \in \{0,1\}^n$, has a designated set of "randomness gates" $r \in \{0,1\}^{\mathrm{poly}(n)}$. We say such a circuit computes a function $f(x)$ with success probability $p(n)$ if, for all $x \in \{0,1\}^n$, $\Pr_r[C(x,r) = f(x)] \geq p(n)$. Here the probability is taken over a uniform random setting to r. By Cook's transformation, any polynomial time randomized algorithm can be expressed as a (logspace-uniform) family of polynomial-size probabilistic circuits.

Problem Definitions. *PC* denotes the set of search problems whose solutions can be verified in polynomial time (following [14]). For $L \subseteq \{0,1\}^*$, χ_L denotes the characteristic vector of L. A *parameterized problem* is a set $Q \subseteq \{0,1\}^* \times \mathbb{N}$.

We use the following notation to define (parameterized) (search) problems in NP or PC: if k is some parameter of an unparameterized problem R, R$/k$ denotes the associated problem parameterized by k. When a problem has a natural search version, we will use this to define it, as the decision version follows from the search version. We use L_Q to denote the decision version of a search problem Q. The following parameterized search problems will be important for this paper:

CKT-Sat **Parameter:** n
Instance: A Boolean circuit C on n variables.
Witness: An assignment $x \in \{0,1\}^n$ such that $C(x) = 1$.

(d)-CNF-Sat **Parameter:** n
Instance: A Boolean (d)-CNF-formula C on n variables.
Witness: An assignment $x \in \{0,1\}^n$ such that $C(x) = 1$.

For any search problem $R \in PC$, we define a search problem $AND(R)$ as follows:

$AND(R)$ **Parameter:** n
Instance: Instances $x_1, \dots, x_t \in \{0,1\}^n$
Witness: $y_1, \dots, y_t$ such that $(x_i, y_i) \in R$ for every i.

Reductions. For search problems[6] $Q, R \in PC$, a *Levin reduction* from Q to R consists of two polynomial time algorithms, A_1 and A_2, such that (i) $\exists y : (x, y) \in Q$ if and only if $\exists y' : (A_1(x), y') \in R$, (ii) if $(A_1(x), y') \in R$, then $(x, A_2(x, y')) \in Q$. A *Monte Carlo reduction* from language L to language L' is a randomized polynomial time algorithm that takes $x \in \{0,1\}^*$ as input and outputs $y \in \{0,1\}^*$ such that (i) if $x \notin L$ then $y \notin L'$, (ii) if $x \in L$ then $\Pr[y \in L'] \geq 1/4$. A *Levin Monte Carlo reduction* from search problem Q to search problem R is a pair of two randomized polynomial time algorithms A and B with the following properties: (i) A is a Monte Carlo reduction from L_Q to L_R mapping x to x', (ii) B takes as input x, x' and y', and if $(x', y') \in R$, then with probability $1/4$, B outputs y such that $(x, y) \in Q$.

Success Probability of Polynomial Time Algorithms. Let $f : \mathbb{N} \times \mathbb{N} \to \mathbb{R}$. We say that an algorithm *solves a parameterized problem Q with success probability f*, if given (x, k) it returns NO if $(x, k) \notin L_Q$ and YES with probability at least $f(|x|, k)$ if $(x, k) \in L_Q$. Moreover, if $Q \in PC$, *it finds solutions for* Q with probability at least f if given (x, k) it returns NO if $(x, k) \notin L_Q$ and it returns a certificate for $(x, k) \in L_Q$ with probability at least $f(|x|, k)$, otherwise. Note that an algorithm finding solutions for Q also solves Q.

By standard boosting arguments we see that if there is a polynomial time algorithm solving Q or finding solutions for Q with probability at least f, then for any polynomial p there is also a polynomial time algorithm solving Q or finding solutions for Q with probability at least $\min\{\frac{1}{2}, p(|x|) f(|x|, k)\}$. Therefore, if $f(|x|, k)$ is $1/(\mathrm{poly}(|x|) f(k))$, we say it solves or finds solutions for Q with probability at least $f'(k)$ where $f'(k) = f(1, k)$.

Non-deterministic Direct Product Reductions. For a function $f : A \to B$ and integer t, we denote $f^{\otimes t} : A^t \to B^t$ to be the t-fold direct product of f, e.g., for $x_1, \dots, x_t \in A$ we let $f^{\otimes t}(x_1, \dots, x_t) = (f(x_1), \dots, f(x_t))$. The following result will be crucial for this work:

▶ **Theorem 2.1** (Theorem 1.2 of [10]). *Let $f = \{f_N\}$ be a family of Boolean functions on N input bits, and suppose that $f \notin NP/poly \cap coNP/poly$. Let $100 \leq t(N) \leq \mathrm{poly}(N)$ be a parameter and let $\{C_N\}_{N>0}$ be any family of polynomial-size probabilistic circuits outputting $t(N)$ bits. Then for infinitely many choices of N and $x \in \{0,1\}^{N \times t(N)}$,*

$$\Pr[C_N = f_N^{\otimes t(N)}(x)] < \exp(-\Omega(t(N)). \tag{2.1}$$

[6] In this work, we implicitly cast a parameterized (search) problem as a normal (search) problem by omitting the parameter where convenient.

3 Branching via OPP Algorithms and Witness Compressions

In this section we present our results on branching algorithms. We first formally define the notion of constructive AND-compositions and state how they exclude OPP algorithms. Then we formally introduce witness compressions and show their close relation with OPP algorithms. Subsequently, we point out implications to parameterized complexity.

Constructive AND-Compositions and Their Consequences.

▶ **Definition 3.1** (Constructive AND-composition). Let L be a search problem, Q be a parameterized search problem and d be a constant. We say that a pair of algorithms (A,B) is a *constructive AND-composition of degree d from L into Q* if the following conditions hold:

1. A is given $x_1, x_2, \dots, x_t$ and outputs an instance $(x, k) \in \Sigma^* \times \mathbb{N}$ in time polynomial in $\sum_{i=1}^{t} |x_i|$ such that $k \leq \mathrm{poly}(\max_i |x_i| \log(t))$ and $|x| \leq \mathrm{poly}(\max_i |x_i| \log(t))t^d$,
2. if for every i there exist y_i such that $(x_i, y_i) \in L$, then B does the following: B takes as input $x_1, x_2, \dots, x_t$, the instance (x, k), and a certificate y such that $(x, k, y) \in Q$, and outputs y_i for every i such that

$$\Pr[\forall i : (x_i, y_i) \in L] \geq \exp\left(-\mathrm{poly}\left(\max_i |x_i|\right) \log(t)\right).$$

This is closely related to AND-compositions as studied in kernelization complexity (see e.g. [7, Section 15.1.3]): it is more strict in the sense that the reduction needs to be Levin, but more general in the sense that we only need a weak probabilistic guarantee on the output. We will see that even constructive AND-compositions of degree 1 with trivial parameterizations have interesting consequences.

▶ **Theorem 3.2.** *If there is a constructive AND-composition of degree d from a PC-hard search problem L into a parameterized search problem Q, then no polynomial time algorithm finds solutions for every instance (x, k) of Q with probability* $\exp(-\mathrm{poly}(k)|x|^{1/d-\Omega(1)})$*, unless $NP \subseteq coNP/poly$.*

As one concrete application we obtain the Theorem as mentioned in the introduction:

▶ **Theorem 1.1** (restated). *If there is a polynomial time algorithm that, given a planar graph, outputs a maximum independent set of n vertices with probability* $\exp(-n^{1-\epsilon})$ *for some* $\epsilon > 0$*, then $NP \subseteq coNP/poly$.*

Proof. Let L be the following search problem: given the adjacency list of a planar graph G and integer θ, find an independent set of G of size at least θ. The decision variant of this problem NP-complete and by inspecting the known reductions, the problem is also seen to be PC-complete. Let Q be L with a trivial parameterization (e.g., the parameter equals 1). We now give a constructive AND-composition of degree 1 from L to Q. Given instances $(G_1 = (V_1, E_1), \theta_1), \dots, (G_t = (V_t, E_t), \theta_t)$, create an instance (G, θ^*) of Q where G is the disjoint union of $G_1, \dots, G_t$ (i.e. it has each graph G_i as a connected component in it), and θ^* is picked uniformly at random from $\{1, \dots, \sum_{i=1}^{t} |V_i|\}$. We see that with probability $1/\sum_{i=1}^{t} |V_i| \geq \exp\left(-\mathrm{poly}\left(\max_i |x_i|\right) \log(t)\right)$, we have that θ^* equals the size of the maximum independent set. Moreover, if we are given a maximum independent set of G, its intersection with every component must be a maximum independent set in that component so if all instances are YES instances we find maximum independent sets of size at least θ_i in G_i for every i. Since (G, θ) is represented with $\mathrm{poly}(\max_i |V_i|)t \log(t)$ bits, we therefore found a constructive AND-composition of degree 1, and no polynomial time algorithm finds solutions for Q with probability $\exp(-|x|^{1-\Omega(1)})$ by Theorem 3.2. This implies the statement since $|x|$ is $n \log n$ for n-vertex graphs. ◀

We remark the naïve guessing procedure here is not optimal (the proof is postponed to the full version):

▶ **Theorem 3.3.** *There exists a polynomial time algorithm that outputs a maximum independent set of a planar graph on n vertices with probability* $\exp(-n/\sqrt{\log n})$.

Witness Compressions. We will now give an equivalent interpretation of OPP algorithms that paves the way for defining models of other paradigms in the next sections.

▶ **Definition 3.4** ((Levin) Witness Compression)**.** A *(Levin) $h(k,N)$-witness compression* for a parameterized (search) problem Q is a (Levin) Monte-Carlo reduction from Q to CKT-SAT that maps (x,k) with $|x| = N$ to (y,n) with $n \le h(k,N)$.

Note that having a $h(k,N)$-witness compression is equivalent to having a $h(k, N+\lg(N))$-witness compression since we can brute-force over all assignments of $\lg(N)$ input bits in polynomial time. We say a (Levin) $h(k,N)$-witness compression is *polynomial* if $h(k,N) \le \text{poly}(k)$ (or equivalently $\text{poly}(k) + \log(N)$). The following lemma shows the equivalence of witness compression and OPP algorithms.

▶ **Lemma 3.5.** *A parameterized (search) problem has a* (Levin) $h(k,N)$-witness compression *if and only if there is a polynomial time algorithm solving it (respectively, finding solutions) with success probability at least* $2^{-h(k,N)}$.

The proof is postponed to the full version. The forward direction in both variants is immediate. For the backward direction, we use the 'Hash-Down lemma' from [24] to prove both variants. Our proof of the equivalence takes advantage of the fact that we allow randomized reductions in the definition of witness compression. If we were to only allow deterministic reductions in the definition, the equivalence would still hold under a sufficiently strong derandomization hypothesis - we omit the details.

We emphasize the power of polynomial witness compression by revisiting a few FPT-algorithms and observing that they give rise to efficient witness compressions. Marx [21] observes that VERTEX COVER and FEEDBACK VERTEX SET have a witness compression with $h(k)$ linear. Here, we add a few non-trivial witness compressions to this list with $h(k)$ quasi-linear. The relevant problem statements and proof of the following theorem can be found in the full version. All these results go via the connection from Theorem 3.5.

▶ **Theorem 3.6.** STEINER TREE *and* LONG PATH *have Levin $O(k \log k)$-witness compressions, and* DFVS *has a Levin $O(k \log^3(k))$-witness compression.*

Implications to Parameterized Complexity. As mentioned in the introduction, polynomial witness compression appears significantly more powerful than polynomial kernelization. Indeed if a problem has an OR-kernelization,[7] it is easily seen we have a polynomial witness compression:

[7] Where rather than computing one small instance from the initial instance as in kernelization, we compute a list of instances at least one of which is in the language if and only if the original instance was, see e.g.[17] where the name 'disjunctive kernelization' was used.

▶ **Observation 3.7.** If Q admits a polynomial (Levin) OR-kernelization to a problem in NP, it admits a polynomial (Levin) witness compression.

On the other hand, let us remark here that there may be problems in NP that admit polynomial compressions[8] but no polynomial witness compression: Wahlström [30] gives an interesting polynomial compression of the K-cycle problem (see the full version for the problem definition) to a language that is not in NP, and remarks that this seems to separate polynomial kernelization from polynomial compression since it is not clear whether K-cycle has polynomial witness compressions.

The above connection is relevant for kernelization complexity because Theorem 3.8 suggests that parameterized problems with AND-compositions have no OR-kernelizations. Another interesting consequence obtained by combining Theorem 3.2 and Theorem 3.6 is (since the problems at hand are easily seen to be PC-complete):

▶ **Theorem 3.8.** Steiner Tree*,* Long Path*, and* DFVS *do not admit constructive AND-compositions unless $NP \subseteq coNP/poly$.*

As mentioned before, this is a useful fact especially for DFVS because the existence of a polynomial compressions for this is a big open problem [6], and we currently only know how to exclude polynomial compressions via AND- and OR-compressions Theorem 3.8. So this indicates researchers attacking this open problem probably should not look for AND-compressions.

Another useful implication concerns the following parameterized problem:

Independent Set (IS/pw) **Parameter:** `pw`
Instance: A graph G, path decomposition of G of width `pw`, integer θ.
Witness: An independent set of G of size at least θ.

As mentioned in the introduction, it is an important open question how fast this problem can be solved using only polynomial space. We show that branching algorithms (which is a subset of all polynomial space algorithms) are not useful here:

▶ **Theorem 3.9.** *Suppose a polynomial time algorithm takes as input a path decomposition of width* `pw` *of a graph G on n vertices, and outputs with probability* $\exp(-\text{poly}(\texttt{pw})n^{1-\Omega(1)})$ *a maximum independent set of G, then $NP \subseteq coNP/poly$.*

Since the proof is very similar to the proof of Theorem 1.1, it is postponed to the full version. Let us remark that several other interesting graph problems admit constructive AND-compositions. For example, following Lemma 7 from [3] we have that

▶ **Observation 3.10.** Let L be a parameterized graph search problem such that for any pair of graphs G_1 and G_2, and integer $k \in \mathbb{N}$, $(G_1, k) \in L \wedge (G_2, k) \in L \leftrightarrow (G_1 \cup G_2, k)$ where $G_1 \cup G_2$ is the disjoint union of G_1 and G_2. Then L admits a constructive AND-composition of degree 1.

Similar as in [3], this implies hardness for several problems. We refer to [3] for the definitions of these problems since our only goal is to point out the applicability of our framework.

[8] A compression is a kernelization where the target problem might be different (and crucially here, not even in NP).

▶ **Theorem 3.11.** *No polynomial time algorithm finds solutions for any instance* (x, k) *of* Cutwidth, Modified Cutwidth, Pathwidth, Branchwidth, Search Number, Gate Matrix Layout, *and* Front Size *with probability* $\exp(-\mathrm{poly}(k)|x|^{1-\Omega(1)})$ *unless* $NP \not\subseteq coNP/poly$.

Proof. Following [3], we have by Observation 3.10 that all the above problems admit constructive AND-compositions. By inspection it can be seen that the reductions from these problems to CKT-Sat (which exist since all the above problems are NP-complete) are all Levin reductions, thus all problems are PC-complete. The claim follows from Theorem 3.2. ◀

4 Parity Compression

As mentioned before, witness compression tightly captures a large part of contemporary FPT-algorithms, but still far from all of them. Motivated by this, we propose the following natural generalization of witness compression, based on the definition of witness compression as a reduction to CKT-Sat. A *parity compression* is a polynomial time Monte Carlo reduction from the problem at hand to the $\oplus$CKT-Sat problem, defined as follows:

$\oplus$CKT-Sat **Parameter:** n
Instance: A Boolean circuit C on n variables
Asked: Whether the parity of the size of the set $\{x \in \{0,1\}^n : C(x) = 1\}$ is odd.

Analogous to witness compressions, we can interpret parity compressions as exponential time algorithms by solving the resulting $\oplus$CKT-Sat instance in time $2^n|x|^{O(1)}$ (the analogue of witness compressions was to solve the CKT-Sat instance by simple brute-force enumeration). By an easy application of the Isolation Lemma of [29], there is a polynomial time Monte Carlo reduction from CKT-Sat to $\oplus$CKT-Sat that increases the number of input variables by $O(\mathrm{polylog}(|C|))$. Thus parity compression is a generalization of witness compression. No polynomial-time reduction in the reverse direction is known, and such a reduction (even randomized) would imply a collapse of PH in light of Toda's theorem [28].

While we are not yet able to show lower bounds for parity compressions since its study is still in its infancy, we do argue in the full version that several interesting contemporary algorithms (mainly, ones using inclusion/exclusion or group algebra) are exponential parities. This motivates a very interesting future research direction:

▶ **Open Problem 1.** Find non-trivial evidence against a polynomial time Monte Carlo reduction from CKT-Sat on n-variable circuits to $\oplus$CKT-Sat on n' circuits where $n' << n$.

Another goal would be to further exclude more polynomial space paradigms that are able to solve Independent Set parameterized by the pathwidth:

▶ **Open Problem 2.** Find evidence against a polynomial time Monte Carlo reduction from IS/pw to $\oplus$CKT-Sat where $n = o(\mathtt{pw}^2)$.

5 Disjunctive Dynamic Programming

One natural other algorithmic paradigm unaddressed so far (as highlighted in Theorem 3.9 and Open Problem 2) is dynamic programming. We focus in this work on a subclass of dynamic programming algorithms which we call 'disjunctive dynamic programming' - this corresponds to dynamic programming tables where the entries are Boolean ORs of previous entries. Specifically, we say a disjunctive dynamic programming algorithm is a polynomial time parameter reduction to the following problem:

CNF-Reach **Parameter:** n
Instance: A CNF-formula $\varphi : \{0,1\}^n \to \{0,1\}$ with m clauses and n even, integer $\ell = \text{poly}(n)$.
Witness: $x^1, \dots, x^\ell \in \{0,1\}^{n/2}$ with $x^1 = 0 \cdots 0, x^\ell = 1 \cdots 1$ and $\varphi(x^i x^{i+1}) = 1$ for every $0 \leq i \leq \ell - 1$.

In the full version we show that IS/PW is almost equivalent to CNF-Reach[9] by giving almost tight reductions between the two problems. Thus IS/PW can be seen as complete for the class of problems efficiently solvable with disjunctive dynamic programming. We feel such a reduction expresses the hardness of a problem typically solved with dynamic programming better than e.g., a reduction to CNF-Sat or even CKT-Sat since these problem do have small witnesses and polynomial-space algorithms. Next to Theorem 3.9, this may be seen as additional evidence that finding fast space-efficient algorithms for IS/PW might be very hard (e.g., we either need to exploit the succinct representation via CNF-formula's or find new algorithms for the directed reachability problem).

We also show that an algorithm for Set Cover is a disjunctive dynamic programming algorithm: we reduce Set Cover to CNF-Reach in the full version.

6 Directions for Further Research

We conclude this paper with a few open questions. First, for several problems, the existence of polynomial witness compression is open (see the full version for missing problem definitions):

▶ **Open Problem 3.** Do Subset Sum, Knapsack, Knapsack/Weight-Value, K-Cycle or Disjoint Paths have polynomial witness compressions?

Note that currently, it is not clear whether there exists a parameterized problem that has a polynomial compression but no polynomial witness compression, although as suggested in [30] the K-Cycle would be a good candidate for such a problem.

One algorithmic paradigm not addressed is exponential time divide and conquer [15], which is also closely related to applications of Savitch's Theorem as used by [19]:

▶ **Open Problem 4.** Is there a good model of exponential time divide and conquer based on reductions to a succinct version of a natural problem? Can it solve IS/PW in $O^*(2^{o(\texttt{pw}^2)})$ time?

Ambitiously, having finer-grained lower bounds would be very insightful:

▶ **Open Problem 5.** Can we rule out linear witness compressions for some problems with quasilinear witness compressions under standard assumptions?

References

1 Michael Alekhnovich, Allan Borodin, Joshua Buresh-Oppenheim, Russell Impagliazzo, Avner Magen, and Toniann Pitassi. Toward a model for backtracking and dynamic programming. In *20th Annual IEEE Conference on Computational Complexity (CCC 2005),*

[9] Such a reduction was already conjectured in the PhD-thesis of the second author [23, Page 35]

11-15 June 2005, San Jose, CA, USA, pages 308–322. IEEE Computer Society, 2005. doi:10.1109/CCC.2005.32.

2 Eric Allender, Shiteng Chen, Tiancheng Lou, Periklis A. Papakonstantinou, and Bangsheng Tang. Width-parametrized SAT: time–space tradeoffs. *Theory of Computing*, 10:297–339, 2014. doi:10.4086/toc.2014.v010a012.

3 Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009. doi:10.1016/j.jcss.2009.04.001.

4 Liming Cai and Jianer Chen. On the amount of nondeterminism and the power of verifying. *SIAM Journal on Computing*, 26(3):733–750, 1997. doi:10.1137/S0097539793258295.

5 Michele Conforti, Gérard Cornuéjols, and Giacomo Zambelli. Extended formulations in combinatorial optimization. *4OR*, 8(1):1–48, 2010. doi:10.1007/s10288-010-0122-z.

6 Marek Cygan, Fedor Fomin, Bart M.P. Jansen, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, and Saket Saurabh Michal Pilipczuk. Open problems for fpt school 2014.

7 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.

8 Evgeny Dantsin and Edward A. Hirsch. Satisfiability certificates verifiable in subexponential time. In Karem A. Sakallah and Laurent Simon, editors, *Theory and Applications of Satisfiability Testing – SAT 2011 – 14th International Conference, SAT 2011, Ann Arbor, MI, USA, June 19-22, 2011. Proceedings*, volume 6695 of *Lecture Notes in Computer Science*, pages 19–32. Springer, 2011. doi:10.1007/978-3-642-21581-0_4.

9 Holger Dell and Dieter van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. *J. ACM*, 61(4):23:1–23:27, 2014. doi:10.1145/2629620.

10 Andrew Drucker. Nondeterministic direct product reductions and the success probability of SAT solvers. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 736–745. IEEE Computer Society, 2013. doi:10.1109/FOCS.2013.84.

11 Andrew Drucker. New limits to classical and quantum instance compression. *SIAM J. Comput.*, 44(5):1443–1479, 2015. doi:10.1137/130927115.

12 David Eppstein. Quasiconvex analysis of multivariate recurrence equations for backtracking algorithms. *ACM Trans. Algorithms*, 2(4):492–509, October 2006. doi:10.1145/1198513.1198515.

13 Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct PCPs for NP. *J. Comput. Syst. Sci.*, 77(1):91–106, 2011. doi:10.1016/j.jcss.2010.06.007.

14 Oded Goldreich. *Computational complexity – a conceptual perspective*. Cambridge University Press, 2008.

15 Yuri Gurevich and Saharon Shelah. Expected computation time for hamiltonian path problem. *SIAM J. Comput.*, 16(3):486–502, 1987. doi:10.1137/0216034.

16 Paul Helman. A common schema for dynamic programming and branch and bound algorithms. *Journal of the ACM*, 36(1):97–128, 1989.

17 Stefan Kratsch. Recent developments in kernelization: A survey. *Bulletin of the EATCS*, 113, 2014. URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/285.

18 Richard J. Lipton and Robert Endre Tarjan. Applications of a planar separator theorem. *SIAM J. Comput.*, 9(3):615–627, 1980. doi:10.1137/0209046.

19 Daniel Lokshtanov, Matthias Mnich, and Saket Saurabh. Planar k-path in subexponential time and polynomial space. In Petr Kolman and Jan Kratochvíl, editors, *Graph-Theoretic*

Concepts in Computer Science – 37th International Workshop, WG 2011, Teplá Monastery, Czech Republic, June 21-24, 2011. Revised Papers, volume 6986 of *Lecture Notes in Computer Science*, pages 262–270. Springer, 2011. doi:10.1007/978-3-642-25870-1_24.

20 Dániel Marx. What's next? reductions other than kernelization. Talk at WorKer 2010: Workshop on Kernelization, Leiden, The Netherlands, 2010.

21 Dániel Marx. What's next? future directions in parameterized complexity. In Hans L. Bodlaender, Rod Downey, Fedor V. Fomin, and Dániel Marx, editors, *The Multivariate Algorithmic Revolution and Beyond – Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday*, volume 7370 of *Lecture Notes in Computer Science*, pages 469–496. Springer, 2012. doi:10.1007/978-3-642-30891-8_20.

22 Jesper Nederlof. Saving space by algebraization. Talks at seminars in Utrecht (January 8) and UiB (Februari 11), UiB ICT Research school (May 18), STOC, Dagstuhl 'Exact Complexity of NP-hard Problems', 2010.

23 Jesper Nederlof. *Space and Time Efficient Structural Improvements of Dynamic Programming Algorithms.* PhD thesis, University of Bergen, 2011. URL: http://www.win.tue.nl/~jnederlo/PhDThesis.pdf.

24 Ramamohan Paturi and Pavel Pudlák. On the complexity of circuit satisfiability. In Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 241–250. ACM, 2010. doi:10.1145/1806689.1806724.

25 Michal Pilipczuk and Marcin Wrochna. On space efficiency of algorithms working on structural decompositions of graphs. In Nicolas Ollinger and Heribert Vollmer, editors, *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, volume 47 of *LIPIcs*, pages 57:1–57:15. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.STACS.2016.57.

26 Rahul Santhanam. On separators, segregators and time versus space. In *Computational Complexity, 16th Annual IEEE Conference on, 2001.*, pages 286–294, 2001. doi:10.1109/CCC.2001.933895.

27 Uwe Schöning. A probabilistic algorithm for k-sat and constraint satisfaction problems. In *40th Annual Symposium on Foundations of Computer Science, FOCS'99, 17-18 October, 1999, New York, NY, USA*, pages 410–414. IEEE Computer Society, 1999. doi:10.1109/SFFCS.1999.814612.

28 Seinosuke Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991. doi:10.1137/0220053.

29 Leslie G. Valiant and Vijay V. Vazirani. NP is as easy as detecting unique solutions. *Theor. Comput. Sci.*, 47(3):85–93, 1986. doi:10.1016/0304-3975(86)90135-0.

30 Magnus Wahlström. Abusing the tutte matrix: An algebraic instance compression for the K-set-cycle problem. In Natacha Portier and Thomas Wilke, editors, *30th International Symposium on Theoretical Aspects of Computer Science, STACS 2013, February 27 – March 2, 2013, Kiel, Germany*, volume 20 of *LIPIcs*, pages 341–352. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2013. URL: http://www.dagstuhl.de/dagpub/978-3-939897-50-7, doi:10.4230/LIPIcs.STACS.2013.341.

31 Ryan Williams. Inductive time-space lower bounds for sat and related problems. *computational complexity*, 15(4):433–470, 2006.

On the Power of Advice and Randomization for Online Bipartite Matching*

Christoph Dürr[1], Christian Konrad[2], and Marc Renault[3]

1 Sorbonne Universités, UPMC Univ Paris 06, CNRS, LIP6, Paris, France
Christoph.Durr@lip6.fr
2 Reykjavik University, Reykjavik, Iceland
christiank@ru.is
3 IRIF, CNRS, Université Paris Diderot, Paris, France
mrenault@liafa.univ-paris-diderot.fr

Abstract

While randomized online algorithms have access to a sequence of uniform random bits, deterministic online algorithms with advice have access to a sequence of *advice bits*, i.e., bits that are set by an all-powerful oracle prior to the processing of the request sequence. Advice bits are at least as helpful as random bits, but how helpful are they? In this work, we investigate the power of advice bits and random bits for online maximum bipartite matching (MBM).

The well-known Karp-Vazirani-Vazirani algorithm [24] is an optimal randomized $(1-\frac{1}{e})$-competitive algorithm for MBM that requires access to $\Theta(n \log n)$ uniform random bits. We show that $\Omega(\log(\frac{1}{\epsilon})n)$ advice bits are necessary and $O(\frac{1}{\epsilon^5}n)$ sufficient in order to obtain a $(1-\epsilon)$-competitive deterministic advice algorithm. Furthermore, for a large natural class of deterministic advice algorithms, we prove that $\Omega(\log \log \log n)$ advice bits are required in order to improve on the $\frac{1}{2}$-competitiveness of the best deterministic online algorithm, while it is known that $O(\log n)$ bits are sufficient [9].

Last, we give a randomized online algorithm that uses cn random bits, for integers $c \geq 1$, and a competitive ratio that approaches $1-\frac{1}{e}$ very quickly as c is increasing. For example if $c = 10$, then the difference between $1-\frac{1}{e}$ and the achieved competitive ratio is less than 0.0002.

1998 ACM Subject Classification F.1.2 Online computation, G.2.2 Graph algorithms

Keywords and phrases On-line algorithms, Bipartite matching, Randomization

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.37

1 Introduction

Online Bipartite Matching. The maximum bipartite matching problem (MBM) is a well-studied problem in the area of online algorithms [24, 5, 12]. Let $G = (A, B, E)$ be a bipartite graph with $A = [n] := \{1, \dots, n\}$ and $B = [m]$, for some integers n, m. We assume $m = \Theta(n)$ allowing bounds to be stated as simple functions of n rather than of n and m. The A-vertices together with their incident edges arrive online, one at a time, in some adversarial chosen order $\pi : [n] \to [n]$. Upon arrival of a vertex $a \in A$, the online algorithm has to irrevocably decide to which of its incident (and yet unmatched) B-vertices it should be matched. The considered quality measure is the well-established *competitive ratio* [32], where the performance of an

* Research supported in part by the ANR projects ANR-11-BS02-0015, ANR-15-CE40-0015, ANR-12-BS02-005, by the Icelandic Research Fund grants-of-excellence no. 120032011 and 152679-051 and by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme no. 648032.

online algorithm is compared to the performance of the best offline algorithm: A randomized online algorithm $\mathbf{A}$ for MBM is c-competitive if the matching M output by $\mathbf{A}$ is such that $\mathbb{E}|M| \geq c \cdot |M^*|$, where the expectation is taken over the random coin flips, and M^* is a maximum matching.

In 1990, Karp, Vazirani and Vazirani [24] initiated research on online MBM and presented a $(1 - \frac{1}{e})$-competitive randomized algorithm denoted KVV. It chooses a permutation $\sigma : [m] \rightarrow [m]$ of the B-vertices uniformly at random and then runs the algorithm Ranking(σ), which matches each incoming A-vertex a to the free incident B-vertex b of minimum rank (i.e., $\sigma(b) < \sigma(c)$ for all free incident vertices $c \neq b$). If there is no free B-vertex, then a remains unmatched. They showed that no online algorithm has a better competitive ratio than $1 - \frac{1}{e}$, implying that KVV is optimal. For deterministic online algorithms, it is well-known that the Greedy matching algorithm, which can be seen as running Ranking(σ) using a fixed arbitrary σ, is $\frac{1}{2}$-competitive, and is optimal for the class of deterministic online algorithms.

Improving on $1 - \frac{1}{e}$. Additional assumptions are needed in order to improve on the competitive ratio $1 - \frac{1}{e}$. For example, Feldman et al. [17] introduced the online stochastic matching problem, where a bipartite graph $G' = (A', B', E')$ and a probability distribution $\mathcal{D}$ is given to the algorithm. The request sequence then consists of vertices of A' that are drawn according to $\mathcal{D}$. Feldman et al. showed that the additional knowledge can be used to improve the competitive ratio to 0.67, which has subsequently been further improved [3, 27]. Another example is a work by Mahdian and Yan [26], who considered the classical online bipartite matching problem with a random arrival order of vertices. They analysed the KVV algorithm for this situation and proved that it is 0.696-competitive.

Online Algorithms with Advice. It is a common theme in online algorithms to equip an algorithm with additional knowledge that allows it to narrow down the set of potential future requests and, thus, design algorithms that have better competitive ratios as compared to algorithms that have no knowledge about the future. Additional knowledge can be provided in many different ways, e.g. access to lookahead [22, 19], probability distributions about future requests [17, 26], or even by giving an isomorphic copy of the input graph to the algorithm beforehand [21]. Dobrev et al. [13] and later Emek et al. [15] first quantified the amount of additional knowledge (advice) given to an online algorithm in an information theoretic sense. They showed that a specific problem requires at least $b(n)$ bits of advice, for some function b, in order to achieve optimality [13] or in order to achieve a particular competitive ratio [15]. Advice lower bounds are meaningful in practice as they apply to any potential type of additional information that could be given to an algorithm.

In the advice model, a computationally all-powerful oracle is given the entire request sequence and computes an advice string that is provided to the algorithm. Algorithms with advice are not usually designed with practical considerations in mind but to show a theoretical limit on what can be done. As such, the algorithms are often impractical due to the nature of the advice or the complexity in calculating the advice. However, from a theoretical perspective, advice algorithms are necessary to determine the exact advice complexity of online problems (how many advice bits are necessary and sufficient) and thus provide limits on the achievable and more practically relevant lower bounds.

Our Objective and Previous Results. Our objectives are to determine the advice complexity of MBM and to investigate the power of random and advice bits for this problem.

A starting point is a result of Böckenhauer et al. [9], who gave a method that allows the transformation of a randomized online algorithm into a deterministic one with advice with a similar approximation ratio. More precisely, given a randomized online algorithm $\mathbf{A}$ for a minimization problem $\mathcal{P}$ with approximation factor c and possible inputs $\mathcal{I}(n)$ of length n, Böckenhauer et al. showed that a $(1+\epsilon)c$-competitive deterministic online algorithm $\mathbf{B}$ with $\log n + 2\log\log n + \log \frac{\log|\mathcal{I}(n)|}{\log(1+\epsilon)}$ bits[1] of advice can be deduced from $\mathbf{A}$, for any $\epsilon > 0$, where log is the binary logarithm in this paper. The calculation of the advice and the computations executed by $\mathbf{B}$ require exponential time, since $\mathbf{A}$ has to be simulated on all potential inputs $\mathcal{I}(n)$ on all potential random coin flips.

The technique of Böckenhauer et al. [9] can also be applied to maximization problems such as MBM[2]. Applied to the KVV algorithm, we obtain:

▶ **Theorem 1.** *There is a deterministic online algorithm with* $\mathrm{O}(\log n)$ *bits of advice for* MBM *with competitive ratio* $(1-\epsilon)(1-1/e)$, *for any* $\epsilon > 0$.

This result is complemented by a recent result of Mikkelsen [28], who showed that for *repeatable problems* (see [28] for details) such as MBM, no deterministic online algorithm with advice sub-linear in n has a substantially better competitive ratio than any randomized algorithm without advice. Thus, using $\mathrm{O}(\log n)$ advice bits, a $(1-\epsilon)(1-\frac{1}{e})$-competitive deterministic algorithm can be obtained, and no algorithm using $o(n)$ advice bits can substantially improve on this result. Furthermore, Miyazaki [29] showed that $\Theta(\log(n!)) = \Theta(n\log n)$ advice bits are necessary and sufficient in order to compute a maximum matching.

Our Results on Online Algorithms with Advice. Consider a deterministic online algorithm with $f(n)$ bits of advice for MBM. Our previous exposition of related works shows that the ranges $f(n) \in \Omega(\log n) \cap o(n)$ and $f(n) \in \Theta(n\log n)$ are well understood. In this work, we thus focus on the ranges $f(n) \in o(\log n)$ and $f(n) \in \Omega(n) \cap o(n\log n)$. Our first set of results concerns $(1-\epsilon)$-competitive deterministic advice algorithms. We show:

1. There is a deterministic $(1-\epsilon)$-competitive online algorithm, using $\mathrm{O}(\frac{1}{\epsilon^5}n)$ advice bits for MBM.
2. Every deterministic $(1-\epsilon)$-competitive online algorithm for MBM uses $\Omega(\log(\frac{1}{\epsilon})n)$ bits of advice.

Our lower bound result is obtained by a reduction from the *string guessing game* of Böckenhauer et al. [6], a problem that is difficult even in the presence of a large number of advice bits. This technique has repeatedly been applied for obtaining advice lower bounds, e.g. [1, 20, 10, 2, 11, 4]. Our algorithm simulates an augmenting-paths-based algorithm by Eggert et al. [14], that has originally been designed for the data streaming model, with the help of advice bits. It is fundamentally different to the KVV algorithm, however, inspired by the simplicity of KVV, we are particularly interested in the following class of algorithms:

▶ **Definition 2** (Ranking-algorithm). An online algorithm $\mathbf{A}$ for MBM is called Ranking-algorithm if it follows the steps: (1) Determine a ranking σ; (2) Return Ranking(σ).

The KVV algorithm is a Ranking-algorithm, where in step (1), the permutation σ is chosen uniformly at random. The algorithm described in Theorem 1 is a deterministic Ranking-algorithm with $\mathrm{O}(\log n)$ bits of advice that computes the permutation σ from the

[1] Throughout the paper, logarithms, where the base is omitted, are implicitly binary logarithms.
[2] It is straightforward to adapt the proof of Theorem 5 of [9] accordingly. For completeness, a proof is given in the full version of this paper.

available advice bits. While we cannot answer the question how many advice bits are needed for deterministic online algorithms in order to obtain a competitive ratio strictly larger than $\frac{1}{2}$ (and thus to improve on Greedy), we make progress concerning Ranking algorithms:

3. Every Ranking-algorithm that chooses σ from a set of at most $C \log \log n$ permutations, for a small constant C, has approximation factor at most $(\frac{1}{2} + \delta)$, for any $\delta > 0$.

The previous result implies that every $(\frac{1}{2} + \delta)$-competitive deterministic online Ranking-algorithm requires $\Omega(\log \log \log n)$ advice bits.

Next, since the computation of the advice and the algorithm of Theorem 1 are not efficient, we are interested in fast and simple Ranking algorithms. We identify a subclass of Ranking algorithms, denoted Category algorithms, that leads to interesting results, both as deterministic algorithms with advice and randomized algorithms without advice.

▶ **Definition 3** (Category-algorithm). A Ranking-algorithm **A** is called a Category-algorithm if it follows the steps:

- Determine a category function $c : B \to \{1, 2, 3, \dots, 2^k\}$ for some integer $k \geq 1$ with $2^k < m$;
- Let $\sigma_c : [m] \to [m]$ be the unique permutation of the B-vertices such that for two vertices $b_1, b_2 \in B : \sigma_c(b_1) < \sigma_c(b_2)$ if and only if $c(b_1) < c(b_2)$ or $(c(b_1) = c(b_2)$ and $b_1 < b_2)$.
- Return Ranking(σ_c).

Categories can be seen as coarsened versions of rankings, where multiple items with adjacent ranks are grouped into the same category and within a category, the natural ordering by vertex identifier is used. We prove the following:

4. There is a deterministic $\frac{3}{5}$-competitive online Category-algorithm, using m bits of advice (and thus two categories).

The oracle determines the categories depending on whether a B-vertex would be matched by a run of Greedy. We believe that this type of advice is particularly interesting since it does not require the oracle to compute an optimal solution.

Our Results on Randomized Algorithms. Last, we consider randomized algorithms with limited access to random bits. The KVV-algorithm selects a permutation σ uniformly at random, and, since there are $m!$ potential permutations, $\log(m!) = \Theta(m \log m)$ random bits are required in order to obtain a uniform choice. We are interested in randomized algorithms that employ fewer random bits. We consider the class of randomized Category-algorithms, where the categories of the B-vertices are chosen uniformly at random. We show:

5. There is a randomized Category-algorithm using km random bits with approximation factor $1 - \left(\frac{2^k}{2^k+1}\right)^{2k}$, for any integer $k \geq 1$.

For $k = 1$, the competitive ratio evaluates to $5/9$. It approaches $1 - 1/e$ very quickly, for example, for $k = 10$ the absolute difference between the competitive ratio and $1 - 1/e$ is less than 0.0002. Our analysis is based on the analysis of the KVV algorithm by Birnbaum and Mathieu [5] and uses a result by Konrad et al. [25] concerning the performance of the Greedy algorithm on a randomly sampled subgraph which was originally developed in the context of streaming algorithms.

The results as described above are summarized in Table 1.

Models for Online Algorithms with Advice. The two main models for online computation with advice are the per-request model of Emek et at. [15] and the tape model of Böckenhauer et al. [7]. Both models were inspired by the original model proposed by Dobrev et al. [13]. In the model of Emek et at. [15], a bit string of a fixed length is received by the algorithm with

Table 1 Overview of our results, sorted with decreasing competitiveness.

Deterministic ratio	# of advice bits	Description and Authors
1	$\Theta(n \log n)$	(Miyazaki [29])
$1-\epsilon$	$\mathrm{O}(\frac{1}{\epsilon^5}n)$	Application of Eggert et al. [14] (here)
$1-\epsilon$	$\Omega(\log(\frac{1}{\epsilon})n)$	LB holds for any online algorithm (here)
$1-\frac{1}{e}+\epsilon$	$\Omega(n)$	LB holds for any online algorithm (Mikkelsen [28])
$1-\frac{1}{e}$	$\mathrm{O}(\log n)$	Exp. time Ranking-alg. (Böckenhauer et al. [9])
$\frac{3}{5}$	m	Category-algorithm using two categories (here)
$\frac{1}{2}+\epsilon$	$\Omega(\log\log\log n)$	LB holds for Ranking-algorithms (here)
Randomized ratio	**# of random bits**	**Description and Authors**
$1-\frac{1}{e}$	$m \log m$	KVV algorithm (Karp, Vazirani, Vazirani [24])
$1-\left(\frac{2^k}{2^k+1}\right)^{2k}$	km	Category-algorithm using 2^k categories (here)

each request for a total amount of advice that is at least linear in the size of the input. For this work, we use the tape model of Böckenhauer et al. [7], where the algorithm has access to an infinite advice string that it can access at any time (see Section 2 for a formal definition), allowing for advice that is sub-linear in the size of the input. Many online problems have been studied in the setting of online algorithms with advice (e.g. metrical task system [15], k-server problem [15, 9, 30, 20], paging [13, 7], bin packing problem [31, 11, 2], knapsack problem [8], reordering buffer management problem [1], list update problem [10], minimum spanning tree problem [4] and others). Interestingly, a variant of the algorithm with advice for list update problem of [10] was used to gain significant improvements in the compression rates for Burrows-Wheeler transform compression schemes [23]. The information-theoretic lower bound techniques for online algorithms with advice proposed by Emek et al. [15] applies to randomized algorithms and uses a reduction to a matching pennies game (essentially equivalent to the string guessing game). The reduction technique using the string guessing game of Böckenhauer et al. [6] is a refinement specifically for deterministic algorithms of the techniques of Emek et al.

Outline. Preliminaries are discussed in Section 2. Our $(1-\epsilon)$-competitive algorithm and a related advice lower bound are presented in Section 3. Then, in Section 4, we give the advice lower bound for $(\frac{1}{2}+\epsilon)$-competitive Ranking-algorithms. Last, in Section 5, we consider our randomized Category algorithm and our $\frac{3}{5}$-competitive advice Category algorithm.

2 Preliminaries

Unless stated otherwise, we consider a bipartite input graph $G=(A,B,E)$ with $A=[n]$ and $B=[m]$, for integers m,n such that $m=\Theta(n)$. The neighbourhood of a vertex v in graph G is denoted by $\Gamma_G(v)$. Let M be a matching in G. We denote the set of vertices matched in M by $V(M)$. For a vertex $v \in V(M)$, $M(v)$ denotes the vertex that is matched to v in M. Generally, we write M^* to denote a maximum matching, i.e., a matching of largest cardinality. For $A' \subseteq A, B' \subseteq B$, $opt(A',B')$ denotes the size of a maximum matching in $G[A' \cup B']$, the subgraph induced by $A' \cup B'$.

The Ranking Algorithm. Given permutations $\pi : [n] \to [n]$ and $\sigma : [m] \to [m]$, we write Ranking(G,π,σ) to denote the output matching of the Ranking algorithm when the A-vertices arrive in the order given by π, and the B-vertices are ranked according to σ. We may write Ranking(σ) to denote Ranking(G,π,σ) if π and G are clear from the context.

The Greedy Matching Algorithm. Greedy processes the edges of a graph in arbitrary order and inserts the current edge e into an initially empty matching M if $M \cup \{e\}$ is a matching. It computes a maximal matching which is of size at least $\frac{1}{2}|M^*|$.

Category Algorithms. For an integer k, let $c : [m] \to \{1, \dots, 2^k\}$ be an assignment of categories to the B-vertices. Then let $\sigma_c : [m] \to [m]$ be the unique permutation of the B-vertices such that for two vertices $b_1, b_2 \in B : \sigma_c(b_1) < \sigma_c(b_2)$ if and only if $c(b_1) < c(b_2)$ or $(c(b_1) = c(b_2)$ and $b_1 < b_2)$. The previous definition of σ_c is based on the natural ordering of the B-vertices. This gives a certain stability to the resulting permutation, since changing the category of a single vertex b does not affect the relative order of the vertices $B \setminus \{b\}$.

The Tape Advice Model. For a given request sequence I of length n for a maximization problem, an *online algorithm with advice* in the *tape advice model* computes the output sequence $\text{ALG}(I, \Phi) = \langle y_1, y_2, \dots, y_n \rangle$, where y_i is a function of the requests from 1 to i of I and the infinite binary advice string Φ. Algorithm ALG has an advice complexity of $b(n)$ if, for all n and any input sequence of length n, ALG reads no more than $b(n)$ bits from Φ.

3 Deterministic $(1 - \epsilon)$-competitive Advice Algorithms

3.1 Algorithm With $O(\frac{1}{\epsilon^5}n)$ Bits of Advice

The main idea of our online algorithm is the simulation of an augmenting-paths-based algorithm with the help of advice bits. We employ the deterministic algorithm of Eggert et al. [14] that has been designed for the data streaming model. It computes a $(1 - \epsilon)$-approximate matching, using $O(\frac{1}{\epsilon^5})$ passes over the edges of the input graph, where each pass i is used to compute a matching M_i in a subgraph $G_i = G[A_i \cup B_i]$, for some subsets $A_i \subseteq A$ and $B_i \subseteq B$, using the Greedy matching algorithm. In the first pass, M_1 is computed in G and thus constitutes a $\frac{1}{2}$-approximation. Let $M = M_1$. Then, $O(\frac{1}{\epsilon^2})$ phases follow, where in each phase, a set of disjoint augmenting paths is computed using $O(\frac{1}{\epsilon^3})$ applications of the Greedy matching algorithm (and thus $O(\frac{1}{\epsilon^3})$ passes per phase). At the end of a phase, M is augmented using the augmenting-paths found in this phase. Upon termination of the algorithm, M constitutes a $(1 - \epsilon)$-approximation (see [14] for the analysis).

The important property that allows us to translate this algorithm into an online algorithm with advice is the simple observation that the computed matching M is a subset of $\bigcup_i M_i$. For every i, we encode the vertices $A_i \subseteq A$ and $B_i \subseteq B$ that constitute the vertices of G_i using $n+m$ advice bits. Furthermore, for every vertex $a \in A$, we also encode the index $j(a)$ of the matching $M_{j(a)}$ that contains the edge that is incident to a in the final matching M (if a is not matched in M, then we set $j(a) = 0$). Last, using $O(\log n)$ bits, we encode the integers n and m, using a self-delimited encoding. Parameters n, m are required in order to determine the word size that allows the storage of the indices $j(a)$, and to determine the subgraphs G_i. The total number of advice bits is hence $O(\frac{1}{\epsilon^5}(n + m) + \log(\frac{1}{\epsilon^5})(m) + \log(n)) = O(\frac{1}{\epsilon^5}n)$.

After having read the advice bits, our online algorithm computes the $O(\frac{1}{\epsilon^5})$ Greedy matchings M_i simultaneously in the background while receiving the requests. Upon arrival of an $a \in A$, we match it to the $b \in B$ such that $ab \in M_{j(a)}$ incident to a if $j(a) \geq 1$, and we leave it unmatched if $j(a) = 0$. We thus obtain the following theorem:

▶ **Theorem 4.** *For every $\epsilon > 0$, there is a $(1 - \epsilon)$-competitive deterministic online algorithm for* MBM *that uses $O(\frac{1}{\epsilon^5}n)$ bits of advice.*

3.2 $\Omega(\log(\frac{1}{\epsilon})n)$ Advice Lower Bound

We complement the advice algorithm of the previous section with an $\Omega(\log(\frac{1}{\epsilon})n)$ advice lower bound for $(1-\epsilon)$-competitive deterministic advice algorithms. To show this, we make use of the lower bound techniques of [6] using the *string guessing game*, which is defined as follows.

▶ **Definition 5** (q-SGKH [6]). The *string guessing problem with known history* over an alphabet Σ of size $q \geq 2$ (q-SGKH) is an online minimization problem. The input consists of n and a request sequence $\sigma = r_1, \ldots, r_n$ of the characters, in order, of an n length string. An online algorithm A outputs a sequence $a_1, \ldots, a_n$ such that $a_i = f_i(n, r_1, \ldots, r_{i-1}) \in \Sigma$ for some computable function f_i. An important aspect of this problem is that the algorithm needs to produce its output character *before* the corresponding request: request r_i is revealed immediately after the algorithm outputs a_i. The cost of A is the Hamming distance between $a_1, \ldots, a_n$ and $r_1, \ldots, r_n$.

In [6], the following lower bound on the number of advice bits is shown for q-SGKH.

▶ **Theorem 6** ([6]). *Consider an input string of length n for q-SGKH. The minimum number of advice bits for any deterministic online algorithm that is correct for more than αn characters, for $\frac{1}{q} \leq \alpha < 1$, is $((1 - H_q(1-\alpha)) \log_2 q)n$, where $H_q(p) = p \log_q(q-1) - p \log_q p - (1-p) \log_q(1-p)$ is the q-ary entropy function.*

First, we define a sub-graph that is used in the construction of the lower bound sequence.

▶ **Definition 7.** A bipartite graph is *c-semi complete*, if it is isomorphic to $G = (A, B, E)$ with $A = \{a_1, \ldots, a_c\}, B = \{b_1, \ldots, b_c\}$, and $E = \{a_i, b_j : j \geq i\}$.

The following lemma presents the reduction from q-SGKH to MBM.

▶ **Lemma 8.** *For an integer $c \geq 3$, suppose that there is a deterministic ρ-competitive online algorithm for* MBM, *using bn bits of advice, where $1 - \frac{1}{c} + \frac{1}{c!} \leq \rho < 1$. Then, there exists a deterministic algorithm for $c!$-SGKH, using cbn bits of advice, that is correct for at least $(1-(1-\rho)c)n$ characters of the n-length string.*

Proof. Let ALG_{MAT} be a deterministic ρ-competitive online algorithm for MBM, using bn bits of advice, with $1 - \frac{1}{c} + \frac{1}{c!} \leq \rho$, for an integer $c \geq 3$. We will present an algorithm $\text{ALG}_{c!\text{-SGKH}}$ that, in an online manner, will generate a request sequence I_{MAT} based on its input, I (of length n), that can be processed by ALG_{MAT}. Further, the advice received by $\text{ALG}_{c!\text{-SGKH}}$ will be the advice that ALG_{MAT} requires for I_{MAT}. As shown below, the length of I_{MAT} is cn, hence $\text{ALG}_{c!\text{-SGKH}}$ requires cbn bits of advice. The solution produced by ALG_{MAT} on I_{MAT} will define the output produced by $\text{ALG}_{c!\text{-SGKH}}$.

Suppose first that the entire input sequence I is known in advance (we will argue later how to get around this assumption). Let Π be an enumeration of all the permutations of length c, and let $g : \Sigma \to \{1, \ldots, c!\}$ be a bijection between Σ, the alphabet of the $c!$-SGKH problem, and an index of a permutation in Π. The request sequence I_{MAT} has a length of cn, consisting of n distinct c-semi-complete graphs, where each graph is based on a request of I. That is, for each request r_i in I, we append c requests to I_{MAT} that correspond to the A-vertices of a c-semi-complete graph, where the indices of the B-vertices are permuted according to the permutation $\Pi[g(r_i)]$.

Since I is not known in advance, we must construct I_{MAT} in an online manner while predicting the requests r_j. For each request r_j, the procedure is as follows:

Let I_{MAT}^{j-1} be the $c(j-1)$-length prefix of I_{MAT}. Note that when predicting request r_j, requests $r_1, \ldots, r_{j-1}$ have already been revealed, and I_{MAT}^{j-1} can thus be constructed. The algorithm $\textsc{alg}_{c!\text{-}\textsc{sgkh}}$ simulates $\textsc{alg}_{\textsc{mat}}$ on $I^{j-1}_{\textsc{mat}}$ followed by another c-semi-complete graph $G_j = (A_j, B_j, E_j)$ such that, for $1 \le k \le c$, when vertex $a_k \in A_j$ is revealed, the B-vertices incident to a_k correspond exactly to the unmatched B-vertices of B_j in the current matching of $\textsc{alg}_{\textsc{mat}}$. By construction, $\textsc{alg}_{\textsc{mat}}$ computes a perfect matching in G_j. The computed perfect matching corresponds to a permutation π at some index z of Π, and algorithm $\textsc{alg}_{c!\text{-}\textsc{sgkh}}$ outputs $g^{-1}(z)$ as a prediction for r_j.

Consider a run of $\textsc{alg}_{\textsc{mat}}$ on $I_{\textsc{mat}}$. If $\textsc{alg}_{\textsc{mat}}$ computes a perfect matching on the jth semi-complete graph, then our algorithm predicted r_j correctly. Similarly, if this matching is not perfect, then our algorithm failed to predict r_j. Let ν be the total number of imperfect matchings, let $\textsc{alg}_{\textsc{mat}}(I_{\textsc{mat}})$ denote the matching computed by $\textsc{alg}_{\textsc{mat}}$ on $I_{\textsc{mat}}$, and let $\textsc{opt}(I_{\textsc{mat}})$ denote a perfect matching in the graph given by $I_{\textsc{mat}}$. Then:

$$|\textsc{alg}_{\textsc{mat}}(I_{\textsc{mat}})| \le |\textsc{opt}(I_{\textsc{mat}})| - \nu \iff \nu \le |\textsc{opt}(I_{\textsc{mat}})| - \rho \cdot |\textsc{opt}(I_{\textsc{mat}})| = (1-\rho)cn\,. \blacktriangleleft$$

We prove now the main lower bound result of this section.

▶ **Theorem 9.** *For an integer $c \ge 3$, any deterministic online algorithm with advice for* MBM *requires at least $\left(\frac{(1-H_q(1-\alpha))}{2}\log c\right) n$ bits of advice to be ρ-competitive for $1 - \frac{1}{c} + \frac{1}{c!} \le \rho < 1$, where H_q is the q-ary entropy function and $\alpha = 1-(1-\rho)c$.*

Proof. For $1 - \frac{1}{c} + \frac{1}{c!} \le \rho < 1$, let $\textsc{alg}_{\textsc{mat}}$ be a deterministic ρ-competitive online algorithm for MBM, using bn bits of advice. By Lemma 8, there exists an algorithm for $c!$-$\textsc{sgkh}$ that uses cbn bits of advice and is correct for at least αn characters of the n-length input string. The bounds on ρ and c imply $1/(c!) \le \alpha \le 1$. Thus, Theorem 6 implies $cbn \ge ((1-H_q(1-\alpha))\log(c!))n$ and, hence,

$$b \ge \frac{(1-H_q(1-\alpha))}{c}\log(c!) \ge \frac{(1-H_q(1-\alpha))}{2}\log c, \text{ as } c! \ge c^{c/2}. \qquad \blacktriangleleft$$

Setting $\varepsilon = 1/(2c) < 1/c - 1/(c!)$ for all $c \ge 3$, we get the following corollary. Note that, as ρ approaches 1 from below, α also approaches 1 from below and $H_q(1-\alpha)$ approaches 0.

▶ **Corollary 10.** *For any $0 < \varepsilon \le 1/6$, any $(1-\varepsilon)$-competitive deterministic online algorithm with advice for* MBM *requires $O(\log(\frac{1}{\epsilon})n)$ bits of advice.*

4 Advice Lower Bound for Ranking Algorithms

Let $\sigma_1, \ldots, \sigma_k : [n] \to [n]$ be rankings. We will show that there is a $2n$-vertex graph $G = (A, B, E)$ and an arrival order π such that $|\textsc{Ranking}(G, \pi, \sigma_i)| \le n(\frac{1}{2} + \epsilon) + o(n)$, for every σ_i and every constant $\epsilon > 0$, while G contains a perfect matching. Furthermore, the construction is such that $k \in \Omega(\log \log n)$.

The key property required for our lower bound is the fact that we can partition the set of B-vertices into disjoint subsets $B_1, \ldots, B_q$, each of large enough size, such that for every B_i with $B_i = \{b_1, \ldots, b_p\}$ and $b_1 < b_2 < \cdots < b_p$, the sequence $(\sigma_j(b_i))_i$ is monotonic, for every $1 \le j \le k$. In other words, the ranks of the nodes $b_1, \ldots, b_p$ appear in the rankings σ_i in either increasing or decreasing order. For each set B_i, we will construct a vertex-disjoint subgraph G_i on which $\textsc{Ranking}$ computes a matching that is close to a $\frac{1}{2}$-approximation. The subgraphs G_i are based on graph H_z that we define next.

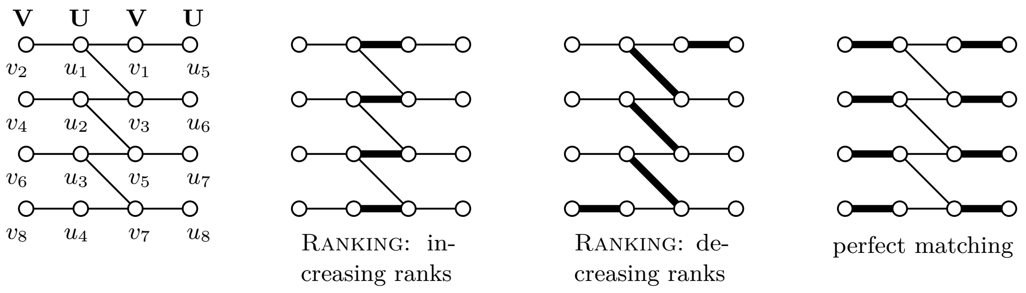


Figure 1 Left: U-vertices arrive in order $u_1, u_2, \ldots, u_8$. 'RANKING: increasing ranks' shows the resulting matching when $\sigma(v_1) < \sigma(v_2) < \cdots < \sigma(v_8)$. 'RANKING: decreasing ranks' shows the resulting matching when $\sigma(v_1) > \sigma(v_2) > \cdots > \sigma(v_8)$. Right: Perfect matching.

Construction of H_z. We construct now graph $H_z = (U, V, F)$ with $U = V = [z]$, for some even integer z, on which RANKING computes a matching that is close to a $\frac{1}{2}$-approximation, provided that the V vertices are ranked in either increasing or decreasing order.

Let $U = \{u_1, \ldots u_z\}$ be so that u_i arrives before u_{i+1} in π. Let $V = \{v_1, \ldots v_z\}$ be so that $v_i < v_{i+1}$ (which implies $v_i = i$). Then, for $1 \leq i \leq z/2$ we define $\Gamma_{H_z}(u_i) = \{v_{2i-1}, v_{2i}, v_{2i+1}\}$, and for $z/2 < i \leq z$ we define $\Gamma_{H_z}(u_i) = \{v_{2i-z-1}\}$. The graph H_8 is illustrated in Figure 1. It has the following properties:

1. If the sequence $(\sigma_i(b_j))_j$ is increasing, then $|\text{RANKING}(H_z, \pi, \sigma_i)| = z/2$.
2. If the sequence $(\sigma_i(b_j))_j$ is decreasing, then $|\text{RANKING}(H_z, \pi, \sigma_i)| = z/2 + 1$.
3. H_z has a perfect matching (of size z).

Lower Bound Proof. We prove first that we can appropriately partition the B-vertices that allow us to define the graphs G_i. Our prove relies on the well-known Erdős-Szekeres theorem [16] that we state in the form we need first.

▶ **Theorem 11** (Erdős-Szekeres [16]). *Every sequence of distinct integers of length n contains a monotonic (either increasing or decreasing) subsequence of length $\lceil \sqrt{n} \rceil$.*

▶ **Lemma 12.** *Let $\epsilon > 0$ be an arbitrary small constant. Then for any k permutations $\sigma_1, \ldots, \sigma_k : [n] \to [n]$ with $k \leq \log\log n - \log\log \frac{1}{\epsilon} - 2$, there is a partition of $B = [n]$ into subsets $C, B_1, B_2, \ldots$ such that:*

1. *$|B_i| \geq 1/\epsilon$ for every i,*
2. *$|C| \leq \sqrt{n}$,*
3. *For every $B_i = \{b_1, \ldots, b_p\}$ with $b_1 < b_2 < \cdots < b_p$, and every σ_j, the sequence $(\sigma_j(b_l))_l$ is monotonic.*

Proof. Let $S = B$. We iteratively remove subsets B_i from S until $|S| \leq \sqrt{n}$. The remaining elements then define set C. Thus, by construction, Item 2 is fulfilled.

Suppose that we have already defined sets $B_1, \ldots, B_i$. We show how to obtain set B_{i+1}. Let $S = B \setminus \bigcup_{j=1}^{i} B_j$ ($S = B$ if $i = 0$). Note that $|S| \geq \sqrt{n}$. By Theorem 11, there is a subset $B'_1 = \{b^1_1, \ldots, b^1_{\lceil n^{1/4} \rceil}\} \subseteq S$ with $b^1_1 < b^1_2 < \cdots < b^1_{\lceil n^{1/4} \rceil}$ such that the sequence $(\sigma_1(b_i))_{b_i \in B'_1}$ is monotonic. Then, again by Theorem 11, there is a subset $B'_2 = \{b^2_1, \ldots, b^2_{\lceil n^{1/8} \rceil}\} \subseteq B'_1$ with $b^2_1 < b^2_2 < \cdots < b^2_{\lceil n^{1/8} \rceil}$ such that the sequences $(\sigma_j(b_i))_{b_i \in B'_2}$ are monotonic, for every $j \in \{1, 2\}$. Similarly, we obtain that there is a subset $B'_w = \{b^w_1, \ldots, b^w_{\lceil n^{(1/2)^{w+1}} \rceil}\} \subseteq B'_{w-1}$ with $b^w_1 < b^w_2 < \cdots < b_{\lceil n^{(1/2)^{w+1}} \rceil}$ such that the sequences $(\sigma_j(b_i))_{b_i \in B'_w}$ are monotonic, for every $j \in \{1, \ldots, w\}$.

In order to guarantee Item 1, we solve the inequality $n^{(\frac{1}{2})^{w+1}} \geq \frac{1}{\epsilon}$ for w, and we obtain $w \leq \log\log n - \log\log\frac{1}{\epsilon} - 2$. This completes the proof. ◀

Equipped with the previous lemma, we are ready to prove our lower bound result.

▶ **Theorem 13.** *Let $\epsilon > 0$ be an arbitrary constant. For any k permutations $\sigma_1, \dots, \sigma_k : [n] \to [n]$ with $k \leq \log\log n - \log\log\frac{2}{\epsilon} - 2$ and arrival order $\pi : [n] \to [n]$, there is a graph $G = (A, B, E)$ such that for every σ_i:*

$$|\textsc{Ranking}(G, \pi, \sigma_i)| \leq (\frac{1}{2} + \epsilon)n + o(n),$$

while G contains a perfect matching.

Proof. Let $\epsilon' = \epsilon/2$. Let $G = (A, B, E)$ denote the hard instance graph. Let $C, B_1, B_2, \dots$ denote the partition of B according to Lemma 12 with respect to value ϵ'. Then, partition A into sets $A_0, A_1, \dots$ such that $|A_0| = |C|$ and for $i \geq 1$, $|A_i| = |B_i|$. Graph G is the disjoint union of subgraphs $G_0 = (A_0, C, E_0)$ and $G_i = (A_i, B_i, E_i)$, for $i \geq 1$. Subgraph G_0 is an arbitrary graph that contains a perfect matching. If $|B_i|$ is even, then G_i is an isomorphic copy of H_i. If $|B_i|$ is odd, then G_i is the disjoint union of an isomorphic copy of H_{i-1} and one edge. Then,

$$|\textsc{Ranking}(G, \pi, \sigma_i)| \leq \sum_{B_i}(|B_i|/2 + 2) + |C| \leq n/2 + 2\epsilon' n + \sqrt{n}.$$ ◀

5 Category Algorithms

5.1 Randomized Category Algorithm

In this section, we analyse the following randomized Ranking-algorithm:

Algorithm 1 Randomized Category Algorithm

```
Require: $G = (A, B, E)$, integer parameter $k \geq 1$
  For every $b \in B : c(b) \leftarrow$ random number in $\{1, 2, 3, \dots, 2^k\}$
  $\sigma_c \leftarrow$ permutation on $[m]$ such that $\sigma_c(b_1) < \sigma_c(b_2)$ iff $(c(b_1) < c(b_2))$ or $(c(b_1) = c(b_2)$
  and $b_1 < b_2)$, for every $b_1, b_2 \in B$
  return Ranking($\sigma_c$)
```

Considering Graphs with Perfect Matchings. First, similar to [5], we argue that the worst-case performance ratio of Algorithm 1 is obtained if the input graph contains a perfect matching. It requires the following observation:

▶ **Theorem 14** (Monotonicity [18, 24]). *Consider a fixed arrival order π and ranking σ for an input graph $G = (A, B, E)$. Let $H = G \setminus \{v\}$ for some vertex $v \in A \cup B$. Let π', σ' be the arrival order/ranking when restricted to vertices $A \cup B \setminus \{v\}$. Then,* Ranking(G, π, σ) *and* Ranking(H, π', σ') *are either identical or differ by a single alternating path starting at v.*

The previous theorem shows that the size of the matching produced by Algorithm 1 is monotonic with respect to vertex removals. Hence, if H is the graph obtained from G by removing all vertices that are not matched by a maximum matching in G, then the performance ratio of Ranking on H cannot be better than on G. We can thus assume that the input graph G has a perfect matching and $|A| = |B| = n$.

Analysis: General Idea. Let $B_i = \{b \in B : c(b) = i\}$, and denote the matching computed by the algorithm by M. The important quantities to consider for the analysis of Algorithm 1 are the probabilities:

$$x_i = \Pr_{b \in B} [b \in V(M) \mid b \in B_i],$$

i.e., the probability that a randomly chosen B-vertex of category i is matched by the algorithm. Determining lower bounds for the quantities x_i is enough in order to bound the expected matching size, since

$$\begin{aligned} \mathbb{E}|M| &= \sum_{b \in B} \Pr[b \in V(M)] = \sum_{b \in B} \sum_{i=1}^{2^k} \Pr[b \in B_i] \cdot \Pr[b \in V(M) \mid b \in B_i] \\ &= \frac{1}{2^k} \sum_{b \in B} \sum_{i=1}^{2^k} \Pr[b \in V(M) \mid b \in B_i] = \frac{n}{2^k} \sum_{i=1}^{2^k} x_i. \end{aligned} \tag{1}$$

We will first prove a bound on x_1 using a previous result of Konrad et al. [25]. Then, using similar ideas as Birnbaum and Mathieu [5], we will prove inequalities of the form $x_{i+1} \geq f(x_i, \ldots, x_1)$, for some function f which allow us to bound the probabilities $(x_i)_{i \geq 2}$.

Bounding x_1. Let $H = (U, V, F)$ be an arbitrary bipartite graph and let $U' \subseteq U$ be a uniform and random sample of U such that a node $u \in U$ is in U' with probability p. Konrad et al. showed in [25] that when running Greedy on the subgraph induced by vertices $U' \cup \Gamma_G(U')$, a relatively large fraction of the U'-vertices will be matched, for any order in which the edges of the input graph are processed that is independent of the choice of U'. More precisely, they prove the following theorem (Greedy(H', ω) denotes the output of Greedy on subgraph H' if edges of H' are considered in the order given by ω):

▶ **Theorem 15** ([25]). *Let $H = (U, V, F)$ be a bipartite graph, M^* a maximum matching, and let $U' \subseteq U$ be a uniform and independent random sample of U such that every vertex belongs to U' with probability p, $0 < p \leq 1$. Then for any edge arrival order ω,*

$$\mathbb{E}|\textsc{Greedy}(H[U' \cup \Gamma_H(U')], \omega)| \geq \frac{p}{1+p} |M^*|.$$

In Ranking, the vertices B_1 are always preferred over vertices $B \setminus B_1$. Thus, the matching $M_1 = \{ab \in M \mid b \in B_1\}$ is identical to the matching obtained when running Ranking on the subgraph induced by $A \cup B_1$. Since the previous theorem holds for any edge arrival order (that is independent from the choice of B'), we can apply the theorem (setting $B' = B_1, p = \frac{1}{2^k}$) and we obtain:

$$\mathbb{E}|B_1 \cap V(M)| \geq \frac{\frac{1}{2^k}}{1 + \frac{1}{2^k}} n = \frac{1}{2^k + 1} n.$$

Since $\mathbb{E}|B_1 \cap V(M)| = \sum_{b \in B} \Pr[b \in B_1] \cdot \Pr[b \in V(M) \mid b \in B_1] = \frac{n}{2^k} x_1$, we obtain $x_1 \geq 1 - \frac{1}{2^k+1}$.

Bounding $(x_i)_{i \geq 2}$. The key idea of the analysis of Birnbaum and Mathieu for the KVV-algorithm is the observation that, if a B-vertex of rank i is not matched by the algorithm, then its partner in an optimal matching is matched to a vertex of rank smaller than i.

Applied to our algorithm, if a B-vertex of category i is not matched, then its optimal partner $M^*(b)$ is matched to a B-vertex that belongs to a category $j \leq i$. Thus:

$$1 - x_i = \Pr_{b \in B}[b \notin V(M) \,|\, b \in B_i] = \\ \Pr_{b \in B}[b \notin V(M) \text{ and } M^*(b) \text{ matched in } M \text{ to a } b' \text{ with } c(b') \leq i \,|\, b \in B_i]\,. \quad (2)$$

The following lemma is similar to a clever argument by Birnbaum and Mathieu [5].

▶ **Lemma 16.**

$$\Pr_{b \in B}[b \notin V(M) \textit{ and } M^*(b) \textit{ matched in } M \textit{ to a } b' \textit{ with } c(b') \leq i \,|\, b \in B_i] \\ \leq \Pr_{b \in B}[M^*(b) \textit{ matched in } M \textit{ to a } b' \textit{ with } c(b') \leq i]\,. \quad (3)$$

Proof. Let c be uniformly distributed and let σ_c be the respective ranking. Pick now a random $\tilde{b} \in B$ and create new categories c' such that $c'(\tilde{b}) = i$ and for all $b \neq \tilde{b} : c'(b) = c(b)$. Let $\sigma_{c'}$ be the ranking given by c'.

Let $\tilde{a} = M^*(\tilde{b})$. Suppose that in a run of Ranking($\sigma_{c'}$), $\tilde{a}$ is matched to a vertex d' with $c'(d') \leq i$ and $\tilde{b}$ remains unmatched. Then, we will show that in the run of Ranking(σ_c), $\tilde{a}$ is matched to a vertex d with $c(d) \leq i$. This implies our result.

First, suppose that $\tilde{b}$ remains unmatched in Ranking(σ_c). Then, Ranking(σ_c) = Ranking($\sigma_{c'}$) and the claim is trivially true. Suppose now that $\tilde{b}$ is matched in Ranking(σ_c). Then, similar to the argument of [5], it can be seen that Ranking(σ_c) and Ranking($\sigma_{c'}$) differ only by one alternating path $b_0, a_1, b_1, a_2, b_2, \ldots$ starting at $b_0 = \tilde{b}$ such that for all i, (1) $a_{i+1}b_i \in$ Ranking(σ_c), (2) $a_i b_i \in$ Ranking($\sigma_{c'}$), and (3) $\sigma_c(b_i) > \sigma_c(b_{i+1})$. Property (3) implies $c(b_i) \leq c(b_{i+1})$. Thus if the category $\sigma_{c'}$ of the node that a_i is matched to in Ranking($\sigma_{c'}$) is k, then the category c of the node that a_i is matched to in Ranking(σ_c) is also at most k. ◀

The right side of Inequality 3 can be computed explicitly as follows:

$$\begin{aligned}\Pr_{b \in B}[M^*(b) \text{ matched in } M \text{ to a } b' \text{ with } c(b') \leq i] &= \Pr_{b \in B}[c(b) \leq i \text{ and } b \in V(M)] \\ &= \frac{1}{2^k} \sum_{j=1}^{i} x_j\,.\end{aligned}$$

This, together with Inequalities 2 and 3, yields $1 - x_i \leq \frac{1}{2^k} \sum_{j=1}^{i} x_j$. We obtain:

▶ **Theorem 17.** *Let $k \geq 1$ be an integer. Then Algorithm 1 is a randomized online algorithm for* MBM *with competitive ratio $1 - \left(\frac{2^k}{2^k+1}\right)^{2k}$ that uses $k \cdot m$ random bits.*

Proof. Following [5], the inequality $1 - x_i \leq \frac{1}{2^k} \sum_{j=1}^{i} x_j$ yields $S_i(1 + \frac{1}{2^k}) \geq 1 + S_{i-1}$, where $S_i = \sum_{j=1}^{i} x_i$ and $S_1 = x_1 \geq 1 - \frac{1}{2^k+1}$. According to Equality 1, we need to bound S_{2^k} from below. Quantity S_{2^k} is minimized if $S_i(1 + \frac{1}{2^k}) = 1 + S_{i-1}$, for all $i \geq 2$, which yields

$$S_i = \sum_{j=1}^{i} (1 - \frac{1}{2^k + 1})^j = 2^k \cdot \left(1 - \left(\frac{2^k}{2^k + 1}\right)^i\right)\,.$$

The result follows by plugging S_{2^k} into Equality 1. ◀

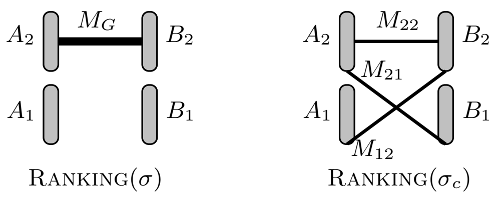


Figure 2 Quantities employed in the analysis of Algorithm 2.

5.2 Advice Category Algorithm

Let $\sigma : [m] \to [m]$ be the identity function, and let $M = \text{Ranking}(\sigma)$. It is well-known that M might be as poor as a $\frac{1}{2}$-approximation. Intuitively, B-vertices that are not matched in M are ranked too high in σ and have therefore no chance of being matched. We therefore assign category 1 to B-vertices that are not matched in M, and category 2 to all other nodes, see Algorithm 2. We will prove that this strategy gives a $\frac{3}{5}$-approximation algorithm.

Algorithm 2 Category-Advice Algorithm

```
**Computation of advice bits**
$\sigma \leftarrow$ permutation such that $\sigma(b) = b$, $M_G \leftarrow \text{Ranking}(\sigma)$, $M^* \leftarrow$ maximum matching
$\forall b \in B : c(b) \leftarrow \begin{cases} 1, & \text{if } b \notin V(M), \\ 2, & \text{otherwise.} \end{cases}$
**Online Algorithm with Advice** {Function $c$ is provided using $m$ advice bits}
$\sigma_c \leftarrow$ permutation on $[m]$ such that $\sigma_c(b_1) < \sigma_c(b_2)$ iff $(c(b_1) < c(b_2))$ or $(c(b_1) = c(b_2)$ and $b_1 < b_2)$, for every $b_1, b_2 \in B$
**return** $\text{Ranking}(\sigma_c)$
```

Our analysis requires a property of Ranking that has been previously used, e.g., in [5].

▶ **Lemma 18** (Upgrading unmatched vertices, Lemma 4 of [5]). *Let σ be a ranking and let $M = \text{Ranking}(\sigma)$. Let $b \in B$ be a vertex that is not matched in M. Let σ' be the ranking obtained from σ by changing the rank of b to any rank that is smaller than $\sigma(b)$ (and shifting the ranks of other vertices accordingly), and let $M' = \text{Ranking}(\sigma')$. Then, every vertex $a \in A$ matched in M to a vertex $b \in B$ is matched in M' to a vertex $b' \in B$ with $\sigma(b') \leq \sigma(b)$.*

▶ **Theorem 19.** *Alg. 2 is a $\frac{3}{5}$-competitive online algorithm for* MBM *using m advice bits.*

Proof. Let M denote the matching computed by the algorithm. Let $A_2 \subseteq A$, $B_2 \subseteq B$ be the subsets of vertices that are matched in M_G. Further, let $A_1 = A \setminus A_2$ and $B_1 = B \setminus B_2$ (the vertices not matched in M_G). See Figure 2 for an illustration of these quantities.

Then, for $i \in \{1, 2\}$, let $B_i^* = B_i \cap V(M^*)$. Let $M_{ij} = \{ab \in M \mid a \in A_i \text{ and } b \in B_j\}$. Then, $M = M_{21} \cup M_{12} \cup M_{22}$ since $M_{11} = \varnothing$ (the input graph does not contain any edges between A_1 and B_1 since otherwise some of them would also be contained in M_G). This setting is illustrated in Figure 2 in the appendix. We will bound now the sizes of M_{21}, M_{12} and M_{22} separately:

- *Bounding* $|M_{21}|$. Since B_1-vertices are preferred over B_2-vertices in $\text{Ranking}(\sigma_c)$ and since there are no edges between A_1 and B_1, M_{21} is a maximal matching between A_2 and B_1. Since $opt(A_2, B_1) = |B_1^*|$, we have $|M_{21}| \geq \frac{1}{2}|B_1^*|$.
- *Bounding* $|M_{22}|$. By Lemma 18, all A_2-vertices are matched in M. Thus, $|M_{22}| = |A_2| - |M_{21}|$.

- *Bounding* $|M_{12}|$. The algorithm finds a maximal matching between A_1 and $B_2 \setminus B(M_{22})$. Since $opt(A_1, B_2) \geq |A_1^*|$, we have $opt(A_1, B_2 \setminus B(M_{22})) \geq |A_1^*| - |M_{22}|$, and thus $|M_{12}| \geq \frac{1}{2}(|A_1^*| - |M_{22}|)$.

We combine the previous bounds and we obtain:

$$|M| = |M_{21}| + |M_{22}| + |M_{12}| \geq |A_2| + \frac{1}{2}(|A_1^*| - |A_2| + |M_{21}|) \geq \frac{1}{2}(|A_1^*| + |A_2| + \frac{1}{2}|B_1^*|).$$

Next, note that $|A_2| \geq |B_1^*|$ and $|A_1^*| + |B_1^*| = |M^*|$. We thus obtain $|M| \geq \frac{1}{2}|M^*| + \frac{1}{4}|B_1^*|$. Since $|B_1^*| \geq |M^*| - |M_G|$, we obtain $|M| \geq \frac{3}{4}|M^*| - \frac{1}{4}|M_G|$. Furthermore, Lemma 18 implies $|M| \geq |M_G|$, and hence $|M| \geq \max\{|M_G|, \frac{3}{4}|M^*| - \frac{1}{4}|M_G|\}$ which is at least $\frac{3}{5}|M^*|$. ◀

References

1 Anna Adamaszek, Marc P. Renault, Adi Rosén, and Rob van Stee. Reordering buffer management with advice. In *11th International Workshop on Approximation and Online Algorithms (WAOA)*, pages 132–143, September 2013.

2 Spyros Angelopoulos, Christoph Dürr, Shahin Kamali, Marc Renault, and Adi Rosén. Online bin packing with advice of small size. In Frank Dehne, Jörg-Rüdiger Sack, and Ulrike Stege, editors, *Proceedings of the 14th International Symposium on Algorithms and Data Structures (WADS)*, pages 40–53. Springer International Publishing, August 2015.

3 Bahman Bahmani and Michael Kapralov. Improved bounds for online stochastic matching. In *Proceedings of the 18th Annual European Conference on Algorithms (ESA)*, pages 170–181. Springer-Verlag, 2010.

4 Maria Paola Bianchi, Hans-Joachim Böckenhauer, Tatjana Brülisauer, Dennis Komm, and Beatrice Palano. Online minimum spanning tree with advice. In *Proceedings of the 42nd International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, pages 195–207, January 2016. doi:10.1007/978-3-662-49192-8_16.

5 Benjamin Birnbaum and Claire Mathieu. On-line bipartite matching made simple. *SIGACT News*, 39(1):80–87, March 2008. doi:10.1145/1360443.1360462.

6 Hans-Joachim Böckenhauer, Juraj Hromkovic, Dennis Komm, Sacha Krug, Jasmin Smula, and Andreas Sprock. The string guessing problem as a method to prove lower bounds on the advice complexity. *Theor. Comput. Sci.*, 554:95–108, 2014.

7 Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královič, Richard Královič, and Tobias Mömke. On the advice complexity of online problems. In Yingfei Dong, Ding-Zhu Du, and Oscar Ibarra, editors, *Proceedings of the 20th International Symposium on Algorithms and Computation (ISAAC)*, pages 331–340. Springer Berlin Heidelberg, December 2009.

8 Hans-Joachim Böckenhauer, Dennis Komm, Richard Královic, and Peter Rossmanith. The online knapsack problem: Advice and randomization. *Theor. Comput. Sci.*, 527:61–72, 2014.

9 Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královič, and Richard Královič. On the advice complexity of the k-server problem. In *Proceedings of the 38th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 6755 of *Lecture Notes in Computer Science*, pages 207–218. Springer Berlin Heidelberg, July 2011. doi:10.1007/978-3-642-22006-7_18.

10 Joan Boyar, Shahin Kamali, Kim S. Larsen, and Alejandro López-Ortiz. On the list update problem with advice. In *Proceedings of the 8th International Conference on Language and Automata Theory and Applications (LATA)*, pages 210–221, March 2014. doi:10.1007/978-3-319-04921-2_17.

11 Joan Boyar, Shahin Kamali, Kim S. Larsen, and Alejandro López-Ortiz. Online bin packing with advice. *Algorithmica*, 74(1):507–527, 2016. doi:10.1007/s00453-014-9955-8.

12 Nikhil R. Devanur, Kamal Jain, and Robert D. Kleinberg. Randomized primal-dual analysis of RANKING for online bipartite matching. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 101–107, January 2013. `doi:10.1137/1.9781611973105.7`.

13 Stefan Dobrev, Rastislav Královič, and Dana Pardubská. How much information about the future is needed? In *Proceedings of the 34th conference on Current trends in theory and practice of computer science (SOFSEM)*, pages 247–258, Berlin, Heidelberg, 2008. Springer-Verlag.

14 Sebastian Eggert, Lasse Kliemann, Peter Munstermann, and Anand Srivastav. Bipartite matching in the semi-streaming model. *Algorithmica*, 63(1):490–508, 2011.

15 Yuval Emek, Pierre Fraigniaud, Amos Korman, and Adi Rosén. Online computation with advice. *Theor. Comput. Sci.*, 412(24):2642–2656, 2011. `doi:10.1016/j.tcs.2010.08.007`.

16 Paul Erdös and George Szekeres. A combinatorial problem in geometry. *Compositio Mathematica*, 2:463–470, 1935.

17 Jon Feldman, Aranyak Mehta, Vahab Mirrokni, and S. Muthukrishnan. Online stochastic matching: Beating 1-1/e. In *Proceedings of the 2009 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 117–126, Washington, DC, USA, 2009. IEEE Computer Society. `doi:10.1109/FOCS.2009.72`.

18 Gagan Goel and Aranyak Mehta. Online budgeted matching in random input models with applications to adwords. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 982–991, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics. URL: `http://dl.acm.org/citation.cfm?id=1347082.1347189`.

19 Edward F. Grove. Online bin packing with lookahead. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 430–436, Philadelphia, PA, USA, 1995. Society for Industrial and Applied Mathematics. URL: `http://dl.acm.org/citation.cfm?id=313651.313781`.

20 Sushmita Gupta, Shahin Kamali, and Alejandro López-Ortiz. On advice complexity of the k-server problem under sparse metrics. In *Proceedings of the 20th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 55–67, July 2013. `doi:10.1007/978-3-319-03578-9_5`.

21 Magnús M. Halldórsson. Online coloring known graphs. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 917–918, Philadelphia, PA, USA, 1999. Society for Industrial and Applied Mathematics. URL: `http://dl.acm.org/citation.cfm?id=314500.315088`.

22 Magnús M. Halldórsson and Márió Szegedy. Lower bounds for on-line graph coloring. In *Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 211–216, Philadelphia, PA, USA, 1992. Society for Industrial and Applied Mathematics. URL: `http://dl.acm.org/citation.cfm?id=139404.139452`.

23 Shahin Kamali and Alejandro López-Ortiz. Better compression through better list update algorithms. In *Proceedings of the Data Compression Conference (DCC)*, pages 372–381, March 2014. `doi:10.1109/DCC.2014.86`.

24 R. M. Karp, U. V. Vazirani, and V. V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the Twenty-second Annual ACM Symposium on Theory of Computing (STOC)*, pages 352–358, New York, NY, USA, 1990. ACM. `doi:10.1145/100216.100262`.

25 Christian Konrad, Frédéric Magniez, and Claire Mathieu. Maximum matching in semi-streaming with few passes. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, volume 7408 of *Lecture Notes in Computer Science*, pages 231–242. Springer Berlin Heidelberg, 2012. `doi:10.1007/978-3-642-32512-0_20`.

26 Mohammad Mahdian and Qiqi Yan. Online bipartite matching with random arrivals: An approach based on strongly factor-revealing lps. In *Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing (STOC)*, pages 597–606, New York, NY, USA, 2011. ACM. doi:10.1145/1993636.1993716.

27 Vahideh H. Manshadi, Shayan Oveis Gharan, and Amin Saberi. Online stochastic matching: Online actions based on offline statistics. In *Proceedings of the Twenty-second Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1285–1294. SIAM, 2011. URL: http://dl.acm.org/citation.cfm?id=2133036.2133134.

28 Jesper W. Mikkelsen. Randomization can be as helpful as a glimpse of the future in online computation. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP)*, July 2016. URL: http://arxiv.org/abs/1511.05886.

29 Shuichi Miyazaki. On the advice complexity of online bipartite matching and online stable marriage. *Inf. Process. Lett.*, 114(12):714–717, December 2014. doi:10.1016/j.ipl.2014.06.013.

30 Marc P. Renault and Adi Rosén. On online algorithms with advice for the k-server problem. *Theory Comput. Syst.*, 56(1):3–21, 2015. doi:10.1007/s00224-012-9434-z.

31 Marc P. Renault, Adi Rosén, and Rob van Stee. Online algorithms with advice for bin packing and scheduling problems. *Theor. Comput. Sci.*, 600:155–170, 2015. doi:10.1016/j.tcs.2015.07.050.

32 Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, February 1985. doi:10.1145/2786.2793.

BlockQuicksort: Avoiding Branch Mispredictions in Quicksort

Stefan Edelkamp[1] **and Armin Weiß**[2]

1 **TZI, Universität Bremen, Bremen, Germany**
2 **Stevens Institute of Technology, Hoboken, NJ, USA**

Abstract

Since the work of Kaligosi and Sanders (2006), it is well-known that Quicksort – which is commonly considered as one of the fastest in-place sorting algorithms – suffers in an essential way from branch mispredictions. We present a novel approach to address this problem by partially decoupling control from data flow: in order to perform the partitioning, we split the input in blocks of constant size (we propose 128 data elements); then, all elements in one block are compared with the pivot and the outcomes of the comparisons are stored in a buffer. In a second pass, the respective elements are rearranged. By doing so, we avoid conditional branches based on outcomes of comparisons at all (except for the final Insertionsort). Moreover, we prove that for a static branch predictor the average total number of branch mispredictions is at most $\epsilon n \log n + \mathcal{O}(n)$ for some small ϵ depending on the block size when sorting n elements.

Our experimental results are promising: when sorting random integer data, we achieve an increase in speed (number of elements sorted per second) of more than 80% over the GCC implementation of C++ `std::sort`. Also for many other types of data and non-random inputs, there is still a significant speedup over `std::sort`. Only in few special cases like sorted or almost sorted inputs, `std::sort` can beat our implementation. Moreover, even on random input permutations, our implementation is even slightly faster than an implementation of the highly tuned Super Scalar Sample Sort, which uses a linear amount of additional space.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases in-place sorting, Quicksort, branch mispredictions, lean programs

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.38

1 Introduction

Sorting a sequence of elements of some totally ordered universe remains one of the most fascinating and well-studied topics in computer science. Moreover, it is an essential part of many practical applications. Thus, efficient sorting algorithms directly transfer to a performance gain for many applications. One of the most widely used sorting algorithms is Quicksort, which has been introduced by Hoare in 1962 [14] and is considered to be one of the most efficient sorting algorithms. For sorting an array, it works as follows: first, it chooses an arbitrary pivot element and then rearranges the array such that all elements smaller than the pivot are moved to the left side and all elements larger than the pivot are moved to the right side of the array – this is called *partitioning*. Then, the left and right side are both sorted recursively. Although its average[1] number of comparisons is not optimal – $1.38n \log n + \mathcal{O}(n)$ vs. $n \log n + \mathcal{O}(n)$ for Mergesort –, its over-all instruction count is very low. Moreover, by

[1] Here and in the following, the average case refers to a uniform distribution of all input permutations assuming all elements are different.

choosing the pivot element as median of some larger sample, the leading term $1.38n \log n$ for the average number of comparisons can be made smaller – even down to $n \log n$ when choosing the pivot as median of some sample of growing size [22]. Other advantages of Quicksort are that it is easy to implement and that it does not need extra memory except the recursion stack of logarithmic size (even in the worst case if properly implemented). A major drawback of Quicksort is its quadratic worst-case running time. Nevertheless, there are efficient ways to circumvent a really bad worst-case. The most prominent is Introsort (introduced by Musser [23]) which is applied in GCC implementation of `std::sort`: as soon as the recursion depth exceeds a certain limit, the algorithm switches to Heapsort.

Another deficiency of Quicksort is that it suffers from branch mispredictions (or branch misses) in an essential way. On modern processors with long pipelines (14 stages for Intel Haswell, Broadwell, Skylake processors – for the older Pentium 4 processors even more than twice as many) every branch misprediction causes a rather long interruption of the execution since the pipeline has to be filled anew. In [16], Kaligosi and Sanders analyzed the number of branch mispredictions incurred by Quicksort. They examined different simple branch prediction schemes (static prediction and 1-bit, 2-bit predictors) and showed that with all of them, Quicksort with a random element as pivot causes on average $cn \log n + \mathcal{O}(n)$ branch mispredictions for some constant $c = 0.34$ (resp. $c = 0.46$, $c = 0.43$). In particular, in Quicksort with random pivot element, every fourth comparison is followed by a mispredicted branch. The reason is that for partitioning, each element is compared with the pivot and depending on the outcome either it is swapped with some other element or not. Since for an optimal pivot (the median), the probability of being smaller the pivot is 50%, there is no way to predict these branches.

Kaligosi and Sanders also established that choosing skewed pivot elements (far off the median) might even decrease the running time because it makes branches more predictable. This also explains why, although theoretically larger samples for pivot selection were shown to be superior, in practice the median-of three variant turned out to be the best. In [5], the skewed pivot phenomenon is confirmed experimentally. Moreover, in [21], precise theoretical bounds on the number of branch misses for Quicksort are given – establishing also theoretical superiority of skewed pivots under the assumption that branch mispredictions are expensive.

In [7] Brodal and Moruz proved a general lower bound on the number of branch mispredictions given that every comparison is followed by a conditional branch which depends on the outcome of the comparison. In this case there are $\Omega(n \log_d n)$ branch mispredictions for a sorting algorithm which performs $\mathcal{O}(dn \log n)$ comparisons. As Elmasry and Katajainen remarked in [10], this theorem does not hold anymore if the results of comparisons are not used for conditional branches. Indeed, they showed that every program can be transformed into a program which induces only a constant number of branch misses and whose running time is linear in the running time of the original program. However, this general transformation introduces a huge constant factor overhead. Still, in [10] and [11] Elmasry, Katajainen and Stenmark showed how to efficiently implement many algorithms related to sorting with only few branch mispredictions. They call such programs *lean*. In particular, they present variants of Mergesort and Quicksort suffering only very little from branch misses. Their Quicksort variant (called Tuned Quicksort, for details on the implementation, see [17]) is very fast for random permutations – however, it does not behave well with duplicate elements because it applies Lomuto's uni-directional partitioner (see e. g. [8]).

Another development in recent years is multi-pivot Quicksort (i. e. several pivots in each partitioning stage [3, 4, 19, 28, 29]). It started with the introduction of Yaroslavskiy's dual-pivot Quicksort [31] – which, surprisingly, was faster than known Quicksort variants and, thus, became the standard sorting implementation in Oracle Java 7 and Java 8. Concerning branch mispredictions all these multi-pivot variants behave essentially like ordinary Quicksort [21]; however, they have one advantage: every data element is accessed only a few times (this is also referred to as the number of *scans*). As outlined in [4], increasing the number of pivot elements further (up to 127 or 255), leads to Super Scalar Sample Sort, which has been introduced by Sanders and Winkel [25]. Super Scalar Sample Sort not only has the advantage of few scans, but also is based on the idea of avoiding conditional branches. Indeed, the correct bucket (the position between two pivot elements) can be found by converting the results of comparisons to integers and then simply performing integer arithmetic. In their experiments Sanders and Winkel show that Super Scalar Sample Sort is approximately twice as fast as Quicksort (`std::sort`) when sorting random integer data. However, Super Scalar Sample Sort has one major draw-back: it uses a linear amount of extra space (for sorting n data elements, it requires space for another n data elements and additionally for more than n integers). In the conclusion of [16], Kaligosi and Sander raised the question:

> *However, an in-place sorting algorithm that is better than Quicksort with skewed pivots is an open problem.*

(Here, in-place means that it needs only a constant or logarithmic amount of extra space.) In this work, we solve the problem by presenting our block partition algorithm, which allows to implement Quicksort without any branch mispredictions incurred by conditional branches based on results of comparisons (except for the final Insertionsort – also there are still conditional branches based on the control-flow, but their amount is relatively small). We call the resulting algorithm BlockQuicksort. Our work is inspired by Tuned Quicksort from [11], from where we also borrow parts of our implementation. The difference is that by doing the partitioning block-wise, we can use Hoare's partitioner, which is far better with duplicate elements than Lomuto's partitioner (although Tuned Quicksort can be made working with duplicates by applying a check for duplicates similar to what we propose for BlockQuicksort as one of the further improvements in Section 3.2). Moreover, BlockQuicksort is also superior to Tuned Quicksort for random permutations of integers.

Our Contributions

- We present a variant of the partition procedure that only incurs few branch mispredictions by storing results of comparisons in constant size buffers.
- We prove an upper bound of $\epsilon n \log n + \mathcal{O}(n)$ branch mispredictions on average, where $\epsilon < \frac{1}{16}$ for our proposed block size (Theorem 1).
- We propose some improvements over the basic version.
- We implemented our algorithm with an `stl`-style interface[2].
- We conduct experiments and compare BlockQuicksort with `std::sort`, Yaroslavskiy's dual-pivot Quicksort and Super Scalar Sample Sort – on random integer data it is faster than all of these and also Katajainen et al.'s Tuned Quicksort.

Outline. Section 2 introduces some general facts on branch predictors and mispredictions, and gives a short account of standard improvements of Quicksort. In Section 3, we give a precise description of our block partition method and establish our main theoretical result – the bound on the number of branch mispredictions. Finally, in Section 4, we

[2] Code available at `https://github.com/weissan/BlockQuicksort`

experimentally evaluate different block sizes, different pivot selection strategies and compare our implementation with other state of the art implementations of Quicksort and Super Scalar Sample Sort.

Further experimental results as well as the C++ Code of the basic version of BlockQuicksort can be found in [9].

2 Preliminaries

Logarithms denoted by log are always base 2. The term *average case* refers to a uniform distribution of all input permutations assuming all elements are different. In the following `std::sort` always refers to its GCC implementation.

Branch Misses. Branch mispredictions can occur when the code contains conditional jumps (i. e. *if* statements, loops, etc.). Whenever the execution flow reaches such a statement, the processor has to decide in advance which branch to follow and decode the subsequent instructions of that branch. Because of the length of the pipeline of modern microprocessors, a wrong predicted branch causes a large delay since, before continuing the execution, the instructions for the other branch have to be decoded.

Branch Prediction Schemes. Precise branch prediction schemes of most modern processors are not disclosed to the public. However, the simplest schemes suffice to make BlockQuicksort induce only few mispredictions.

The easiest branch prediction scheme is the *static predictor*: for every conditional jump the compiler marks the more likely branch. In particular, that means that for every *if* statement, we can assume that there is a misprediction if and only if the *if* branch is not taken; for every *loop* statement, there is precisely one misprediction for every time the execution flow reaches that loop: when the execution leaves the loop. For more information about branch prediction schemes, we refer to [13, Section 3.3].

How to avoid Conditional Branches. The usual implementation of sorting algorithms performs conditional jumps based on the outcome of comparisons of data elements. There are at least two methods how these conditional jumps can be avoided – both are supported by the hardware of modern processors:

- Conditional moves (`CMOVcc` instructions on x86 processors) – or, more general, conditional execution. In C++ compilation to a conditional move can be (often) triggered by

```
i = (x < y) ? j : i;
```

- Cast Boolean variables to integer (`SETcc` instructions x86 processors). In C++:

```
int i = (x < y);
```

Also many other instruction sets support these methods (e. g. ARM [2], MIPS [24]). Still, the Intel Architecture Optimization Reference Manual [15] advises only to use these instructions to avoid unpredictable branches (as it is the case for sorting) since correctly predicted branches are still faster. For more examples how to apply these methods to sorting, see [11].

Quicksort and improvements. The central part of Quicksort is the partitioning procedure. Given some pivot element, it returns a pointer p to an element in the array and rearranges the array such that all elements left of the p are smaller or equal the pivot and all elements on the right are greater or equal the pivot. Quicksort first chooses some pivot element, then performs the partitioning, and, finally, recurses on the elements smaller and larger the pivot – see Algorithm 1. We call the procedure which organizes the calls to the partitioner the *Quicksort main loop*.

Algorithm 1 Quicksort

```
1: procedure QUICKSORT($A[\ell, \dots, r]$)
2:     if $r > \ell$ then
3:         pivot ← choosePivot($A[\ell, \dots, r]$)
4:         cut ← partition($A[\ell, \dots, r]$, pivot)
5:         Quicksort($A[\ell, \dots, \text{cut} - 1]$)
6:         Quicksort($A[\text{cut}, \dots, r]$)
7:     end if
8: end procedure
```

There are many standard improvements for Quicksort. For our optimized Quicksort main loop (which is a modified version of Tuned Quicksort [11, 17]), we implemented the following:

- A very basic optimization due to Sedgewick [27] avoids recursion partially (e. g. `std::sort`) or totally (here – this requires the introduction of an explicit stack).
- Introsort [23]: there is an additional counter for the number of recursion levels. As soon as it exceeds some bound (`std::sort` uses $2 \log n$ – we use $2 \log n + 3$), the algorithms stops Quicksort and switches to Heapsort [12, 30] (only for the respective sub-array). By doing so, a worst-case running time of $\mathcal{O}(n \log n)$ is guaranteed.
- Sedgewick [27] also proposed to switch to Insertionsort (see e. g. [18, Section 5.2.1]) as soon as the array size is less than some fixed small constant (16 for `std::sort` and our implementation). There are two possibilities when to apply Insertionsort: either during the recursion, when the array size becomes too small, or at the very end after Quicksort has finished. We implemented the first possibility (in contrast to `std::sort`) because for sorting integers, it hardly made a difference, but for larger data elements there was a slight speedup (in [20] this was proposed as *memory-tuned Quicksort*).
- After partitioning, the pivot is moved to its correct position and not included in the recursive calls (not applied in `std::sort`).
- The basic version of Quicksort uses a random or fixed element as pivot. A slight improvement is to choose the pivot as median of three elements – typically the first, in the middle and the last. This is applied in `std::sort` and many other Quicksort implementations. Sedgewick [27] already remarked that choosing the pivots from an even larger sample does not provide a significant increase of speed. In view of the experiments with skewed pivots [16], this is no surprise. For BlockQuicksort, a pivot closer to the median turns out to be beneficial (Figure 2 in Section 4). Thus, it makes sense to invest more time to find a better pivot element. In [22], Martinez and Roura show that the number of comparisons incurred by Quicksort is minimal if the pivot element is selected as median of $\Theta(\sqrt{n})$ elements. Another variant is to choose the pivot as median of three (resp. five) elements which themselves are medians of of three (resp. five) elements. We implemented all these variants for our experiments – see Section 4.

Our main contribution is the block partitioner, which we describe in the next section.

3 Block Partitioning

The idea of block partitioning is quite simple. Recall how Hoare's original partition procedure works (Algorithm 2):

Algorithm 2 Hoare's Partitioning

```
1: procedure Partition($A[\ell, \dots, r]$, pivot)
2:     while $\ell < r$ do
3:         while $A[\ell] <$ pivot do $\ell$++ end while
4:         while $A[r] >$ pivot do $r$−− end while
5:         if $\ell < r$ then swap($A[\ell], A[r]$); $\ell$++; $r$−− end if
6:     end while
7:     return $\ell$
8: end procedure
```

Two pointers start at the leftmost and rightmost elements of the array and move towards the middle. In every step the current element is compared to the pivot (Line 3 and 4). If some element on the right side is less or equal the pivot (resp. some element on the left side is greater or equal), the respective pointer stops and the two elements found this way are swapped (Line 5). Then the pointers continue moving towards the middle.

The idea of BlockQuicksort (Algorithm 3) is to separate Lines 3 and 4 of Algorithm 2 from Line 5: fix some block size B; we introduce two buffers $\mathrm{offsets}_L[0, \dots, B-1]$ and $\mathrm{offsets}_R[0, \dots, B-1]$ for storing pointers to elements ($\mathrm{offsets}_L$ will store pointers to elements on the left side of the array which are greater or equal than the pivot element – likewise $\mathrm{offsets}_R$ for the right side). The main loop of Algorithm 3 consists of two stages: the scanning phase (Lines 5 to 18) and the rearrangement phase (Lines 19 to 26).

Like for classical Hoare partition, we also start with two pointers (or indices as in the pseudocode) to the leftmost and rightmost element of the array. First, the scanning phase takes place: the buffers which are empty are refilled. In order to do so, we move the respective pointer towards the middle and compare each element with the pivot. However, instead of stopping at the first element which should be swapped, only a pointer to the element is stored in the respective buffer (Lines 8 and 9 resp. 15 and 16 – actually the pointer is always stored, but depending on the outcome of the comparison a counter holding the number of pointers in the buffer is increased or not) and the pointer continues moving towards the middle. After an entire block of B elements has been scanned (either on both sides of the array or only on one side), the rearranging phase begins: it starts with the first positions of the two buffers and swaps the data elements they point to (Line 21); then it continues until one of the buffers contains no more pointers to elements which should be swapped. Now the scanning phase is restarted and the buffer that has run empty is filled again.

The algorithm continues this way until fewer elements than two times the block size remain. Now, the simplest variant is to switch to the usual Hoare partition method for the remaining elements (in the experiments with suffix `Hoare finish`). But, we also can continue with the idea of block partitioning: the algorithm scans the remaining elements as one or two final blocks (of smaller size) and performs a last rearrangement phase. After that, some elements to swap in one of the two buffers might still remain, while the other buffer is empty. With one run through the buffer, all these elements can be moved to the left resp. right (similar as it is done in the Lomuto partitioning method, but without performing actual comparisons). We do not present the details for this final rearranging here because on one hand it gets a little tedious and on the other hand it does neither provide a lot of insight into the algorithm nor is it necessary to prove our result on branch mispredictions.

Algorithm 3 Block partitioning

```
1: procedure BlockPartition($A[\ell, \dots, r]$, pivot)
2:     integer $\text{offsets}_L[0, \dots, B-1], \text{offsets}_R[0, \dots, B-1]$
3:     integer $\text{start}_L, \text{start}_R, \text{num}_L, \text{num}_R \leftarrow 0$
4:     while $r - \ell + 1 > 2B$ do                                    ▷ start main loop
5:         if $\text{num}_L = 0$ then                          ▷ if left buffer is empty, refill it
6:             $\text{start}_L \leftarrow 0$
7:             for $i = 0, \dots, B-1$ do
8:                 $\text{offsets}_L[\text{num}_L] \leftarrow i$
9:                 $\text{num}_L \mathrel{+}= (\text{pivot} \geq A[\ell + i])$        ▷ scanning phase for left side
10:            end for
11:        end if
12:        if $\text{num}_R = 0$ then                         ▷ if right buffer is empty, refill it
13:            $\text{start}_R \leftarrow 0$
14:            for $i = 0, \dots, B-1$ do
15:                $\text{offsets}_R[\text{num}_R] \leftarrow i$
16:                $\text{num}_R \mathrel{+}= (\text{pivot} \leq A[r - i])$       ▷ scanning phase for right side
17:            end for
18:        end if
19:        integer $\text{num} = \min(\text{num}_L, \text{num}_R)$
20:        for $j = 0, \dots, \text{num} - 1$ do
21:            swap($A\big[\ell + \text{offsets}_L[\text{start}_L + j]\big], A\big[r - \text{offsets}_R[\text{start}_R + j]\big]$) ▷ rearrangement phase
22:        end for
23:        $\text{num}_L, \text{num}_R \mathrel{-}= \text{num}$; $\text{start}_L, \text{start}_R \mathrel{+}= \text{num}$
24:        if $(\text{num}_L = 0)$ then $\ell \mathrel{+}= B$ end if
25:        if $(\text{num}_R = 0)$ then $r \mathrel{-}= B$ end if
26:    end while                                                    ▷ end main loop
27:    compare and rearrange remaining elements
28: end procedure
```

3.1 Analysis

If the input consists of random permutations (all data elements different), the average numbers of comparisons and swaps are the same as for usual Quicksort with median-of-three. This is because both Hoare's partitioner and the block partitioner preserve randomness of the array.

The number of scanned elements (total number of elements loaded to the registers) is increased by two times the number of swaps, because for every swap, the data elements have to be loaded again. However, the idea is that due to the small block size, the data elements still remain in L1 cache when being swapped – so the additional scan has no negative effect on the running time. In Section 4 we see that for larger data types and from a certain threshold on, an increasing size of the blocks has a negative effect on the running time. Therefore, the block size should not be chosen too large – we propose $B = 128$ and fix this constant throughout (thus, already for inputs of moderate size, the buffers also do not require much more space than the stack for Quicksort).

Branch mispredictions. The next theorem is our main theoretical result. For simplicity we assume here that BlockQuicksort is implemented without the worst-case-stopper Heapsort (i. e. there is no limit on the recursion depth). Since there is only a low probability that a high recursion depth is reached while the array is still large, this assumption is not a real restriction. We analyze a static branch predictor: there is a misprediction every time a loop is left and a misprediction every time the *if* branch of an *if* statement is not taken.

▶ **Theorem 1.** *Let $\mathcal{C}$ be the average number of comparisons of Quicksort with constant size pivot sample. Then BlockQuicksort (without limit to the recursion depth and with the same pivot selection method) with blocksize B induces at most $\frac{6}{B} \cdot \mathcal{C} + \mathcal{O}(n)$ branch mispredictions on average. In particular, BlockQuicksort with median-of-three induces less then $\frac{8}{B} n \log n + \mathcal{O}(n)$ branch mispredictions on average.*

Theorem 1 shows that when choosing the block size sufficiently large, the $n \log n$-term becomes very small and – for real-world inputs – we can basically assume a linear number of branch mispredictions. Moreover, Theorem 1 can be generalized to samples of non-constant size for pivot selection. Since the proof might become tedious, we stick to the basic variant here. The constant 6 in Theorem 1 can be replaced by 4 when implementing Lines 19, 24, and 25 of Algorithm 3 with conditional moves.

▶ Remark. The $\mathcal{O}(n)$-term in Theorem 1 can be bounded by $3n$ by taking a closer look to the final rearranging phase. For a heuristic argument see [9].

Proof. First, we show that every execution of the block partitioner Algorithm 3 on an array of length n induces at most $\frac{6}{B} n + c$ branch mispredictions for some constant c. In order to do so, we only need to look at the main loop (Line 4 to 27) of Algorithm 3 because the final scanning and rearrangement phases consider only a constant (at most $2B$) number of elements. Inside the main loop there are three *for* loops (starting Lines 7, 14, 20), four *if* statements (starting Lines 5, 12, 24, 25) and the min calculation (whose straightforward implementation is an *if* statement – Line 19). We know that in every execution of the main loop at least one of the conditions of the *if* statements in Line 5 and 12 is true because in every rearrangement phase at least one buffer runs empty. The same holds for the two *if* statements in Line 24 and 25. Therefore, we obtain at most two branch mispredictions for the *if*s, three for the *for* loops and one for the min in every execution of the main loop.

In every execution of the main loop, there are at least B comparisons of elements with the pivot. Thus, the number of branch misses in the main loop is at most $\frac{6}{B}$ times the number of comparisons. Hence, for every input permutation the total number of branch mispredictions of BlockQuicksort is at most $\frac{6}{B} \cdot \#\text{comparisons} + (c + c') \cdot \#\text{calls to partition} + \mathcal{O}(n)$, where c' it the number of branch mispredictions of one execution of the main loop of Quicksort (including pivot selection, which only needs a constant number of instructions) and the $\mathcal{O}(n)$ term comes from the final Insertionsort. The number of calls to partition is bounded by n because each element can be chosen as pivot only once (since the pivots are not contained in the arrays for the recursive calls). Thus, by taking the average over all input permutations, the first statement follows.

The second statement follows because Quicksort with median-of-three incurs $1.18 n \log n + \mathcal{O}(n)$ comparisons on average [26]. ◀

3.2 Further Tuning of Block Partitioning

We propose and implemented further tunings for our block partitioner:

1. Loop unrolling: since the block size is a power of two, the loops of the scanning phase can be unrolled four or even eight times without causing additional overhead.
2. Cyclic permutations instead of swaps: We replace

```
1: for $j = 0, \dots, \text{num} - 1$ do
2:     swap($A\big[\ell + \text{offsets}_L[\text{start}_L + j]\big], A\big[r - \text{offsets}_R[\text{start}_R + j]\big]$)
3: end for
```

by the following code, which does not perform exactly the same data movements, but still in the end all elements less than the pivot are on the left and all elements greater are on the right:

```
1: temp ← $A\big[\ell + \text{offsets}_L[\text{start}_L]\big]$
2: $A\big[\ell + \text{offsets}_L[\text{start}_L]\big] \leftarrow A\big[r - \text{offsets}_R[\text{start}_R]\big]$
3: for $j = 1, \ldots, \text{num} - 1$ do
4:     $A\big[r - \text{offsets}_R[\text{start}_R + j - 1]\big] \leftarrow A\big[\ell + \text{offsets}_L[\text{start}_L + j]\big]$
5:     $A\big[\ell + \text{offsets}_L[\text{start}_L + j]\big] \leftarrow A\big[r - \text{offsets}_R[\text{start}_R + j]\big]$
6: end for
7: $A\big[r - \text{offsets}_R[\text{start}_R + \text{num} - 1]\big] \leftarrow$ temp
```

Note that this is also a standard improvement for partitioning – see e. g. [1].

In the following, we always assume these two improvements since they are of very basic nature (plus one more small change in the final rearrangement phase). We call the variant without them `block_partition_simple`.

The next improvement is a slight change of the algorithm: in our experiments we noticed that for small arrays with many duplicates the recursion depth becomes often higher than the threshold for switching to Heapsort – a way to circumvent this is an additional check for duplicates equal to the pivot if one of the following two conditions applies:

- the pivot occurs twice in the sample for pivot selection (in the case of median-of-three),
- the partitioning results very unbalanced for an array of small size.

The check for duplicates takes place after the partitioning is completed. Only the larger half of the array is searched for elements equal to the pivot. This check works similar to Lomuto's partitioner (indeed, we used the implementation from [17]): starting from the position of the pivot, the respective half of the array is scanned for elements equal to the pivot (this can be done by one *less than* comparison since elements are already known to be greater or equal (resp. less or equal) the pivot)). Elements which are equal to the pivot are moved to the side of the pivot. The scan continues as long as at least every fourth element is equal to the pivot (instead every fourth one could take any other ratio – this guarantees that the check stops soon if there are only few duplicates).

After this check, all elements which are identified as being equal to the pivot remain in the middle of the array (between the elements larger and the elements smaller than the pivot); thus, they can be excluded from further recursive calls. We denote this version with the suffix `duplicate check` (dc).

4 Experiments

We ran thorough experiments with implementations in C++ on different machines with different types of data and different kinds of input sequences. The experiments are run on an Intel Core i5-2500K CPU (3.30GHz, 4 cores, 32KB L1 instruction and data cache, 256KB L2 cache per core and 6MB L3 shared cache) with 16GB RAM and operating system Ubuntu Linux 64bit version 14.04.4. We used GNU's `g++` (4.8.4); optimized with flags `-O3 -march=native`.

For time measurements, we used `std::chrono::high_resolution_clock`, for generating random inputs, the Mersenne Twister pseudo-random generator `std::mt19937`. All time measurements were repeated with the same 20 deterministically chosen seeds – the displayed numbers are the average of these 20 runs. Moreover, for each time measurement, at least 128MB of data were sorted – if the array size is smaller, then for this time measurement several arrays have been sorted and the total elapsed time measured. Our running time plots all display the actual time divided by the number of elements to sort on the y-axis.

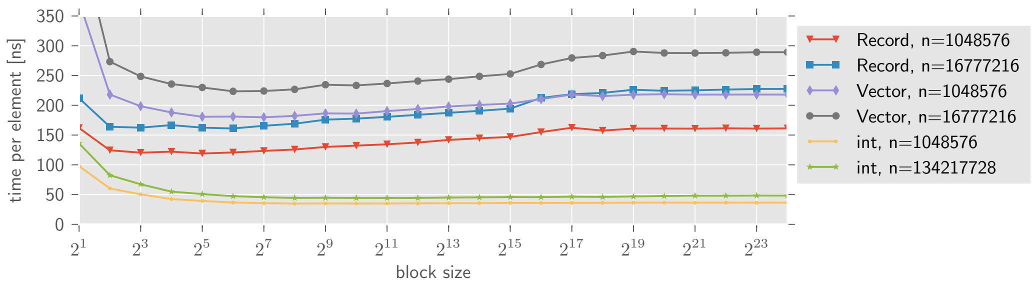


Figure 1 Different block sizes for random permutations.

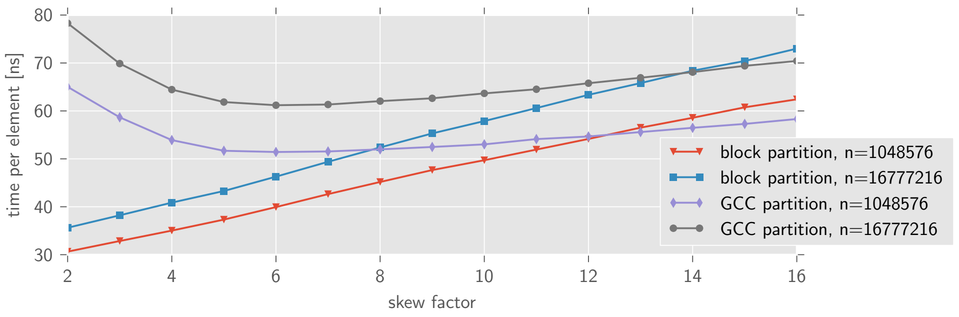


Figure 2 Sorting random permutations of 32-bit integers with skewed pivot. A skew factor k means that $\left\lfloor \frac{n}{k} \right\rfloor$-th element is chosen as pivot of an array of length n.

We performed our running time experiments with three different data types:

- `int`: signed 32-bit integers.
- `Vector`: 10-dimensional array of 64-bit floating-point numbers (`double`). The order is defined via the Euclidean norm – for every comparison the sums of the squares of the components are computed and then compared.
- `Record`: 21-dimensional array of 32-bit integers. Only the first component is compared.

The code of our implementation of BlockQuicksort as well as the other algorithms and our running time experiments is available at `https://github.com/weissan/BlockQuicksort`.

Different Block Sizes. Figure 1 shows experimental results on random permutations for different data types and block sizes ranging from 4 up to 2^{24}.

We see that for integers only at the end there is a slight negative effect when increasing the block size. Presumably this is because up to a block size of 2^{19}, still two blocks fit entirely into the L3 cache of the CPU. On the other hand for `Vector` a block size of 64 and for `Record` of 8 seem to be optimal – with a considerably increasing running time for larger block sizes.

As a compromise we chose to fix the block size to 128 elements for all further experiments. An alternative approach would be to choose a fixed number of bytes for one block and adapt the block size according to the size of the data elements.

Skewed Pivot Experiments. We repeated the experiments from [16] with skewed pivot for both the usual Hoare partitioner (`std::__unguarded_partition`, from the GCC implementation of `std::sort`) and our block partition method. For both partitioners we used our tuned Quicksort loop. The results can be seen in Figure 2: classic Quicksort benefits from skewed pivot, whereas BlockQuicksort works best with the exact median. Therefore, for BlockQuicksort it makes sense to invest more effort to find a good pivot.

Different Pivot Selection Methods. We implemented several strategies for pivot selection:

- median-of-three, median-of-five, median-of-twenty-three,
- median-of-three-medians-of-three, median-of-three-medians-of-five, median-of-five-medians-of-five: first calculate three (resp. five) times the median of three (resp. five) elements, then take the pivot as median of these three (resp. five) medians,
- median-of-$\sqrt{n}$.

All pivot selection strategies switch to median-of-three for small arrays. Moreover, the median-of-$\sqrt{n}$ variant switches to median-of-five-medians-of-five for arrays of length below 20000 (for smaller n even the number of comparisons was better with median-of-five-medians-of-five). The medians of larger samples are computed with `std::nth_element`.

Despite the results on skewed pivots Figure 2, there was no big difference between the different pivot selection strategies (for the results, see [9]). As expected, median-of-three was always the slowest for larger arrays. Median-of-five-medians-of-five was the fastest for `int` and median-of-$\sqrt{n}$ for `Vector`. We think that the small difference between all strategies is due to the large overhead for the calculation of the median of a large sample – and maybe because the array is rearranged in a way that is not favorable for the next recursive calls.

4.1 Comparison with other Sorting Algorithms

We compare variants of BlockQuicksort with the GCC implementation of `std::sort`[3] (which is known to be one of the most efficient Quicksort implementations – see e. g. [6]), Yaroslavskiy's dual-pivot Quicksort [31] (we converted the Java code of [31] to C++) and an implementation of Super Scalar Sample Sort [25] by Hübschle-Schneider[4]. For random permutations and random values modulo $\sqrt{n}$, we also test Tuned Quicksort [17] and three-pivot Quicksort implemented by Aumüller and Bingmann[5] from [4] (which is based on [19]) – for other types of inputs we omit these algorithms because of their poor behavior with duplicate elements.

Branch mispredictions. We experimentally determined the number of branch mispredictions of BlockQuicksort and the other algorithms with the *chachegrind* branch prediction profiler, which is part of the profiling tool *valgrind*[6]. The results of these experiments on random `int` data can be seen in Figure 3 – the y-axis shows the number of branch misprediction

[3] For the source code see e. g. `https://gcc.gnu.org/onlinedocs/gcc-4.7.2/libstdc++/api/a01462_source.html` – be aware that in newer versions of GCC the implementation is slightly different: the old version uses the first, middle and last element as sample for pivot selection, whereas the new version uses the *second*, middle and last element. For decreasingly sorted arrays the newer version works far better – for random permutations and increasingly sorted arrays, the old one is better. We used the old version for our experiment. The new version is included in some plots in [9] (Figures 9 and 10); this reveals a enormous difference between the two versions for particular inputs and underlines the importance of proper pivot selection.

[4] URL: `https://github.com/lorenzhs/ssssort/blob/b931c024cef3e6d7b7e7fd3ee3e67491d875e021/ssssort.h` – retrieved April 12, 2016

[5] URL: `http://eiche.theoinf.tu-ilmenau.de/Quicksort-experiments/` – retrieved March, 2016

[6] For more information on valgrind, see `http://valgrind.org/`. To perform the measurements we used the same Python script as in [11, 17], which first measures the number of branch mispredictions of the whole program including generation of test cases and then, in a second run, measures the number of branch mispredictions incurred by the generation of test cases.

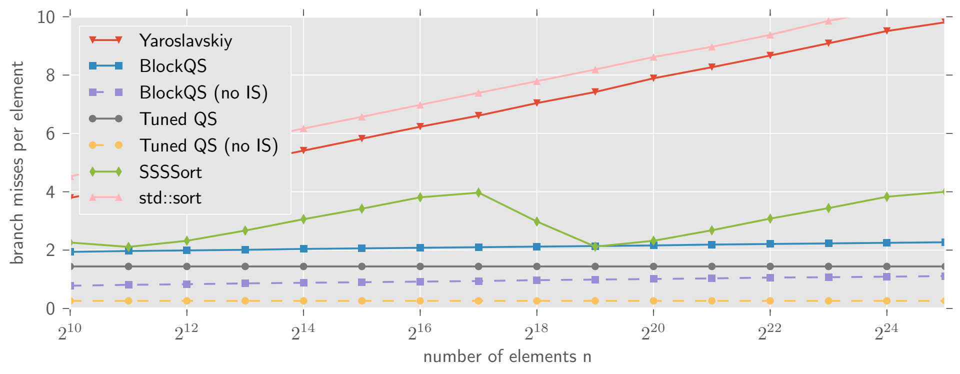


Figure 3 Number of branch mispredictions.

divided the the array size. We only display the median-of-three variant of BlockQuicksort since all the variants are very much alike. We also added plots of BlockQuicksort and Tuned Quicksort skipping final Insertionsort (i. e. the arrays remain partially unsorted).

We see that both `std::sort` and Yaroslavskiy's dual-pivot Quicksort incur $\Theta(n \log n)$ branch mispredictions. The up and down for Super Scalar Sample Sort presumably is because of the variation in the size of the arrays where the base case sorting algorithm `std::sort` is applied to. For BlockQuicksort there is an almost non-visible $n \log n$ term for the number of branch mispredictions. Indeed, we computed an approximation of $0.02n \log n + 1.75n$ branch mispredictions. Thus, the actual number of branch mispredictions is still better then our bounds in Theorem 1. There are two factors which contribute to this discrepancy: our rough estimates in the mentioned results, and that the actual branch predictor of a modern CPU might be much better than a static branch predictor. Also note that approximately one half of the branch mispredictions are incurred by Insertionsort – only the other half by the actual block partitioning and main Quicksort loop.

Finally, Figure 3 shows that Katajainen et al.'s Tuned Quicksort is still more efficient with respect to branch mispredictions (only $\mathcal{O}(n)$). This is no surprise since it does not need any checks whether buffers are empty etc. Moreover, we see that over 80% of the branch misses of Tuned Quicksort come from the final Insertionsort.

Running Time Experiments. In Figure 4 we present running times on random `int` permutations of different BlockQuicksort variants and the other algorithms including Katajainen's Tuned Quicksort and Aumüller and Bingmann's three-pivot Quicksort. The optimized BlockQuicksort variants need around 45ns per element when sorting 2^{28} elements, whereas `std::sort` needs 85ns per element – thus, there is a speed increase of 88% (i. e. the number of elements sorted per second is increased by 88%)[7].

The same algorithms are displayed in Figure 5 for sorting random `ints` between 0 and $\sqrt{n}$. Here, we observe that Tuned Quicksort is much worse than all the other algorithms (already for $n = 2^{12}$ it moves outside the displayed range). All variants of BlockQuicksort are faster than `std::sort` – the `duplicate check` (dc) version is almost twice as fast.

[7] In an earlier version of [9], we presented slightly different outcomes of our experiments. One reason it the usage of another random number generator. Otherwise, we introduced only minor changes in test environment – and no changes at all in the sorting algorithms themselves.

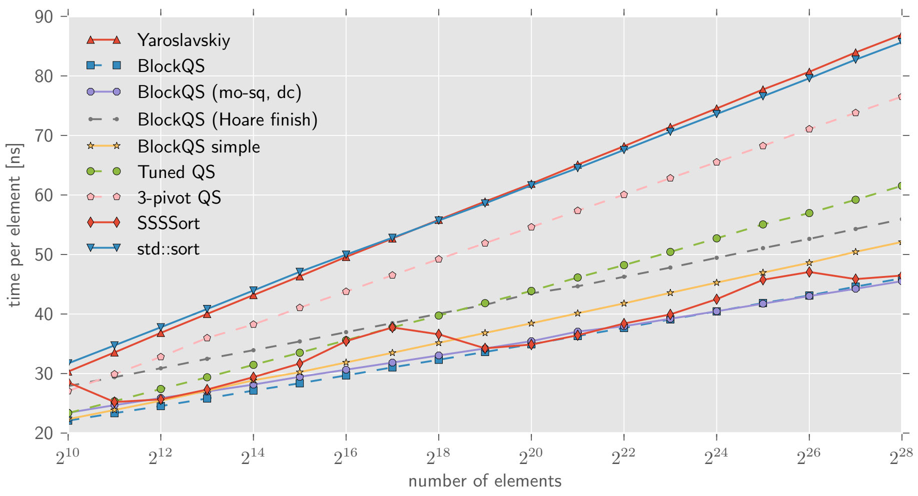


Figure 4 Random permutation of `int`.

Figure 6 presents experiments with data containing a lot of duplicates and having specific structures – thus, maybe coming closer to "real-world" inputs (although it is not clear what that means). Since here Tuned Quicksort and three-pivot Quicksort are much slower than all the other algorithms, we exclude these two algorithms from the plots. The array for the left plot contains long already sorted runs. This is most likely the reason that `std::sort` and Yaroslavskiy's dual-pivot Quicksort have similar running times to BlockQuicksort (for sorted sequences the conditional branches can be easily predicted what explains the fast running time). The arrays for the middle and right plot start with sorted runs and become more and more erratic; the array for the right one also contains a extremely high number of duplicates. Here the advantage of BlockQuicksort – avoiding conditional branches – can be observed again. In all three plots the check for duplicates (dc) established a considerable improvement.

In Figure 7, we show the results of selected algorithms for random permutations of `Vector` and `Record`. We conjecture that the good results of Super Scalar Sample Sort on `Records` are because of its better cache behavior (since `Record` are large data elements with very cheap comparisons). More running time experiments also on other machines and compiler flags can be found in [9].

More Statistics. Table 1 shows the number of branches taken / branch mispredicted as well as the instruction count and cache misses. Although `std::sort` has a much lower instruction count than the other algorithms, it induces most branch misses and (except Tuned Quicksort) most L1 cache misses (= L3 refs since no L2 cache is simulated). BlockQuicksort does not only have a low number of branch mispredictions, but also a good cache behavior – one reason for this is that Insertionsort is applied during the recursion and not at the very end.

5 Conclusions and Future Research

We have established an efficient in-place general purpose sorting algorithm, which avoids branch predictions by converting results of comparisons to integers. In the experiments we have seen that it is competitive on different kinds of data. Moreover, in several benchmarks it is almost twice as fast as `std::sort`. Future research might address the following issues:

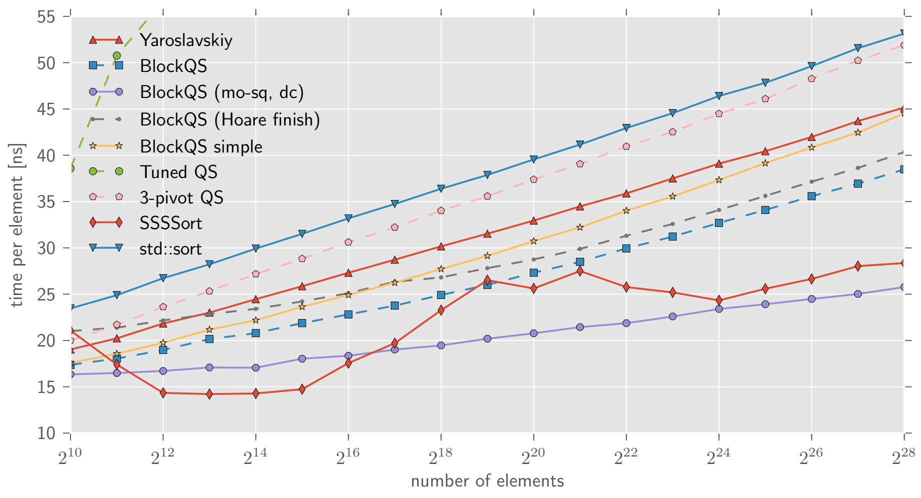


Figure 5 Random `int` values between 0 and $\sqrt{n}$.

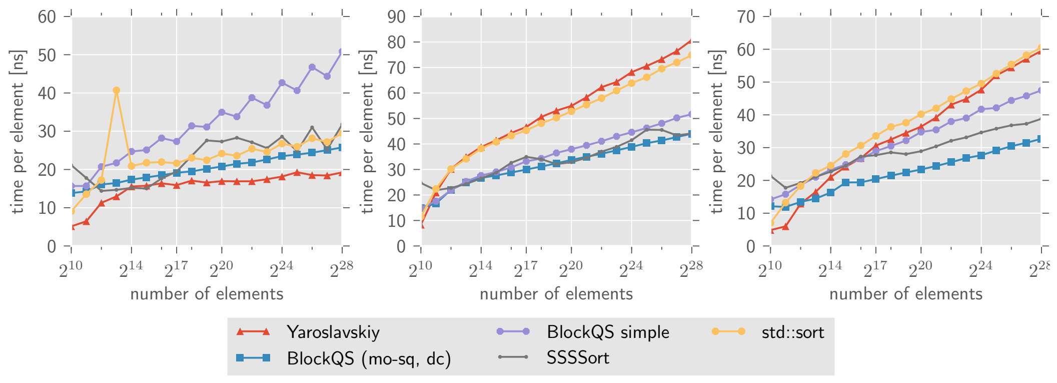


Figure 6 Arrays A of `int` with duplicates: *left*: $A[i] = i \mod \lfloor\sqrt{n}\rfloor$; *middle*: $A[i] = i^2 + n/2 \mod n$; *right*: $A[i] = i^8 + n/2 \mod n$. Since n is always a power of two, the value $n/2$ occurs approximately $n^{7/8}$ times in the last case.

- We used Insertionsort as recursion stopper – inducing a linear number of branch misses. Is there a more efficient recursion stopper that induces fewer branch mispredictions?
- More efficient usage of the buffers: in our implementation the buffers on average are not even filled half. To use the space more efficiently one could address the buffers cyclically and scan until one buffer is filled. By doing so, also both buffers could be filled in the same loop – however, with the cost of introducing additional overhead.
- The final rearrangement of the block partitioner is not optimal: for small arrays the similar problems with duplicates arise as for Lomuto's partitioner.
- Pivot selection strategy: though theoretically optimal, median-of-$\sqrt{n}$ pivot selection is not best in practice. Also we want to emphasize that not only the sample size but also the selection method is important (compare the different behavior of the two versions of `std::sort` observed in [9]). It might be even beneficial to use a fast pseudo-random generator (e. g. a linear congruence generator) for selecting samples for pivot selection.

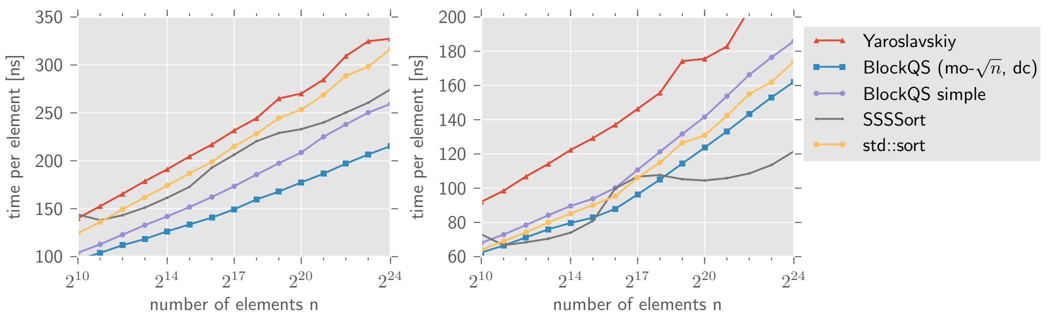


Figure 7 Random permutations – *left:* `Vector`; *right:* `Record`.

Table 1 Instruction count, branch and cache misses when sorting random `int` permutations of size $16777216 = 2^{24}$. All displayed numbers are divided by the number of elements.

algorithm	branches taken	branch misses	instructions	L1 refs	L3 refs	L3 misses
`std::sort`	37.81	10.23	174.82	51.96	1.05	0.41
SSSSort	16.2	3.87	197.06	68.47	0.82	0.5
Yaroslavskiy	52.92	9.51	218.42	59.82	0.79	0.27
BlockQS (mo-$\sqrt{n}$, dc)	20.55	2.39	322.08	89.9	0.77	0.27
BlockQS (mo5-mo5)	20.12	2.31	321.49	88.63	0.78	0.28
BlockQS	20.51	2.25	337.27	92.45	0.88	0.3
BlockQS (no IS)	15.38	1.09	309.85	84.66	0.88	0.3
Tuned QS	29.66	1.44	461.88	105.43	1.23	0.39
Tuned QS (no IS)	24.53	0.26	434.53	97.65	1.22	0.39

- Parallel versions: the block structure is very well suited for parallelism.
- A three-pivot version might be interesting, but efficient multi-pivot variants are not trivial: our first attempt was much slower.

Acknowledgments. Thanks to Jyrki Katajainen and Max Stenmark for allowing us to use their Python scripts for measuring branch mispredictions and cache missses and to Lorenz Hübschle-Schneider for his implementation of Super Scalar Sample Sort. We are also indebted to Jan Philipp Wächter for all his help with creating the plots, to Daniel Bahrdt for answering many C++ questions, and to Christoph Greulich for his help with the experiments.

References

1 D. Abhyankar and M. Ingle. Engineering of a quicksort partitioning algorithm. *Journal of Global Research in Computer Science*, 2(2):17–23, 2011.
2 ARMv8 Instruction Set Overview, 2011. Document number: PRD03-GENC-010197 15.0.
3 Martin Aumüller and Martin Dietzfelbinger. Optimal partitioning for dual pivot quicksort – (extended abstract). In *ICALP*, pages 33–44, 2013.
4 Martin Aumüller, Martin Dietzfelbinger, and Pascal Klaue. How good is multi-pivot quicksort? *CoRR*, abs/1510.04676, 2015.
5 Paul Biggar, Nicholas Nash, Kevin Williams, and David Gregg. An experimental study of sorting and branch prediction. *J. Exp. Algorithmics*, 12:1.8:1–39, 2008.
6 Gerth Stølting Brodal, Rolf Fagerberg, and Kristoffer Vinther. Engineering a cache-oblivious sorting algorithm. *J. Exp. Algorithmics*, 12:2.2:1–23, 2008.

7 Gerth Stølting Brodal and Gabriel Moruz. Tradeoffs between branch mispredictions and comparisons for sorting algorithms. In *WADS*, volume 3608 of *LNCS*, pages 385–395. Springer, 2005.

8 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 3nd edition, 2009.

9 Stefan Edelkamp and Armin Weiß. Blockquicksort: How branch mispredictions don't affect quicksort. *CoRR*, abs/1604.06697, 2016.

10 Amr Elmasry and Jyrki Katajainen. Lean programs, branch mispredictions, and sorting. In *FUN*, volume 7288 of *LNCS*, pages 119–130. Springer, 2012.

11 Amr Elmasry, Jyrki Katajainen, and Max Stenmark. Branch mispredictions don't affect mergesort. In *SEA*, pages 160–171, 2012.

12 Robert W. Floyd. Algorithm 245: Treesort 3. *Comm. of the ACM*, 7(12):701, 1964.

13 John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 5th edition, 2011.

14 Charles A. R. Hoare. Quicksort. *The Computer Journal*, 5(1):10–16, 1962.

15 Intel 64 and IA-32 Architecture Optimization Reference Manual, 2016. Order Number: 248966-032.

16 Kanela Kaligosi and Peter Sanders. How branch mispredictions affect quicksort. In *ESA*, pages 780–791, 2006.

17 Jyrki Katajainen. Sorting programs executing fewer branches. CPH STL Report 2263887503, Department of Computer Science, University of Copenhagen, 2014.

18 Donald E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison Wesley Longman, 2nd edition, 1998.

19 Shrinu Kushagra, Alejandro López-Ortiz, Aurick Qiao, and J. Ian Munro. Multi-pivot quicksort: Theory and experiments. In *ALENEX*, pages 47–60, 2014.

20 Anthony LaMarca and Richard E Ladner. The influence of caches on the performance of sorting. *J. Algorithms*, 31(1):66–104, 1999.

21 Conrado Martínez, Markus E. Nebel, and Sebastian Wild. Analysis of branch misses in quicksort. In *Workshop on Analytic Algorithmics and Combinatorics, ANALCO 2015, San Diego, CA, USA, January 4, 2015*, pages 114–128, 2015.

22 Conrado Martínez and Salvador Roura. Optimal Sampling Strategies in Quicksort and Quickselect. *SIAM J. Comput.*, 31(3):683–705, 2001. `doi:10.1137/S0097539700382108`.

23 David R. Musser. Introspective sorting and selection algorithms. *Software—Practice and Experience*, 27(8):983–993, 1997.

24 Charles Price. MIPS IV Instruction Set, 1995.

25 Peter Sanders and Sebastian Winkel. Super Scalar Sample Sort. In *ESA*, pages 784–796, 2004.

26 Robert Sedgewick. The analysis of quicksort programs. *Acta Inf.*, 7(4):327–355, 1977.

27 Robert Sedgewick. Implementing quicksort programs. *Commun. ACM*, 21(10):847–857, 1978.

28 Sebastian Wild and Markus E. Nebel. Average case analysis of java 7's dual pivot quicksort. In *ESA*, pages 825–836, 2012.

29 Sebastian Wild, Markus E. Nebel, and Ralph Neininger. Average case and distributional analysis of dual-pivot quicksort. *ACM Transactions on Algorithms*, 11(3):22:1–42, 2015.

30 J. W. J. Williams. Algorithm 232: Heapsort. *Communications of the ACM*, 7(6):347–348, 1964.

31 Vladimir Yaroslavskiy. Dual-Pivot Quicksort algorithm, 2009. URL: `http://codeblab.com/wp-content/uploads/2009/09/DualPivotQuicksort.pdf`.

Counting Linear Extensions: Parameterizations by Treewidth*

Eduard Eiben[1], Robert Ganian[†2], Kustaa Kangas[3], and Sebastian Ordyniak[4]

1 Algorithms and Complexity Group, TU Wien, Vienna, Austria
2 Algorithms and Complexity Group, TU Wien, Vienna, Austria
3 Helsinki Institute for Information Technology, Department of Computer Science, University of Helsinki, Finland
4 Algorithms and Complexity Group, TU Wien, Vienna, Austria

Abstract

We consider the #P-complete problem of counting the number of linear extensions of a poset (#LE); a fundamental problem in order theory with applications in a variety of distinct areas. In particular, we study the complexity of #LE parameterized by the well-known decompositional parameter treewidth for two natural graphical representations of the input poset, i.e., the cover and the incomparability graph. Our main result shows that #LE is fixed-parameter intractable parameterized by the treewidth of the cover graph. This resolves an open problem recently posed in the Dagstuhl seminar on Exact Algorithms. On the positive side we show that #LE becomes fixed-parameter tractable parameterized by the treewidth of the incomparability graph.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes, G.2.1 Combinatorics

Keywords and phrases Partially ordered sets, Linear extensions, Parameterized Complexity, Structural parameters, Treewidth

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.39

1 Introduction

Counting the number of linear extensions of a poset is a fundamental problem of order theory that has applications in a variety of distinct areas such as sorting [30], sequence analysis [25], convex rank tests [27], sampling schemes of Bayesian networks [28], and preference reasoning [24]. Determining the exact number of linear extensions of a given poset is known to be #P-complete [6] already for posets of height at least 3. Informally, #P-complete problems are as hard as counting the number of accepting paths of any polynomial time nondeterministic Turing machine, implying that such problems are not tractable unless P = NP. The currently fastest known method for counting linear extensions of a general n-element poset is by dynamic programming over the lattice of downsets and runs in time $\mathcal{O}(2^n \cdot n)$ [10]. Polynomial time algorithms have been found for various special cases such as series-parallel posets [26] and posets whose cover graph is a (poly)tree [2]. Fully polynomial time randomized approximation schemes are known for estimating the number of linear extensions [13, 7].

Due to the inherent difficulty of the problem, it is natural to study whether it can be solved efficiently by exploiting the structure of the input poset. In this respect, the

* The authors acknowledge support by the Austrian Science Fund (FWF, projects P26696 and W1255-N23).
† Robert Ganian is also affiliated with FI MU, Brno, Czech Republic.

parameterized complexity framework [12, 9] allows a refined view of the interactions between various forms of structure in the input and the running time of algorithms. The idea of the framework is to measure the complexity of problems not only in terms of input sizes, but also with respect to an additional numerical parameter. The goal is then to develop so-called *fpt algorithms*, which are algorithms that run in time $f(k)n^{\mathcal{O}(1)}$ where n is the input size and f is a computable function depending only on the parameter k. A less favorable outcome is a so-called XP algorithm, which runs in time $n^{f(k)}$; the existence of such algorithms then gives rise to the respective complexity classes FPT (*fixed-parameter tractable*) and XP.

The first steps in this general direction have been taken, e.g., in [19], using the decomposition diameter as a parameter, in [15] using a parameter called activity for N-free posets, and very recently in [22], where the treewidth of the so-called cover graph was considered as a parameter. Also the exact dynamic programming algorithm [10] can be shown to run in time $\mathcal{O}(n^w \cdot w)$ for a poset with n elements and width w (the size of the largest anti-chain). Interestingly, none of these efforts has so far led to an fpt algorithm.

We believe that this uncertainty about the exact complexity status of counting linear extensions with respect to these various parameterizations is at least partly due to the fact that we deal with a counting problem whose decision version is trivial, i.e., every poset has at least one linear extension. This fact makes it considerably harder to show that the problem is fixed-parameter intractable; in particular, the usual approach for counting problems based on parsimonious reductions (i.e., polynomial time one-one reductions) fails. On the other hand, the same predicament makes studying the complexity of counting linear extensions significantly more interesting, as noted also by Flum and Grohe [16]:

The theory gets interesting with those counting problems that are harder than their corresponding decision versions.

1.1 Results

In this paper we study the complexity of counting linear extensions when the parameter is the treewidth – a fundamental graph parameter which has already found a plethora applications in many areas of computer science [18, 17, 29]. In particular, we settle the fixed-parameter (in)tractability of the problem when parameterizing by the treewidth of two of the most prominent graphical representations of posets, the cover graph (also called the Hasse diagram) and the incomparability graph.

Our main result then provides the first evidence that the problem does not allow for an fpt algorithm parameterized by the treewidth of the cover graph unless FPT = W[1]. We remark that this complements the XP algorithm of [22] and resolves an open problem recently posed in the Dagstuhl seminar on Exact Algorithms [21]. The result is based on a so-called *fpt turing reduction* from Equitable Coloring parameterized by treewidth [14], and combines a counting argument with a fine-tuned construction to link the number of linear extensions with the existence of an equitable coloring. To the best of our knowledge, this is the first time this technique has been used to show fixed-parameter intractability of a counting problem.

We complement this negative result by obtaining an fpt algorithm for the problem when the parameter is the treewidth of the incomparability graph of the poset. To this end, we use the so-called *combined graph* (also called the cover-incomparability graph [5]) of the poset, which is obtained from the cover graph by adding the edges of the incomparability graph. We employ a special normalization procedure on a decomposition of the incomparability graph to show that the treewidth of the combined graph must be bounded by the treewidth of the incomparability graph. Once this is established, the result follows by giving a formulation of the problem in *Monadic Second Order Logic* and applying an extension of Courcelle's Theorem for counting.

The paper is organized as follows. Section 2 introduces the required preliminaries and notation. Section 3 is then dedicated to proving the fixed-parameter intractability of the problem when parameterized by the treewidth of the cover graph, and the subsequent Section 4 presents our positive results for the problem. Concluding notes are then provided in Section 5.

2 Preliminaries

For standard terminology in graph theory, such as the notions of a graph, digraph, path, etc. we refer readers to [11]. Given a graph G, we let $V(G)$ denote its vertex set and $E(G)$ its edge set. The (open) neighborhood of a vertex $x \in V(G)$ is the set $\{y \in V(G) : (x,y) \in E(G)\}$ and is denoted by $N(x)$. The closed neighborhood $N[v]$ of x is defined as $N(v) \cup \{v\}$. A path between two disjoint vertex sets $A, B \subseteq V(G)$ is a path with one endpoint in A, one endpoint in B, and all internal vertices disjoint from $A \cup B$. A set $X \subseteq V(G)$ is a separator in G if $G - X$ contains at least two connected components.

We use $[i]$ to denote the set $\{0, 1, \ldots, i\}$. The following fact about prime numbers will also be useful later.

▶ **Fact 1** ([6]). *For any $n \geq 4$, the product of primes strictly between n and n^2 is at least $n!2^n$.*

2.1 Treewidth

A *tree-decomposition* of a graph G is a pair $(T, \mathcal{X} = \{X_t\}_{t \in V(T)})$, where T is a rooted tree whose every vertex t is assigned a vertex subset $X_t \subseteq V(G)$, called a *bag*, such that the following properties hold:

(T1) $\cup_{t \in V(T)} X_t = V(G)$,

(T2) for every $u \in V(G)$, the set $T_u = \{t \in V(T) : u \in X_t\}$ induces a connected subtree of T (*monotonicity*), and

(T3) for each $uv \in E(G)$ there exists $t \in V(T)$ such that $u, v \in X_t$.

To distinguish between the vertices of the tree T and the vertices of the graph G, we will refer to the vertices of T as *nodes*. The *width* of the tree-decomposition $\mathcal{T}$ is $\max_{t \in V(T)} |X_t| - 1$. The *treewidth* of G, $\mathrm{tw}(G)$, is the minimum width over all tree-decompositions of G.

A path-decomposition is a tree-decomposition where each node of T has degree at most 2, and the notion of *pathwidth* is then defined analogously to treewidth. A tree-decomposition $\mathcal{T} = (T, \mathcal{X})$ is *nice* if T contains a root r, the root and all leaves have empty bags, and each non-leaf node belongs to one of three categories: **Introduce**, **Forget**, **Join** (see, e.g., [9]). A nice tree-decomposition (path-decomposition) can be obtained from a tree-decomposition (path-decomposition) of the same width in polynomial time [23]. Observe that any path-decomposition can be fully characterized by the order of appearance of its bags along T, and hence we will consider succinct representations of path-decompositions in the form $\mathcal{Q} = (Q_1, \ldots, Q_d)$, where Q_i is the i-th bag in $\mathcal{Q}$.

We list some useful facts about treewidth and pathwidth.

▶ **Fact 2** ([3, 4]). *There exists an algorithm which, given a graph G and an integer k, runs in time $\mathcal{O}(k^{\mathcal{O}(k^3)}n)$ and either outputs a tree-decomposition of G of width at most k or correctly* *identifies that* $\mathrm{tw}(G) > k$*. Furthermore, there exists an algorithm which, given a graph* G *and an integer* k*, runs in time* $\mathcal{O}(k^{\mathcal{O}(k^3)}n)$ *and either outputs a path-decomposition of* G *of width at most* k *or correctly identifies that* $\mathrm{pw}(G) > k$*.*

▶ **Fact 3** (Folklore). *Let* $\mathcal{T}$ *be a tree-decomposition of* G *and* $t \in V(T)$*. Then each connected component of* $G - X_t$ *lies in a single subtree of* $T - t$*. In particular, for each connected component* C *of* $G - X_t$ *there exists a subtree* T' *of* $T - t$ *such that for each vertex* $a \in C$ *there exists* $t_a \in V(T')$ *such that* $a \in X_{t_a}$*.*

We note that if G is a directed graph, then $\mathrm{tw}(G)$ and a tree-decomposition of G refers to the treewidth and a tree-decomposition of the underlying undirected graph of G, i.e., the undirected graph obtained by replacing each directed edge with an edge.

2.2 Monadic Second Order Logic

We consider *Monadic Second Order* (MSO) logic on (edge-)labeled directed graphs in terms of their incidence structure whose universe contains vertices and edges; the incidence between vertices and edges is represented by a binary relation. We assume an infinite supply of *individual variables* $x, x_1, x_2, \dots$ and of *set variables* $X, X_1, X_2, \dots$ The *atomic formulas* are Vx ("x is a vertex"), Ey ("y is an edge"), Ixy ("vertex x is incident with edge y"), Hxy ("vertex x is the head of the edge y"), Txy ("vertex x is the tail of the edge y"), $x = y$ (equality), $x \neq y$ (inequality), $P_a x$ ("vertex or edge x has label a"), and Xx ("vertex or edge x is an element of set X"). *MSO formulas* are built up from atomic formulas using the usual Boolean connectives ($\neg, \wedge, \vee, \rightarrow, \leftrightarrow$), quantification over individual variables ($\forall x$, $\exists x$), and quantification over set variables ($\forall X$, $\exists X$).

Let $\Phi(X)$ be an MSO formula with a free set variable X. For a labeled graph $G = (V, E)$ and a set $S \subseteq E$ we write $G \models \Phi(S)$ if the formula Φ holds true on G whenever X is instantiated with S.

The following result (an extension of the well-known Courcelle's Theorem [8]) shows that if G has bounded treewidth then we can count the number of sets S with $G \models \Phi(S)$.

▶ **Fact 4** ([1]). *Let* $\Phi(X)$ *be an MSO formula with a free set variable* X *and* w *a constant. Then there is a linear-time algorithm that, given a labeled directed graph* $G = (V, E)$ *of treewidth at most* w*, outputs the number of sets* $S \subseteq E$ *such that* $G \models \Phi(S)$*.*

2.3 Posets

A *partially ordered set* (*poset*) $\mathcal{P}$ is a pair $(P, \leq^P)$ where P is a set and $\leq^P$ is a reflexive, antisymmetric, and transitive binary relation over P. The *size* of a poset $\mathcal{P} = (P, \leq^P)$ is $|\mathcal{P}| := |P|$. We say that p *covers* p' for $p, p' \in P$, denoted by $p' \lhd^P p$, if $p' \leq^P p$, $p \neq p'$, and for every p'' with $p' \leq^P p'' \leq^P p$ it holds that $p'' \in \{p, p'\}$. We say that p and p' are *incomparable* (in $\mathcal{P}$), denoted $p \parallel^P p'$, if neither $p \leq^P p'$ nor $p' \leq^P p$.

A *chain* C of $\mathcal{P}$ is a subset of P such that $x \leq^P y$ or $y \leq^P x$ for every $x, y \in C$. An *antichain* A of $\mathcal{P}$ is a subset of P such that for all $x, y \in A$ it is true that $x \parallel^P y$. A family $C_1, \dots, C_\ell$ of pairwise disjoint subsets of P forms a *total order* if for each $i, j \in [\ell]$ and each $a \in C_i$, $b \in C_j$, it holds that $a \leq b$ iff $i < j$. Furthermore, for each $i \in [\ell - 1]$ we say that C_i and C_{i+1} are *consecutive*. We call a poset $\mathcal{P}$ such that every two elements of $\mathcal{P}$ are comparable a *linear order*. A *linear extension* of a poset $\mathcal{P} = (P, \leq^P)$ is a reflexive, antisymmetric, and transitive binary relation $\preceq$ over P such that $x \preceq y$ whenever $x \leq^P y$ and a poset $\mathcal{P}^* = (P, \preceq)$ is a linear order.

We denote the number of linear extensions of $\mathcal{P}$ by $e(\mathcal{P})$. For completeness, we provide a formal definition of the problem of counting the number of linear extensions below.

#LE
Instance: A poset $\mathcal{P}$.
Task: Compute $e(\mathcal{P})$.

We consider the following graph representations of a poset $\mathcal{P} = (P, \leq^{\mathcal{P}})$. The *cover graph* of $\mathcal{P}$, denoted $C(\mathcal{P})$, is the directed graph with vertex set P and edge set $\{(a, b) \mid a \lhd b\}$. The *incomparability graph* of $\mathcal{P}$, denoted $I(\mathcal{P})$, is the undirected graph with vertex set P and edge set $\{\{a, b\} \mid a \parallel b\}$. The *combined graph* of $\mathcal{P}$, denoted $I_C(\mathcal{P})$, is the directed graph with vertex set P and edge set $\{(a, b) \mid (a \lhd b) \vee (a \parallel b)\}$; observe that $I_C(\mathcal{P})$ can be obtained by taking disjoint union of the edge sets of $C(\mathcal{P})$ and $I(\mathcal{P})$ and then replacing undirected edges by two directed ones. Finally, the *poset graph* of $\mathcal{P}$, denoted $P_G(\mathcal{P})$, is the directed graph with vertex set P and edge set $\{(a, b) \mid a \leq b\}$. We will use the following known fact about tree-decompositions and path-decompositions of incomparability graphs.

▶ **Fact 5** ([20, Theorem 2.1]). *Let $\mathcal{P}$ be a poset. Then* $\mathrm{tw}(I(\mathcal{P})) = \mathrm{pw}(I(\mathcal{P}))$.

▶ **Corollary 6** (of Fact 2 and 5). *Let $\mathcal{P}$ be a poset and $k = \mathrm{tw}(I(\mathcal{P}))$. Then it is possible to compute a nice path-decomposition $\mathcal{Q}$ of $I(\mathcal{P})$ of width at most k in time $\mathcal{O}(k^{\mathcal{O}(k^3)}n)$.*

2.4 Parameterized Complexity

We refer the reader to [12, 9, 16] for an in-depth introduction to parameterized complexity. In particular, we will need the notions of *parameterized (decision) problem*, the complexity classes $W[1]$ and *FPT*, *fpt algorithm*, and *fpt turing reduction*. Informally, recall that an fpt turing reduction from problem $\mathcal{A}$ to problem $\mathcal{B}$ is an fpt-algorithm that solves $\mathcal{A}$ using an oracle for $\mathcal{B}$. A *parameterized counting problem* $\mathcal{P}$ is a function $\Sigma^* \times \mathbb{N} \to \mathbb{N}$ for some finite alphabet Σ. We call a parameterized counting problem $\mathcal{P}$ *fixed-parameter tractable* (FPT) if $\mathcal{P}$ can be computed in time $f(k) \cdot |x|^{\mathcal{O}(1)}$ where f is an arbitrary computable function and (x, k) is the instance. To avoid confusion, we remark that there also exists the complexity class #W[1] which is an analog of #P for parameterized counting problems. Our main negative result is based on an fpt turing reduction from the following fairly well-known W[1]-hard decision problem [14].

Equitable Coloring[tw]
Instance: A graph G and an integer r.
Parameter: $\mathrm{tw}(G) + r$.
Question: Does G admit a proper r-coloring such that the number of vertices in any two color classes differ by at most one?

We denote by $\#\mathrm{EC}(G, r)$ the number of equitable colorings of graph G with r colors.

3 Fixed-Parameter Intractability of Counting Linear Extensions

The goal of this section is to prove Theorem 7, stated below.

▶ **Theorem 7.** *#LE parameterized by the treewidth of the cover graph of the input poset does not admit an fpt algorithm unless* W[1]=FPT.

We begin by giving a brief overview of the proof, whose general outline follows the #P-hardness proof of the problem [6]. However, since our parameter is treewidth, we needed to reduce from a problem that is not fixed-parameter tractable parameterized by treewidth. Consequently, instead of reducing from SAT, we will use Equitable Coloring. This made the reduction considerably more complicated and required the introduction of novel gadgets, which allow us to encode the problem without increasing the treewidth too much.

The proof is based on solving an instance (G, r) of Equitable Coloring[tw] in FPT time using an oracle that solves #LE in FPT time parameterized by the treewidth of the cover graph (i.e., an fpt turing reduction). The first step is the construction of an auxiliary poset $\mathcal{P}(G, r)$ of size $2(r-1)|V(G)| + (r^2-1)|E(G)|$. Then, for a given sufficiently large (polynomially larger than $|V(G)|$) prime number p, we show how to construct a poset $\mathcal{P}(G, r, p)$ such that $e(\mathcal{P}(G, r, p)) \equiv e(\mathcal{P}(G, r)) \cdot \#\mathrm{EC}(G, r) \cdot A_p \mod p$, where A_p is a constant that depends on p and is not divisible by p. Therefore, if we choose a prime p that does not divide $e(\mathcal{P}(G, r)) \cdot \#\mathrm{EC}(G, r)$, then $e(\mathcal{P}(G, r, p))$ will not be divisible by p. Using Fact 1 we show that if $\#\mathrm{EC}(G, r) \neq 0$, then there always exists a prime p within a specified polynomial range of $|V(G)|$ such that p does not divide $e(\mathcal{P}(G, r)) \cdot \#\mathrm{EC}(G, r)$.

From the above, it follows that there exists an equitable coloring of G with r colors if and only if, for at least one prime p within a specified (polynomial) number range, the number of linear extensions of $\mathcal{P}(G, r, p)$ is not divisible by p. Moreover, we show that all inputs for the oracle will have size polynomial in the size of G and treewidth bounded by polynomial in $\mathrm{tw}(G) + r$. Before proceeding to a formal proof of Theorem 7, we state two auxiliary lemmas which will be useful for counting linear extensions later in the proof.

▶ **Lemma 8.** *If a poset $\mathcal{P}$ is a disjoint union of posets $\mathcal{P}_1, \dots, \mathcal{P}_k$ for some positive integer k, then*

$$e(\mathcal{P}) = \frac{(\sum_{i=1}^{k} |\mathcal{P}_i|)!}{\prod_{i=1}^{k} |\mathcal{P}_i|!} \prod_{i=1}^{k} e(\mathcal{P}_i)\,.$$

▶ **Lemma 9.** *Let p be a prime number and $\mathcal{Q}$ be a connected component of poset $\mathcal{P}$ such that $|\mathcal{Q}| = p - 1$. If the number of linear extensions of $\mathcal{P}$ is not divisible by p, then the number of elements in each connected component of $\mathcal{P}$ other than $\mathcal{Q}$ is divisible by p.*

We now proceed to the proof of the theorem.

Proof of Theorem 7. The proof is structured into the construction of $\mathcal{P}(G, r)$, the construction of $\mathcal{P}(G, r, p)$, establishing the desired properties of $\mathcal{P}(G, r, p)$ and $\mathcal{P}(G, r)$, and the conclusion.

Construction of $\mathcal{P}(G, r)$ and the main gadget

Let (G, r) be an instance of Equitable Coloring[tw] such that $|V(G)|$ is divisible by r (if this is not the case, then this can be enforced by padding the instance with isolated vertices, see also [14]). We begin by constructing the poset $\mathcal{P}(G, r)$, which will play an important role later on. For every vertex v of $V(G)$ we create $2(r-1)$ elements denoted $v_{i,j}$, where $1 \leq i \leq r-1$ and $j \in \{0, 1\}$, such that the only dependencies in the poset between these elements are $v_{i,1} \leq v_{i,0}$ for all $v \in V(G)$, for all $i \in \{1, \dots, r-1\}$. For every edge $e = uv \in E(G)$ we create $r^2 - 1$ pairwise-incomparable elements $e_{i,j}$, such that $(i, j) \in (\{0, \dots, r-1\}^2 \setminus \{(0, 0)\})$. The dependencies of $e_{i,j}$ are: if $i > 0$ then $u_{i,0} \leq e_{i,j}$, and if $j > 0$ then $v_{j,0} \leq e_{i,j}$ (see also Fig. 1).

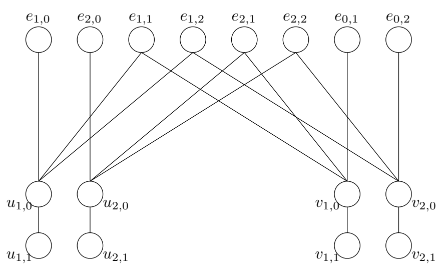


Figure 1 The cover graph for an edge $e = uv$ in $\mathcal{P}(G, 3)$.

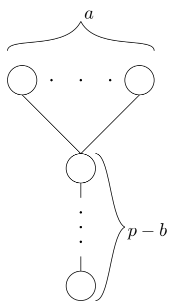


Figure 2 An (a, b)-flower.

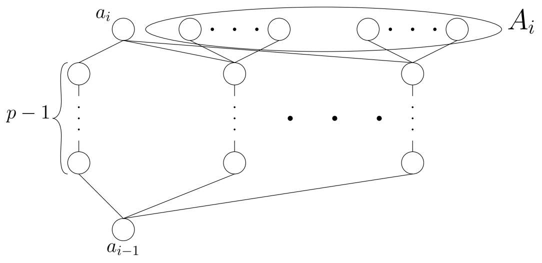


Figure 3 Each level consists of a chain of length $p - 1$ and a few flowers. The set of petals associated with level L_i is denoted by A_i.

Let us now fix a prime number p such that p does not divide $e(\mathcal{P}(G, r))$ and $p > 2r|V(G)| + r^2|E(G)|$. The main gadget in our reduction is a so-called (a, b)-*flower*, which consists of an antichain of a vertices (called the *petals*) covering a chain of $p - b$ elements (called the *stalk*); see Fig. 2. Due to Lemma 9, (a, b)-flowers will later allow us to force a choice of exactly b vertices out of a.

Construction of $\mathcal{P}(G, r)$

Let G be a graph, r be an integer and p be a prime number as above. Recall that $|V(G)|$ is divisible by r and let $s = \frac{|V(G)|}{r}$ (note that this implies that each color in an equitable coloring of G must occur precisely s times in G). We proceed with a description of the poset $\mathcal{P}(G, r, p)$. The poset $\mathcal{P}(G, r, p)$ is split into $r + 3$ "*levels*" $L_1, \ldots, L_{r+3}$ by linearly ordered elements $a_0 \leq a_1 \leq \cdots \leq a_{r+2} \leq a_{r+3}$, called the *anchors*. Each of these levels, besides L_{r+3}, will consist of some flowers and a chain of $p - 1$ elements which we call a *stick*; each of these flowers and the stick will always be pairwise incomparable. The anchors a_0 and a_{r+3} are the unique minimum and maximum elements, respectively. The stick and all the stalks of flowers in level L_i will always lie between two consecutive elements a_{i-1} and a_i, and the petals of these flowers will be incomparable with a_i as well as some anchors above that (as defined later). Observe that while the relative position of any stalk and any anchor is fixed in every linear extension, petals can be placed above a_i.

We say that a flower (or its stalk, petals, or elements) is *associated* with the level in which it is constructed, i.e., with the level L_i such that $a_{i-1} \leq c \leq a_i$ for stalk elements c and $a_{i-1} \leq d$ and $d \parallel a_i$ for petals d. We denote the set of all petals associated with level L_i as A_i (see Fig. 3).

For the construction, it will be useful to keep in mind the following intended goal: whenever an (a, b)-flower is placed in level i, it will force the selection of precisely b petals (from its total of a petals), where selected elements remain on level i (i.e., between a_{i-1} and a_i) in the linear extension and unselected elements are moved to level $r + 2$ (i.e., between

a_{r+2} and a_{r+3}) in the linear extension. We will later show that the total number of linear extensions which violate this goal must be divisible by p, and hence such extensions can all be disregarded modulo p.

The first r levels are so-called *color class* levels, each representing one color class. We use these levels to make sure that every color class contains exactly s vertices. Aside from the stick, each such level contains a single $(|V(G)|, s)$-flower. Recall that the stalk and the stick on level $1 \leq i \leq r$ both lie between anchors a_{i-1} and a_i, and that the stick and the flower are incomparable. We associate each petal of the flower at level L_i with a unique vertex $v \in V(G)$ and denote the petal v_i. Each petal v_i will be incomparable with all anchors above a_{i-1} up to a_{r+3}, i.e., $v_i \parallel a_j$ for $i \leq j \leq r+2$ and $v_i \leq a_{r+3}$. Intuitively, the flower in each color class level will later force a choice of s vertices to be assigned the given color.

Level L_{r+1} is called the *vertex* level and consists of one stick and $|V(G)|$-many $(r,1)$-flowers; the purpose of this level is to ensure that every vertex is assigned exactly one color. Each flower is associated with one vertex $v \in V(G)$ and we denote the petals of the flower associated with vertex v as v^i for $1 \leq i \leq r$. We set $v_i \leq v^i$ for all $v \in V(G)$ and $1 \leq i \leq r$.

Level L_{r+2} is called the *edge* level, and its purpose is to ensure that the endpoints of every edge have a different color. It consists of a stick and $|E(G)|$-many $(r^2, 1)$-flowers. Each flower is associated with one edge $e = uv \in V(G)$ and we denote the petals of the flower associated with e as $e_{i,j}$ for $1 \leq i \leq r$ and $1 \leq j \leq r$. Moreover, for edge $e = uv$ we set $u^i \leq e_{i,j}$, $v^j \leq e_{i,j}$, and we set $a_{r+2} \leq e_{i,j}$ whenever $i = j$. Observe that this forces any petal $e_{i,i}$ to lie between a_{r+2} and a_{r+3} in every linear extension (i.e., prevents $e_{i,i}$ from being "selected").

Level L_{r+3} is called the *trash level.* It does not contain any new elements in the poset, but it plays an important role in the reduction: we will later show that any petals which are interpreted as "not selected" must be located between a_{r+2} and a_{r+3} in any linear extension that is not automatically "canceled out" due to counting modulo p.

A high-level overview of the whole constructed poset $\mathcal{P}(G, r, p)$ is presented in Fig. 4.

Establishing the desired properties of $\mathcal{P}(G, r, p)$ and $\mathcal{P}(G, r)$

We begin by formalizing the notion of selection. Let a *configuration* be a partition ϕ of petals of all flowers into $r+3$ sets $L_1^\phi, \ldots, L_{r+3}^\phi$. Let Φ denote a set of all configurations. We say that a linear extension $\preceq$ of $\mathcal{P}(G, r, p)$ *respects* the configuration ϕ if $L_1^\phi \preceq a_1 \preceq L_2^\phi \preceq a_2 \preceq \cdots \preceq a_{r+2} \preceq L_{r+3}^\phi$ and we denote the set of all linear extensions of $\mathcal{P}(G, r, p)$ that respects ϕ by $\mathcal{L}^\phi$. We say that a configuration ϕ is *consistent* if $\mathcal{L}^\phi$ is non-empty; this merely means that $L_1^\phi \leq a_1 \leq L_2^\phi \leq a_2 \leq \cdots \leq a_{r+2} \leq L_{r+3}^\phi$ does not violate any inequalities in $\mathcal{P}(G, r, p)$. Observe that if ϕ is consistent, then $\mathcal{L}^\phi$ is exactly the set of linear extension of the partial order $\mathcal{P}^\phi(G, r, p)$, where $\mathcal{P}^\phi(G, r, p)$ is obtained by enriching $\mathcal{P}(G, r, p)$ with the relations $L_1^\phi \leq a_1 \leq L_2^\phi \leq a_2 \leq \cdots \leq a_{r+2} \leq L_{r+3}^\phi$ and performing transitive closure (in other words, $\mathcal{P}^\phi(G, r, p)$ is obtained by enforcing ϕ onto $\mathcal{P}(G, r, p)$).

Since every linear extension of $\mathcal{P}(G, r, p)$ respects exactly one configuration, it is easy to see that $e(\mathcal{P}(G, r, p)) = \sum_{\phi \in \Phi} |\mathcal{L}^\phi| = \sum_{\phi \in \Phi} e(\mathcal{P}^\phi(G, r, p))$. Intuitively, a configuration ϕ *contributes* to the above sum modulo p if $e(\mathcal{P}^\phi(G, r, p))$ is not divisible by p. We shall prove that the only configurations which contribute to this sum modulo p are those where from every (a, b)-flower there are exactly b petals in the same level as the stalk, and the remaining $a - b$ petals are in the trash. Furthermore, in each configuration ϕ which contributes to the above sum modulo p, the petals in L_{r+1}^ϕ represent a proper equitable coloring of G with r colors, and each such configuration is respected by the same number of linear extensions.

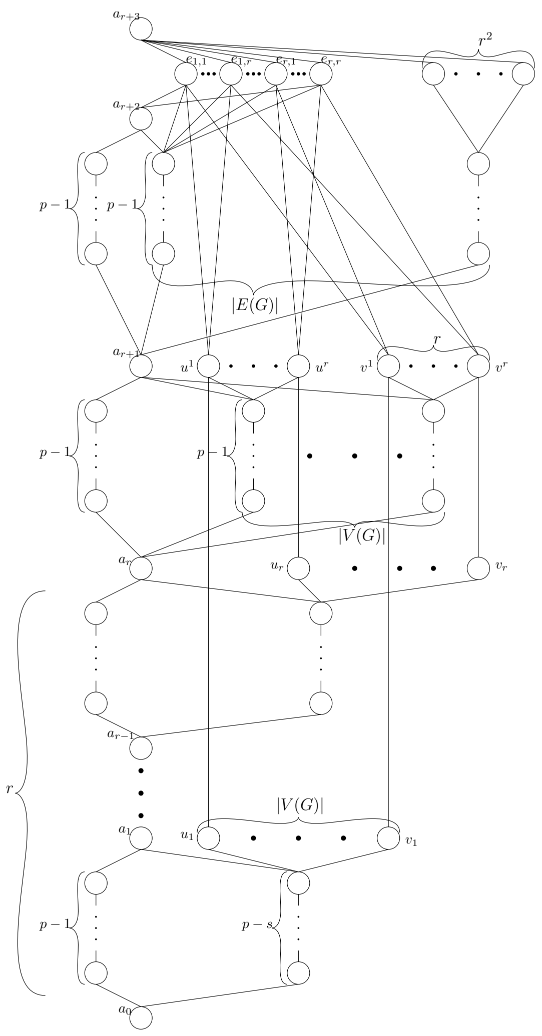


Figure 4 The cover graph of $\mathcal{P}(G,r,p)$. The edge e is the edge in G between vertices u and v.

Let us first remark that for any configuration ϕ, the anchors $a_0, a_1, \dots, a_{r+3}$ are comparable to all elements of $\mathcal{P}^\phi(G,r,p)$. Now, let $\mathcal{P}^\phi_{L_i}$ be the poset induced by all elements $e \in \mathcal{P}^\phi(G,r,p)$ such that $a_{i-1} \le e \le a_i$. It is readily seen that $e(\mathcal{P}^\phi(G,r,p)) = \prod_{i=1}^{r+3} e(\mathcal{P}^\phi_{L_i})$. We proceed by stating a series of claims about our construction.

▶ **Claim 10.** *For each $i \in \{1,\dots,r\}$, it holds that either $e(\mathcal{P}^\phi_{L_i}) \equiv 0 \mod p$, or $e(\mathcal{P}^\phi_{L_i}) = s!\binom{2p-1}{p}$ and L^ϕ_i contains exactly s petals of A_i and no other petals.*

Proof of the claim. Assume that $e(\mathcal{P}^\phi_{L_i}) \not\equiv 0 \mod p$ and recall that level L_i contains a stick, which is a chain of $p-1$ elements that is incomparable with all elements of $\mathcal{P}^\phi_{L_i}$ in every configuration ϕ. By Lemma 9 this implies that every connected component of $\mathcal{P}^\phi_{L_i}$ has size divisible by p. Clearly, L^ϕ_i contains only those stalks that are associated with the level L_i, and it contains all such stalks. It is readily seen from the construction that any petal in $\cup_{j<i} A_j$ would necessarily form a component of size one in $\mathcal{P}^\phi_{L_i}$. Hence, $\mathcal{P}^\phi_{L_i}$ contains only elements associated with level L_i, namely elements of the chain with $p-1$ vertices and elements of a $(|V(G)|, s)$-flower. Moreover, by Lemma 9 and the fact that $|V(G)|+p-s < 2p$, each such flower has exactly p elements in level $\mathcal{P}^{\phi}_{L_i}$. Since the $p-s$ elements of the stalk must be in $\mathcal{P}^{\phi}_{L_i}$, the poset $\mathcal{P}^{\phi}_{L_i}$ contains exactly s elements of A_i. Clearly, the number of linear extensions of the petals of the $(|V(G)|, s)$-flower in $\mathcal{P}^{\phi}_{L_i}$ is $s!$ and hence by Lemma 8 $e(\mathcal{P}^{\phi}_{L_i}) = s!\binom{2p-1}{p}$, which concludes the proof. ◀

▶ **Claim 11.** *Either $e(\mathcal{P}^{\phi}_{L_{r+1}}) \equiv 0 \mod p$, or $e(\mathcal{P}^{\phi}_{L_{r+1}}) = \frac{(|V(G)|p+p-1)!}{(p-1)}!(p!)^{|V(G)|}$ and L^{ϕ}_{r+1} contains exactly $|V(G)|$ elements of A_{r+1}, specifically one petal for each $(r,1)$-flower on level L_{r+1}.*

Proof of the claim. Assume $e(\mathcal{P}^{\phi}_{L_{r+1}}) \not\equiv 0 \mod p$, and let us first examine elements that are not associated with level L_{r+1}. Clearly, no element associated with level L_{r+2} can appear in $\mathcal{P}^{\phi}_{L_{r+1}}$ and the only elements associated with any level $i < r+1$ that can end up in $\mathcal{P}^{\phi}_{L_{r+1}}$ are petals. Each of these elements is smaller then exactly one petal at level L_{r+1} and independent to all other elements associated with this level. It is easy to see that largest possible size of a connected component of $\mathcal{P}^{\phi}_{L_{r+1}}$ is $p-1+2r < 2p$. By Lemma 9, every connected component in $\mathcal{P}^{\phi}_{L_{r+1}}$ (except for the stick) will have size p, and therefore $\mathcal{P}^{\phi}_{L_{r+1}}$ will contain exactly one element for every antichain associated with L_{r+1} and no other elements. Hence, $\mathcal{P}^{\phi}_{L_{r+1}}$ consists of $|V(G)|$ chains of length p and one chain of length $p-1$. Then $e(\mathcal{P}^{\phi}_{L_{r+1}}) = \frac{(|V(G)|p+p-1)!}{(p-1)}!(p!)^{|V(G)|}$ follows from Lemma 8. ◀

▶ **Claim 12.** *Either $e(\mathcal{P}^{\phi}_{L_{r+2}}) \equiv 0 \mod p$, or $e(\mathcal{P}^{\phi}_{L_{r+2}}) = \frac{(|E(G)|p+p-1)!}{(p-1)}!(p!)^{|E(G)|}$ and L^{ϕ}_{r+2} contains exactly $|E(G)|$ elements of A_{r+2}, specifically one petal for each $(r^2,1)$-flower on level L_{r+2}.*

Proof of the claim. The idea of the proof is similar to the proof of the previous claim, with one additional obstacle: that several flowers can be connected with petals from lower levels into one connected component on level L_{r+2} through the petals of flowers on level L_{r+1}. So, assume $e(\mathcal{P}^{\phi}_{L_{r+2}})$ contains a connected component C which contains at least a single stalk. For each stalk in C, there must be at least one petal in the same flower (otherwise the stalk cannot be connected to the rest of C); in other words, the intersection of each flower and C contains at least p vertices. Let a denote the number of flowers which intersect C, b_2 denote $|A_{r+2} \cap C|$, b_1 denote $|A_{r+1} \cap C|$ and b_0 denote $\sum_{i=1}^{r} |A_r \cap C|$. Then it follows that $|C| = p \cdot a + (b_2 - a) + b_1 + b_0 \leq p \cdot a + r^2|E(G)| + r|V(G)| + r|V(G)|$, and recall that $r^2|E(G)| + r|V(G)| + r|V(G)| < p$. Furthermore, if $b_1 > 0$ (and at least one petal from A_{r+1} is required unless C contains only a single flower), we have $a \cdot p < |C| < (a+1) \cdot p$. Hence any such C cannot have size divisible by p and by Lemma 9 we have $e(\mathcal{P}^{\phi}_{L_{r+2}}) \equiv 0 \mod p$. Otherwise, if no two flowers are connected through a petal of a flower associated with level L_{r+1}, then every connected component of $\mathcal{P}^{\phi}_{L_{r+2}}$ of size p must consist of a stalk and exactly one petal and the claim follows analogously as the proof of Claim 11. ◀

▶ **Claim 13.** *If ϕ is a consistent configuration and for all $i \in \{1, \ldots, r+2\}$ it holds that $e(\mathcal{P}^{\phi}_{L_i}) \not\equiv 0 \mod p$, then the petals in L^{ϕ}_{r+1} encode a proper equitable coloring of $V(G)$ where vertex v receives color i iff the petal v_i lies in L^{ϕ}_{i} and $\mathcal{P}^{\phi}_{L_{r+3}}$ is isomorphic with $\mathcal{P}(G, r)$.*

Proof of the claim. From Claims 10, 11 and 12 together with the assumption that $e(\mathcal{P}^{\phi}_{L_i}) \not\equiv 0$ mod p, it follows that each of the levels $L^{\phi}_1, \ldots, L^{\phi}_r$ contains exactly s petals associated with the corresponding level, level L^{ϕ}_{r+1} contains exactly one petal for each vertex of G and level L^{ϕ}_{r+2} contains exactly one petal for each edge of G.

For the first part of this claim, we observe that each pair of petals in $L_1^\phi, \ldots, L_r^\phi$ are associated with distinct vertices of G. If this were not the case, then since $|V(G)| = rs$ there would exist a vertex v such that no element of $L_1^\phi, \ldots, L_r^\phi$ is associated with v. But due to the construction at level $r+1$ there exists some $i \in 1, \ldots, r$ such that $v^i \in L_{r+1}^\phi$. Then, since $v_i \leq v^i$ and v_i can only occur either in level L_i^ϕ or L_{r+3}^ϕ (the latter of which lies above v^i in the linear extension due to the configuration ϕ), this would lead to a contradiction. In particular, we conclude that there is a matching between the petals in level $r+1$ (encoding the color for each vertex) and the union of petals in levels $1, 2, \ldots r$ (encoding the vertices assigned to each color class), and by Claim 10 it follows that there are exactly s petals in L_{r+1} associated with each color class.

We now argue that the coloring is proper. Observe that by the same argument as above, if an edge $e = uv$ satisfies $e_{i,j} \in L_{r+2}^\phi$, then $u_i \in L_{r+1}^\phi$ and $v_j \in L_{r+1}^\phi$. From the construction of $\mathcal{P}(G, r, p)$ it follows that if $i = j$, then $e_{i,j} \notin L_{r+2}$. Combining these two facts we get that the coloring encoded in L_{r+1}^ϕ is indeed proper.

Now let us take a look at level L_{r+3}^ϕ. To prove the claim, we will construct an isomorphism f from elements of $\mathcal{P}_{L_{r+3}}^\phi$ to elements of $\mathcal{P}(G, r)$. For every vertex $v \in V(G)$, precisely one element $v^i \in L_{r+1}^\phi$ and precisely one of the first r levels contains an element associated with v; to be precise, $v_i \in L_i^\phi$ and $v_j \in L_{r+3}^\phi$ and hence also $v^j \in L_{r+3}^\phi$ for all $j \neq i$. We set $f(v^j) = v_{j,0}$ and $f(v_j) = v_{j,1}$, whenever $j \neq i$ and $j < r$. For the last remaining elements, we set $f(v^r) = v_{i,0}$ and $f(v_r) = v_{i,1}$. Next, for every edge $e = uv$ there is exactly one $e_{a,b} \in L_{r+2}^\phi$. Moreover, if $e_{a,b} \in L_{r+2}^\phi$ then $u_a \in L_{r+1}^\phi$ and $v_b \in L_{r+1}^\phi$, and all other petals for this edge e are in L_{r+3}^ϕ. Let $g_i(r) = i$, $g_i(i) = 0$, and $g_i(k) = k$ otherwise. Then we set $f(e_{i,j}) = e_{g_a(i), g_b(j)}$. Observe that, since $e_{a,b}$ does not lie in L_{r+3}^ϕ, no edge is mapped to the non-existent element $e_{0,0}$ in $\mathcal{P}(G, r)$. It is straightforward to verify that f is really bijective mapping between elements of $\mathcal{P}_{L_{r+3}}^\phi$ and $\mathcal{P}(G, r)$. Moreover, $f(u) \leq f(v)$ in $\mathcal{P}(G, r)$ if and only if $u \leq v$ in $\mathcal{P}_{L_{r+3}}^\phi$. Therefore, $\mathcal{P}_{L_{r+3}}^\phi$ is isomorphic with $\mathcal{P}(G, r)$ and the claim holds. ◀

▶ **Claim 14.** *$e(\mathcal{P}(G, r, p)) \not\equiv 0 \mod p$ if and only if $e(\mathcal{P}(G, r)) \cdot \#\mathrm{EC}(G, r) \not\equiv 0 \mod p$.*

▶ **Claim 15.** *If $\#\mathrm{EC}(G, r) \neq 0$, then there is a prime number p greater than $2r|V(G)| + r^2|E(G)|$ and smaller than $(2r|V(G)| + r^2|E(G)|)^2$ such that p does not divide $e(\mathcal{P}(G, r)) \cdot \#\mathrm{EC}(G, r)$.*

▶ **Claim 16.** $\mathrm{tw}(C(\mathcal{P}(G, r, p))) \leq r \cdot (\mathrm{tw}(G) + 3) + 6$.

Concluding the proof

Let us summarize the fpt turing reduction used to prove Theorem 7. Given an instance (G, r) of Equitable Coloring[tw], we loop over all primes p such that $2r|V(G)| + r^2|E(G)| < p < (2r|V(G)| + r^2|E(G)|)^2$, and for each such prime we construct the poset $\mathcal{P}(G, r, p)$; from Claim 15 it follows that if $\#\mathrm{EC}(G, r) \neq 0$, then at least one such prime will not divide $e(\mathcal{P}(G, r)) \cdot \#\mathrm{EC}(G, r)$, and by Claim 16 each of the constructed posets $\mathcal{P}(G, r, p)$ has bounded treewidth of the cover graph. For each such poset $\mathcal{P}(G, r, p)$, we compute $e(\mathcal{P}(G, r, p))$ by the black-box procedure provided as part of the reduction. If for any prime p we get $e(\mathcal{P}(G, r, p)) \not\equiv 0 \mod p$, then we conclude that (G, r) is a yes-instance, and otherwise we reject (G, r), and this is correct by Claim 14. ◀

We remark that the above construction can be extended to also compute the exact number of equitable colorings. However, because Equitable Coloring[tw] is not known to be

#W[1]-hard, this does not immediately imply #W[1]-hardness for counting the number of linear extensions.

4 Fixed-Parameter Tractability of Counting Linear Extensions

This section is dedicated to proving our algorithmic result, stated below.

▶ **Theorem 17.** #LE *is fixed-parameter tractable parameterized by the treewidth of the incomparability graph of the input poset.*

The proof of Theorem 17 is divided into two steps. First, we apply a transformation process to a path-decomposition $\mathcal{Q}$ of small width (the existence of which is guaranteed by Corollary 6) of $I(\mathcal{P})$ which results in a tree-decomposition $\mathcal{T}$ of $I(\mathcal{P})$ satisfying certain special properties. We call these "blocked tree-decompositions" and the construction is given in Lemma 21. The properties of $\mathcal{T}$ are then used to prove that $I_C(\mathcal{P})$ has treewidth bounded by the treewidth of $I(\mathcal{P})$ (Corollary 26). In the second step, we construct an MSO formulation which enumerates all the linear extensions of $\mathcal{P}$ using $I_C(\mathcal{P})$, and apply Fact 4.

4.1 The Treewidth of Combined Graphs

We begin by arguing a useful property of separators in incomparability graphs.

▶ **Lemma 18.** *Let $S \subseteq V(I(\mathcal{P}))$. Then for each pair of distinct connected components C_1, C_2 in $I(\mathcal{P}) - S$, it holds that for any $a_1, b_1 \in C_1$ and any $a_2, b_2 \in C_2$ we have $a_1 \leq a_2$ iff $b_1 \leq b_2$. Namely, the poset contains a total order of all connected components in $I(\mathcal{P}) - S$.*

Proof. We begin by proving the following claim.

▶ **Claim 19.** *Let a, b, c be three distinct elements of $\mathcal{P}$ such that $a \parallel b$ and both pairs a, c and b, c are comparable. Then $a \leq c$ iff $b \leq c$.*

Proof of the claim. Suppose that, w.l.o.g., $a \leq c$ and $c \leq b$. Then by the transitivity of $\leq$, we get $a \leq b$ which contradicts our assumption that $a \parallel b$. ◀

Now to prove Lemma 18, assume for a contradiction that, w.l.o.g., there exist $a_1, b_1 \in C_1$ and $a_2, b_2 \in C_2$ such that $a_1 \leq b_1$ and $b_2 \leq a_2$. Let Q_1 be an a_1-a_2 path in $I[C_1]$. By Claim 19, $a_1 \leq b_1$ implies that every element q on Q_1 satisfies $q \leq b_1$, and in particular $a_2 \leq b_1$. Next, let Q_2 be a b_1-b_2 path in $I[C_2]$. Then Claim 19 also implies that each element q' on Q_2 satisfies $a_2 \leq q'$. Since b_2 lies on Q_2, this would imply that $a_2 \leq b_2$, a contradiction. ◀

To proceed further, we will need some notation. Let $\mathcal{T} = (T, \mathcal{X})$ be a rooted tree-decomposition and $t \in V(T)$. We denote by $L(t)$ the set of all vertices which occur in the "branch" of $T - t$ containing the root r; formally, $L(t) = \{v \in X_{t'} \setminus X_t \mid t'$ lies in the same connected component as r in $T - t\}$. We then set $R(t) = V(G) \setminus (L(t) \cup X_t)$ (the intuition behind L and R is that they represent "left" and "right"). We also let T_t^r denote the connected component of $T - t$ which contains the root r.

Next, recall that each connected component of the graph obtained after deleting X_t must lie in a subtree of $T - t$ (Fact 3). A *block* of a bag X_t in a rooted tree-decomposition $\mathcal{T} = (T, \mathcal{X})$ is a sequence of consecutive connected components in $(I(\mathcal{P}) - X_t) \cap R(t)$. We say that a node $t \in V(T)$ has z blocks if there exist z distinct blocks of X_t. Blocks will play an important role in the tree-decomposition we wish to obtain from our initial path-decomposition of $I(\mathcal{P})$. The following lemma captures the operation we will use to alter our path-decomposition.

▶ **Lemma 20.** *Let $\mathcal{T} = (T, \mathcal{X})$ be a rooted tree-decomposition of a graph G and let $t \in V(T)$ be such that there are z blocks of X_t. Then there is a tree-decomposition $\mathcal{T}'(T', \mathcal{X}')$ satisfying:*

1. *The width of $\mathcal{T}'$ is at most the width of $\mathcal{T}$.*
2. *The tree T' contains T_t^r as a subtree which is separated from the rest of T' by t.*
3. *The degree of t in T' is $z + 1$.*
4. *There exists a bijection α between the z blocks of X_t and the z trees in $T' - t$ other than T_t^r such that for each block B of X_t, we have $\bigcup_{s \in \alpha(B)} X'_s \setminus X_t = B$.*
5. *For each $t' \in N[t] \setminus V(T_t^r)$, we have $X_{t'} = X_t$.*

We proceed by showing how Lemma 20 is applied to transform a given path-decomposition.

▶ **Lemma 21.** *Let $\mathcal{Q}$ be a nice path-decomposition of $I(\mathcal{P})$. Then there is a rooted tree-decomposition $\mathcal{T} = (T, \mathcal{X})$ of $I(\mathcal{P})$ with the following properties. $\mathcal{T}$ is rooted at a leaf r and $X_r = \emptyset$, the width of $\mathcal{T}$ is at most the width of $\mathcal{Q}$, and for any node $t \in V(T)$ with $z > 1$ blocks:*

1. *The degree of t in T is $z + 1$.*
2. *There exists a bijection α between the z blocks of X_t and the z trees in $T' - t$ other than T_t^r such that for each block B of X_t, we have $\bigcup_{s \in \alpha(B)} X_s \setminus X_t = B$.*
3. *For $t' \in N(t) \cap V(T_t^r)$ there exists a vertex v such that $X_{t'} = X_t \setminus \{v\}$, and furthermore t' has degree 2 and 1 block.*
4. *For each pair of neighbors $t, t' \in V(T)$, it holds that $|X_t \setminus X_{t'}| + |X_{t'} \setminus X_t| \leq 1$.*

We call a tree-decomposition rooted at a leaf with $X_r = \emptyset$ which satisfies the properties of Lemma 21 a *blocked tree-decomposition*. The next ingredient we will need for proving that $I_C(\mathcal{P})$ has small treewidth is the notion of *cover-guards*.

Let $\mathcal{T} = (T, \mathcal{X})$ be a tree-decomposition of $I(\mathcal{P})$ rooted at r and let $t \neq r$. Then the *cover-guard* of t, denoted $\mathcal{A}_t$, is the set of vertices in $L(t)$ which are incident to a cover edge whose other endpoint lies in $R(t)$; formally, $\mathcal{A}_t = \{v \in L(t) \mid \exists u \in R(t) : (uv \in E(C(\mathcal{P})) \vee vu \in E(C(\mathcal{P})))\}$. For a vertex $v \in I(\mathcal{P})$, we let $\mathcal{A}^v = \{t \in V(T) \mid v \in \mathcal{A}_t\}$ and $X^v = \{t \in V(T) \mid v \in X_t\}$.

Our next aim is to add all the cover-guards into each bag. The following lemma will allow us to argue that the result is still a tree-decomposition; it is worth noting that the assumption that the decomposition is blocked is essential for the lemma to hold.

▶ **Lemma 22.** *Let $\mathcal{T} = (T, \mathcal{X})$ be a blocked tree-decomposition of $I(\mathcal{P})$ rooted at r and let $v \in I(\mathcal{P})$. Then $T[\mathcal{A}^v \cup X^v]$ is a tree.*

Next we show that the cover-guards in blocked tree-decompositions are never too large.

▶ **Lemma 23.** *Let $\mathcal{T} = (T, \mathcal{X})$ be a blocked tree-decomposition of $I(\mathcal{P})$ of width k. Then for each $t \in V(T)$ it holds that $|\mathcal{A}_t| \leq 2k + 2$.*

Proof. First, observe that if a node $t \in V(T)$ has 0 blocks, then $R(t) = \mathcal{A}_t = \emptyset$. So, consider a node t which has exactly 1 block consisting of connected components $(D_1, \ldots, D_j)$ in $(I(\mathcal{P}) - X_t) \cap R(t)$.

▶ **Claim 24.** $|\mathcal{A}_t| \leq 2k + 2$.

Proof of the claim. Assume for a contradiction that $|\mathcal{A}_t| > 2k+2$. By Lemma 18 we have that $(D_1, \dots, D_j)$ are consecutive connected components in a total order of connected components in $I(\mathcal{P}) - X_t$. Hence any edge in $C(\mathcal{P}) - X_t$ between $R(t)$ and $L(t)$ must necessarily have one endpoint in $D_1 \cup D_j$. Furthermore, an element in $\mathcal{A}_t$ cannot be adjacent to both D_1 and D_j in $C(\mathcal{P}) - X_t$ due to transitivity and acyclicity. So, we may partition $\mathcal{A}_t$ into $\mathcal{A}_t^1 = \{v \in \mathcal{A}_t \mid \exists u \in D_1 : v \lhd^{\mathcal{P}} u\}$ and $\mathcal{A}_t^2 = \{v \in \mathcal{A}_t \mid \exists u \in D_j : u \lhd^{\mathcal{P}} v\}$.

By Lemma 18, it also follows that $\mathcal{A}_t^1$ and $\mathcal{A}_t^2$ must each lie in separate connected components of $I(\mathcal{P}) - X_t$, say C^1 and C^2 respectively. Furthermore, each element in $\mathcal{A}_t^1$ is maximal in C^1 and each element in $\mathcal{A}_t^2$ is minimal in C^2. In particular, each of $\mathcal{A}_t^1$, $\mathcal{A}_t^2$ forms a clique in $I(\mathcal{P})$. But by our assumption on the size of $\mathcal{A}_t$, at least one of $\mathcal{A}_t^2$ and $\mathcal{A}_t^1$ must have size greater than $k+1$, which implies that $I(\mathcal{P})$ contains a clique of size at least $k+2$. It is well-known that each clique must be completely contained in at least one bag of a tree-decomposition, and so we arrive at a contradiction with $\mathrm{tw}(I(\mathcal{P})) \le k$. Hence we conclude that $|\mathcal{A}_t| \le 2k+2$ and the claim holds. ◀

Finally, consider a node t which has at least 2 blocks. By Property **3** of Lemma 21, it holds that t has a neighbor t' in T_t^r such that $X_{t'} = X_t \setminus \{v\}$ and t' has 1 block. By Claim 24 we know that $\mathcal{A}_{t'} \le 2k+2$. Since $L(t) = L(t')$ and $R(t) \subseteq R(t')$, it follows that $\mathcal{A}_t \subseteq \mathcal{A}_{t'}$, and in particular $|\mathcal{A}_t| \le |\mathcal{A}_{t'}|$. We have now proved the desired bound for all nodes in $\mathcal{T}$, and so the lemma holds. ◀

With Lemma 22 and Lemma 23, we have the tools necessary for arguing that there exists a tree-decomposition of the combined graph of small width.

▶ **Lemma 25.** *Let $\mathcal{T} = (T, \mathcal{X})$ be a blocked tree-decomposition of $I(\mathcal{P})$ such that $\mathrm{tw}(\mathcal{T}) \le k$. Then there exists a tree-decomposition $\mathcal{T}'$ of $I_C(\mathcal{P})$ of width at most $3k+2$.*

Proof. Consider the tree-decomposition $\mathcal{T}' = (T, \mathcal{X}')$ where $\mathcal{X}' = \{X'_t \mid t \in V(T)\}$ is defined as follows. For each $t \in V(T)$ such that its unique neighbor s in T_t^r satisfies $|X_t \setminus X_s| = 1$, we set $X'_t = X_t \cup \mathcal{A}_s$; it will be useful to observe that $\mathcal{A}_s \supseteq \mathcal{A}_t$. For all other nodes $t \in V(T)$, we then set $X'_t = X_t \cup \mathcal{A}_t$. We call nodes of the first type *non-standard* and nodes of the second type standard.

First, we note that the size of each bag in $\mathcal{T}'$ is at most $3k+2$, since every node $t \in V(T)$ satisfies $|\mathcal{A}_t| \le 2k+2$ by Lemma 23. Furthermore, $\mathcal{T}'$ satisfies condition (T1) because $\mathcal{T}$ was a tree-decomposition of $I(\mathcal{P})$. $\mathcal{T}'$ also satisfies condition (T2); indeed, for each $v \in \mathcal{P}$ it holds that X'^v restricted to standard nodes is a connected tree by Lemma 22, and by construction every non-standard node t such that $v \in X'_t \setminus X_t$ is adjacent to a standard node containing v. So, it only remains to argue condition (T3).

Obviously, condition (T3) holds for any edge of $I(\mathcal{P})$. So, consider two elements u, v of $\mathcal{P}$ such that $u \lhd^{\mathcal{P}} v$ or $v \lhd^{\mathcal{P}} u$. If there exists a node $t \in V(T)$ such that $u, v \in X_t$, then $u, v \in X'_t$ and the condition also holds for this edge in $I_C(\mathcal{P})$. So, assume that X^v and X^u are disjoint and let Q be the unique X^v-X^u path in T. By Property **4**, the X^v-X^u path Q in T must contain at least one internal node.

Consider the case where one of these subtrees, say w.l.o.g. X^v, lies in the connected component T_t^r of $T - Q$. Then for each internal node $q \in Q$, it holds that $v \in L(q)$ and $u \in R(q)$, which in turn implies that $v \in \mathcal{A}_q$. Let q_u be the endpoint of Q in X^u and let q_0 be the neighbor of q_u in Q. By Property **4** we have $X_{q_u} \setminus X_{q_0} = \{u\}$, which implies that q_u is a non-standard node and in particular $\mathcal{A}_{q_0} \subseteq X'_{q_u}$. Since q_0 is an internal node of Q, it follows that $v \in X'_{q_u}$ which means that condition (T3) also holds for any edge uv in this case.

Finally, consider the case where there exists a node $q \in Q$ of degree at least 3 such that each of X^u, X^v and r occur in different components of $T - q$. Then we reach a contradiction similarly as in the proof of Lemma 22. In particular, since $u, v \in R(q)$ due to the location of the root and there is a cover edge between them, it follows that either u, v occur in the same connected component of X_q or in two consecutive ones, but in either case u, v must lie in the same block of q, say block B. But since $u, v \notin X_q$, this contradicts Property **2** in Lemma 21; indeed, each tree in $T - q$ contains at most one of v, u in its bags, and hence there exists no tree T' in $T - q$ satisfying $\bigcup_{t' \in V(T')} X_{t'} \setminus X_q = B$. Hence this case in fact violates our assumptions and cannot occur.

Summarizing the above arguments, we conclude that each bag in $\mathcal{T}'$ has size at most $3k + 2$ and that $\mathcal{T}'$ satisfies all of the conditions of a tree-decomposition. ◀

▶ **Corollary 26.** *Let $\mathcal{P}$ be a poset such that $\text{tw}(I(\mathcal{P})) \leq k$. Then $\text{tw}(I_C(\mathcal{P})) \leq 3k + 2$.*

Proof. By Corollary 6 we know that there exists a nice path-decomposition of $I(\mathcal{P})$ of width at most k. By Lemma 21, it follows that there exists a blocked tree-decomposition of $I(\mathcal{P})$ of width at most k. The corollary then follows by Lemma 25. ◀

4.2 MSO Formulation

In this subsection, we use Fact 4 to prove the following result, which forms the second ingredient required for our proof of Theorem 17.

▶ **Lemma 27.** *#LE is fixed-parameter tractable parameterized by the treewidth of the combined graph of the input poset.*

Sketch of the Proof. Let $\mathcal{P} := (P, \leq^P)$ be a poset. Let G be the (edge-)labeled directed graph obtained from $I_C(\mathcal{P})$ by directing every bidirectional edge of $I_C(\mathcal{P})$, i.e., every edge of $I(\mathcal{P})$, in an arbitrary way and labeling it with the label $\|$.

For a set of edges $E \subseteq E(G)$ with label $\|$, let $G[E]$ be the graph obtained from G after reversing every edge in E. Moreover, for a linear extension $\preceq$ of $\mathcal{P}$ let $E_G(\preceq)$ be the set of edges (u, v) of G such that $v \preceq u$. Note that because every linear extension of $\mathcal{P}$ has to respect the direction of the edges in G given by C, it holds that every edge in $E_G(\preceq)$ has label $\|$.

▶ **Claim 28.** *$E_G(\preceq)$ defines a bijection between the set of linear extensions of $\mathcal{P}$ and the set of subsets E of edges of G with label $\|$ such that $G[E]$ is acyclic.*

Proof of the claim. Let $\preceq$ be a linear extension of $\mathcal{P}$. Then, as observed above, $E_G(\preceq)$ is a set of edges of G with label $\|$. Moreover, because $G[E_G(\preceq)]$ is a subgraph of $P_G(\preceq)$ and $P_G(\preceq)$ is acyclic so is $G[E_G(\preceq)]$. Hence, $E_G(\preceq)$ is a function from the set of linear extensions of $\mathcal{P}$ to the set of subsets E of edges of G with label $\|$ such that $G[E]$ is acyclic. Towards showing that $E_G(\preceq)$ is injection assume for a contradiction that this is not the case, i.e., there are two distinct linear extensions $\preceq_1$ and $\preceq_2$ of $\mathcal{P}$ such that $E_G(\preceq_1) = E_G(\preceq_2)$ and let u and v be two elements of $\mathcal{P}$ ordered differently by $\preceq_1$ and $\preceq_2$. Then $\{u, v\} \in I(\mathcal{P})$ and hence either $(u, v) \in G$ or $(v, u) \in G$ the label of (u, v) or (v, u) respectively is $\|$. W.l.o.g. assume that $(u, v) \in G$ with label $\|$. But then, because $\preceq_1$ and $\preceq_2$ differ on u and v, either $(u, v) \in E_G(\preceq_1)$ but not $(u, v) \in E_G(\preceq_2)$ or $(u, v) \in E_G(\preceq_2)$ but not $(u, v) \in E_G(\preceq_1)$. In both cases we get a contradiction to our assumption that $E_G(\preceq_1) = E_G(\preceq_2)$. It remains to show that $E_G(\preceq)$ is surjective. To see this let E be a subsets of the edges of G with label $\|$ such that $G[E]$ is acyclic. Because $G[E]$ is acyclic it has a topological ordering, say $\preceq$, of its vertices. Because $G[E]$ contains $C(\mathcal{P})$ as a subgraph and any topological ordering of $C(\mathcal{P})$ is a linear extension of $\mathcal{P}$, we obtain that $\preceq$ is a linear extension and also $E = E_G(\preceq)$. ◀

It follows from the above that instead of counting the number of linear extensions of $\mathcal{P}$ directly, we can count the number of subsets E of the edges of G with label $\|$ such that $G[E]$ is acyclic. It can be shown that there exists an MSO formula Φ (with length independent of G) such that $G \models \Phi(X)$ if and only if X is a subset of the edges of G with label $\|$ such that $G[X]$ is acyclic. Because of Fact 4, this implies that #LE is fixed-parameter tractable when parameterized by $\mathrm{tw}(G)$ and hence also when parameterized by $\mathrm{tw}(I_C(\mathcal{P}))$, concluding the proof of the lemma. Generally speaking, $\Phi(X)$ only needs to check that X is a set of edges of G with label $\|$ and there is no non-empty set of edges C of $G[X]$ that forms a cycle. ◀

We conclude this section by stating the proof of Theorem 17.

Proof of Theorem 17. Let $\mathcal{P}$ be the input poset and let $k = \mathrm{tw}(I(\mathcal{P}))$. Then $\mathrm{tw}(I_C(\mathcal{P})) \leq 3k + 2$ by Corollary 26, and the theorem follows by Lemma 27. ◀

5 Conclusions and Future Work

We have given the first parameterized intractability result for counting linear extensions. We hope that the employed techniques will inspire similar results and expand our knowledge about the parameterized complexity of counting problems. In particular, even for #LE there remain many open questions concerning other very natural parameterizations such as the width of the poset or the treewidth of the poset graph. Moreover, our intractability result for the treewidth of the cover graph poses the question whether there are stronger parameterizations under which #LE becomes tractable, e.g., the treewidth of the poset graph, the treedepth or even vertex cover number of the poset- or cover graph, as well as combinations of these parameters with parameters such as the width, the dimension, or the height of the poset. These numerous examples illustrate that the parameterized complexity of #LE is still largely unexplored. As a side note it would also be interesting to establish whether our hardness result for #LE can be sharpened to #W[1]-hardness and to obtain matching membership results.

References

1 Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2):308–340, 1991.

2 Mike D Atkinson. On computing the number of linear extensions of a tree. *Order*, 7(1):23–25, 1990.

3 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

4 Hans L. Bodlaender and Ton Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *J. Algorithms*, 21(2):358–402, 1996.

5 Bostjan Bresar, Manoj Changat, Sandi Klavzar, Matjaz Kovse, Joseph Mathews, and Antony Mathews. Cover-incomparability graphs of posets. *Order*, 25(4):335–347, 2008.

6 Graham Brightwell and Peter Winkler. Counting linear extensions is #P-complete. In *Proceedings of the Twenty-third Annual ACM Symposium on Theory of Computing*, STOC'91, pages 175–181, 1991.

7 Russ Bubley and Martin Dyer. Faster random generation of linear extensions. *Discrete Mathematics*, 201(1-–3):81–88, 1999.

8 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990.

9 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

10 Karel De Loof, Hans De Meyer, and Bernard De Baets. Exploiting the lattice of ideals representation of a poset. *Fundam. Inf.*, 71(2,3):309–321, February 2006. URL: `http://dl.acm.org/citation.cfm?id=1227505.1227514`.

11 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

12 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

13 Martin Dyer, Alan Frieze, and Ravi Kannan. A random polynomial-time algorithm for approximating the volume of convex bodies. *J. ACM*, 38(1):1–17, January 1991. `doi:10.1145/102782.102783`.

14 Michael R. Fellows, Fedor V. Fomin, Daniel Lokshtanov, Frances A. Rosamond, Saket Saurabh, Stefan Szeider, and Carsten Thomassen. On the complexity of some colorful problems parameterized by treewidth. *Inf. Comput.*, 209(2):143–153, 2011.

15 Stefan Felsner and Thibault Manneville. Linear extensions of N-free orders. *Order*, 32(2):147–155, 2014. `doi:10.1007/s11083-014-9321-0`.

16 Jörg Flum and Martin Grohe. The parameterized complexity of counting problems. *SIAM J. Comput.*, 33(4):892–922, 2004.

17 Georg Gottlob, Reinhard Pichler, and Fang Wei. Bounded treewidth as a key to tractability of knowledge representation and reasoning. *Artificial Intelligence*, 174(1):105–132, 2010. `doi:10.1016/j.artint.2009.10.003`.

18 Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *Journal of the ACM*, 54(1), 2007.

19 M. Habib and R.H. Möhring. On some complexity properties of N-free posets and posets with bounded decomposition diameter. *Discrete Mathematics*, 63(2):157–182, 1987. `doi:10.1016/0012-365X(87)90006-9`.

20 Michel Habib and Rolf H Möhring. Treewidth of cocomparability graphs and a new order-theoretic parameter. *Order*, 11(1):47–60, 1994.

21 Thore Husfeldt, Ramamohan Paturi, Gregory B. Sorkin, and Ryan Williams. Exponential Algorithms: Algorithms and Complexity Beyond Polynomial Time (Dagstuhl Seminar 13331). *Dagstuhl Reports*, 3(8):40–72, 2013. `doi:10.4230/DagRep.3.8.40`.

22 Kustaa Kangas, Teemu Hankala, Teppo Niinimäki, and Mikko Koivisto. Counting linear extensions of sparse posets. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence, IJCAI 2016, New York City, USA*, 2016. to appear.

23 T. Kloks. *Treewidth: Computations and Approximations*. Springer Verlag, Berlin, 1994.

24 Thomas Lukasiewicz, Maria Vanina Martinez, and Gerardo I. Simari. Probabilistic preference logic networks. In Torsten Schaub, Gerhard Friedrich, and Barry O'Sullivan, editors, *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 561–566. IOS Press, 2014.

25 Heikki Mannila and Christopher Meek. Global partial orders from sequential data. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD'00, pages 161–168, New York, NY, USA, 2000. ACM. `doi:10.1145/347090.347122`.

26 Rolf H. Möhring. *Algorithms and Order*, chapter Computationally Tractable Classes of Ordered Sets, pages 105–193. Springer Netherlands, 1989.

27 Jason Morton, Lior Pachter, Anne Shiu, Bernd Sturmfels, and Oliver Wienand. Convex rank tests and semigraphoids. *SIAM Journal on Discrete Mathematics*, 23(3):1117–1134, 2009. doi:10.1137/080715822.

28 Teppo Mikael Niinimäki and Mikko Koivisto. Annealed importance sampling for structure learning in Bayesian networks. In *IJCAI*. IJCAI/AAAI, 2013.

29 Daniel Paulusma, Friedrich Slivovsky, and Stefan Szeider. Model counting for CNF formulas of bounded modular treewidth. *Algorithmica*, pages 1–27, 2015. doi:10.1007/s00453-015-0030-x.

30 Marcin Peczarski. New results in minimum-comparison sorting. *Algorithmica*, 40(2):133–145, July 2004. doi:10.1007/s00453-004-1100-7.

A Constant Approximation Algorithm for Scheduling Packets on Line Networks*

Guy Even[1], Moti Medina[2], and Adi Rosén[3]

1 **Tel Aviv University, Tel Aviv, Israel**
guy@eng.tau.ac.il
2 **MPI for Informatics, Saarbrücken, Germany**
mmedina@mpi-inf.mpg.de
3 **CNRS and Université Paris Diderot, Paris, France**
adiro@liafa.univ-paris-diderot.fr

Abstract

In this paper we improve the approximation ratio for the problem of scheduling packets on line networks with bounded buffers with the aim of maximizing the throughput. Each node in the network has a local buffer of bounded size B, and each edge (or link) can transmit a limited number c of packets in every time unit. The input to the problem consists of a set of packet requests, each defined by a source node, a destination node, and a release time. We denote by n the size of the network. A solution for this problem is a schedule that delivers (some of the) packets to their destinations without violating the capacity constraints of the network (buffers or edges). Our goal is to design an efficient algorithm that computes a schedule that maximizes the number of packets that arrive to their respective destinations.

We give a randomized approximation algorithm with constant approximation ratio for the case where the buffer-size to link-capacity ratio, B/c, does not depend on the input size. This improves over the previously best result of $O(\log^* n)$ [11]. Our improvement is based on a new combinatorial lemma that we prove, stating, roughly speaking, that if packets are allowed to stay put in buffers only a limited number of time steps, $2d$, where d is the longest source-destination distance, then the optimal solution is decreased by only a constant factor. This claim was not previously known in the integral (unsplitable, zero-one) case, and may find additional applications for routing and scheduling algorithms.

While we are not able to give the same improvement for the related problem when packets have hard deadlines, our algorithm does support "soft deadlines". That is, if packets have deadlines, we achieve a constant approximation ratio when the produced solution is allowed to miss deadlines by at most $\log n$ time units.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases approximation algorithms, linear programming, randomized rounding, packet scheduling, admission control

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.40

1 Introduction

In this paper we give an approximation algorithm with an improved approximation ratio for a network-scheduling problem which has been studied in numerous previous works in a number of variants (cf. [2, 3, 5, 8, 14, 11]). The problem consists of a directed line network

* The full version of this paper can be found in http://arxiv.org/abs/1602.06174.

over nodes $\{0, \ldots, n-1\}$, where each node i can send packets to node $i+1$, and can also store packets in a local buffer. The maximum number of packets that can be sent in a single time unit over a given link is denoted by c, and the number of packets each node can store at any given time is denoted by B. An instance of the problem is further defined by a set of packets $r_i = (a_i, b_i, t_i)$, $1 \leq i \leq M$, where a_i is the source node of the packet, b_i is its destination node, and $t_i \geq 1$ is the release time of the packet at vertex a_i. The goal is that of maximizing the number of packets that reach their respective destinations without violating the links or the buffers capacities. See Section 2 for a formal definition of the problem.

We present a randomized approximation algorithm for that problem, which has a *constant* approximation ratio for the case that the ratio B/c does not depend on the input size, improving upon the previous $O(\log^* n)$ approximation ratio given in [11, Theorem 3]. While this constant approximation result does not hold for the variant of the problem where packets have deadlines, our algorithm does provide a constant-approximation solution that abides to "soft deadlines". That is, in that solution each packet is delivered at most $\log n$ time units past its deadline.

Our algorithm is based on a novel combinatorial lemma 3 which states the following. Consider a set of packets such that all source-destination distances are bounded from above by some d. The throughput of an optimal solution in which every packet r_i must reach its destination no later than time $t_i + 2d$ is an $\Omega(B/c)$-fraction of the unrestricted optimal throughput. This lemma plays a crucial role in our algorithm, and we believe that it may find additional applications for scheduling and routing algorithms in networks. We emphasize that the fractional version of a similar property, i.e., when packets are splitable and one accrues a benefit also from the delivery of partial packets, presented first in [5], does not imply the integral version that we prove here.

We emphasize that the problem studied here, namely, maximizing the throughput on a network with bounded buffers, has resisted substantial efforts in its (more applicable) distributed, online setting, even for the simple network of a directed line. Indeed, even the question whether or not there exists a constant competitive online distributed algorithm for that problem on the line network remains unanswered at this point. We therefore study here the offline setting with the hope that, in addition to its own interest, results and ideas from this setting will contribute to progress on the distributed problem.

Related Work. The problem of scheduling packets so as to maximize the throughput (i.e., maximize the number of packets that reach their destinations) in a network with bounded buffers was first considered in [2], where this problem is studied for various types of networks in the distributed setting. The results in that paper, even for the simple network of a directed line, were far from tight but no substantial progress has been made since on the realistic, distributed and online, setting. This has motivated the study of this problem in easier settings, as a first step towards solving the realistic, possibly applicable, scenario.

Angelov et al. [3] give centralized online randomized algorithms for the line network, achieving an $O(\log^3 n)$-competitive ratio. Azar and Zachut [5] improved the randomized competitive ratio to $O(\log^2 n)$ which was later improved by Even and Medina [6, 8] to $O(\log n)$. A deterministic $O(\log^5 n)$-competitive algorithm was given in [7, 8], which was later improved in [9] to $O(\log n)$ if buffer and link capacities are not very small (not smaller than 5).

The related problem of maximizing the throughput when packets have deadlines (i.e., a packet is counted towards the quality of the solution only if it arrives to its destination before a known deadline) on line network with unbounded input queues is known to be NP-hard [1].

The same problem in a variant of the setting, where the input queues are bounded, is shown in [11] to have a $O(\log^* n)$-approximation randomized algorithm. The setting in the present paper is the same setting as the one of the latter paper, and the results of [11] immediately give an $O(\log^* n)$-approximation randomized algorithm for the problem and setting we study in the present paper.

2 Preliminaries

2.1 Model and problem statement

We consider the standard model of synchronous store-and-forward packet routing networks [2, 3, 5]. The network is modeled by a directed path over n vertices. Namely, the network is a directed graph $G = (V, E)$, where $V = \{0, \ldots, (n-1)\}$ and there is a directed edge from vertex u to vertex v if $v = u + 1$. The network resources are specified by two positive integer parameters B and c that describe, respectively, the local buffer capacity of every vertex and the capacity of every edge. In every time step, at most B packets can be stored in the local buffer of each vertex, and at most c packets can be transmitted along each edge.

The input consists of a set of packet requests $R = \{r_i\}_{i=1}^{M}$. A packet request is specified by a 3-tuple $r_i = (a_i, b_i, t_i)$, where $a_i \in V$ is the *source node* of the packet, $b_i \in V$ is its *destination node*, and $t_i \in \mathbb{N}$ is the release time of the packet at vertex a_i. Note that $b_i > a_i$, and r_i is ready to leave a_i in time step t_i.

A solution is a schedule S. For each request r_i, the schedule S specifies a sequence s_i of transitions that packet r_i undergoes. A *rejected* request r_i is simply discarded at time t_i, and no further treatment is required (i.e., $s_i = \{\mathit{reject}\}$). An *accepted* request r_i is delivered from a_i to b_i by a sequence s_i of actions, where each action is either "store" or "forward". Consider the packet of request r_i. Suppose that in time t the packet is in vertex v. A store action means that the packet is stored in the buffer of v, and will still be in vertex v in time step $t + 1$. A forward action means that the packet is transmitted to vertex $v + 1$, and will be in vertex $v + 1$ in time step $t + 1$. The packet of request r_i reaches its destination b_i after exactly $b_i - a_i$ forward steps. Once a packet reaches its destination, it is removed from the network and it no longer consumes any of the network's resources.

A schedule must satisfy the following constraints:

1. The *buffer capacity constraint* asserts that at any time step t, and in every vertex v, at most B packets are stored in v's buffer.
2. The *link capacity constraint* asserts that at any step t, at most c packets can be transmitted along each edge.

The *throughput* of a schedule S is the number of accepted requests. We denote the throughput of a schedule S by $|S|$. As opposed to online algorithms, there is no point in injecting a packet to the network unless it reaches its destination. Namely, a packet that is not rejected and does not reach its destination only consumes network resources without any benefit. Hence, without loss of generality, we assume that every packet that is dropped before reaching its designation is rejected at its source node at its release time.

We consider the offline optimization problem of finding a schedule that maximizes the throughput. We propose a centralized constant-ratio approximation algorithm. By *offline* we mean that the algorithm receives all requests in advance[1]. By *centralized* we mean that all

[1] The number of requests M is finite and known in the offline setting. This is not the case in the online setting in which the number of requests is not known in advance and may be unbounded.

the requests are known in one location where the algorithm is executed. Let $\mathsf{opt}(R)$ denote a schedule of maximum throughput for the set of requests R. Let $\mathsf{alg}(R)$ denote the schedule computed by alg on input R. We say that the approximation ratio of a scheduling algorithm alg is c if $\forall R : |\mathsf{alg}(R)| \geq c \cdot |\mathsf{opt}(R)|$. For a randomized algorithm we say that the expected approximation ratio is c if $\forall R : \mathbf{E}[|\mathsf{alg}(R)|] \geq c \cdot |\mathsf{opt}(R)|$.

The Max-Pkt-Line Problem. The problem of maximum throughput scheduling of packet requests on directed line (Max-Pkt-Line) is defined as follows. The input consists of: n - the size of the network, B - node buffer capacities, c - link capacities, and M packet requests $\{r_i\}_{i=1}^M$. The output is a schedule S. The goal is to maximize the throughput of S.

2.2 Path Packing in a uni-directed 2D-Grid

In this section we define a problem of maximum cardinality path packing in a two-dimensional uni-directed grid (Max-Path-Grid). This problem is equivalent to Max-Pkt-Line, and was used for that purpose in previous work, where the formal reduction is also presented [4, 1, 5, 11]. As the two problems are equivalent, we use in the sequel terminology from both problems interchangeably.

The grid, denoted by $G^{st} = (V^{st}, E^{st})$, is an infinite directed acyclic graph. The vertex set V^{st} equals $V \times \mathbb{N}$, where $V = \{0, 1, \ldots, (n-1)\}$. Note that we use the first coordinate (that corresponds to vertices in V) for the y-axis and the second coordinate (that corresponds to time steps) for the x-axis. The edge set consists of horizontal edges (also called store edges) directed to the right and vertical edge (also called forward edges) directed upwards. The capacity of vertical edges is c and the capacity of horizontal edges is B. We often refer to G^{st} as the space-time grid (in short, grid) because the x-axis is related to time and the y-axis corresponds to the vertices in V.

A *path request* in the grid is a tuple $r^{st} = (a_i, t_i, b_i)$, where $a_i, b_i \in V$ and $t_i \in \mathbb{N}$. The request is for a path that starts in node (a_i, t_i) and ends in any node in the row of b_i (i.e., the end of the path can be any node (b_i, t), where $t \geq t_i$).

A *packing* is a set of paths S^{st} that abides the capacity constraints. For every grid edge e, the number of paths in S^{st} that contain e is not greater than the capacity of e.

Given a set of path requests $R^{st} = \{r_i^{st}\}_{i=1}^M$, the goal in the Max-Path-Grid problem is to find a packing S^{st} with the largest cardinality. (Each path in S^{st} serves a distinct path request.)

Multi-Commodity Flows (MCFs). Our use of path packing problems gives rise to *fractional* relaxations of that problem, namely to multi-commodity flows (MCFs) with unit demands on uni-directional grids. We deferred the definitions and terminology of MCFs to the full version.

2.3 Tiling, Classification, and Sketch Graphs

To define our algorithm we make use of partitions of the space-time grid described above into sub-grids. We define here the notions we use for this purpose. In this section we focus on the case of unit capacities, namely, $B = c = 1$. An extension to other values of B and c can be found [8].

Tiling. *Tiling* is a partitioning of the two-dimensional space-time grid (in short, grid) into squares, called *tiles*. Two parameters specify a tiling: the side length k, an even integer, of the squares and the shifting (φ_x, φ_y) of the squares. The shifting refers to the x- and y-coordinates of the bottom left corner of the tiles modulo k. Thus, the tile $T_{i,j}$ is the subset of the grid vertices defined by

$$T_{i,j} \triangleq \{(v,t) \in V \times \mathbb{N} \mid ik \leq v - \varphi_y < (i+1)k \text{ and } jk \leq t - \varphi_x < (j+1)k\},$$

where φ_x and φ_y denote the horizontal and vertical shifting, respectively. We consider two possible shifts for each axis, namely, $\varphi_x, \varphi_y \in \{0, k/2\}$.

Quadrants and Classification. Consider a tile T. Let (x', y') denote the lower left corner (i.e., south-west corner) of T. The *south-west quadrant* of T is the set of vertices (x,y) such that $x' \leq x \leq x' + k/2$ and $y' \leq y \leq y' + k/2$.

For every vertex (x,y) in the grid, there exists exactly one shifting $(\varphi_x, \varphi_y) \in \{0, k/2\}^2$ such that (x,y) falls in the south-west (SW) quadrant of a tile. Fix the tile side length k. We define a *class* for every shifting (φ_x, φ_y). The class that corresponds to the shifting (φ_x, φ_y) consists of all the path requests r_i^{st} whose origin (a_i, t_i) belongs to a SW quadrant of a tile in the tiling that uses the shifting (φ_x, φ_y).

Sketch graph and paths. Consider a fixed tiling. The *sketch graph* is the graph obtained from the grid after coalescing each tile into a single node. There is a directed edge (s_1, s_2) between two tiles s_1, s_2 in the sketch graph if there is a directed edge $(\alpha, \beta) \in E^{st}$ such that $\alpha \in s_1$ and $\beta \in s_2$. Let p^s denote the projection of a path p in the grid to the sketch graph. We refer to p^s as the *sketch path* corresponding to p. Note that the length of p^s is at most $\lceil |p|/k \rceil + 1$.

3 Outline of our Algorithm

For the sake of simplicity we focus hereafter on the case of unit capacities, namely $B = c = 1$. Extension to non-unit capacities are discussed in Section 6.1.

Packet requests are categorized into three categories: short, medium, and long, according to the source-destination distance of each packet. A separate approximation algorithm is executed for each category. The algorithm returns a highest throughput solution among the solutions computed for the three categories.

Notation. Two thresholds are used for defining short, medium, and long requests: $\ell_M \triangleq 3 \ln n, \ell_S \triangleq 3 \cdot \ln(\ell_M) = 3 \cdot \ln(3 \ln n)$.

▶ **Definition 1.** A request r_i is a *short* request if $b_i - a_i \leq \ell_S$. A request r_i is a *medium* request if $\ell_S < b_i - a_i \leq \ell_M$. A request r_i is a *long* request if $b_i - a_i > \ell_M$.

We use a deterministic algorithm for the class of short packets, and in Theorem 7 we prove that this deterministic algorithm achieves a constant approximation ratio. We use a randomized algorithm for each of the classes of medium and long packets; in Theorem 15 we prove that this randomized algorithm achieves a constant approximation ratio in expectation for each of these classes. Thus, we obtain the following corollary.

▶ **Corollary 2** (Main Result). *If $B = c = 1$, then there exists a randomized approximation algorithm for the Max-Pkt-Line problem that achieves a constant approximation ratio in expectation.*

In Section 6.1 we discuss non-unit capacities, give the approximation ratio for this case and show that we achieve a constant approximation ratio as long as the ratio B/c does not depend on the input size.

4 Approximation Algorithm for Short Packets

In this section we present a constant ratio deterministic approximation algorithm for short packets. This algorithm, which is key to achieving the results of the present paper, makes use of a new combinatorial lemma that we prove in the next subsection, stating, roughly speaking, that if packets from a given set of packets are allowed to stay put in buffers (i.e., use horizontal edges in the grid) only a limited number of time steps, $2d$ (where d is the longest source-destination distance in the set of packets), then the optimal solution is decreased by only a constant factor. We believe that this lemma may find additional applications in future work on routing and scheduling problems.

4.1 Bounding Path Lengths in the Grid

In this section we prove that bounding, from above, the number of horizontal edges along a path incurs only a small reduction in the throughput. Previously known bounds along these lines hold only for fractional solutions [5], while we present here the first such claim for integral schedules.

Let R_d denote a set of packet requests r_i, $i \geq 1$, such that $b_i - a_i \leq d$ for any i. Consider the paths in the space-time grid that are allocated to the accepted requests. We prove that restricting the path lengths to $2d$ decreases both the optimal fractional and the optimal *integral* throughput only by a multiplicative factor of $O(c/B)$. We note that if the ratio B/c is a constant, then we are guaranteed an optimal solution which is only a constant away from the unrestricted optimal solution.

Notation. For a single commodity acyclic flow f_i, let $p_{\max}(f_i)$ denote the diameter of the support of f_i (i.e., length of longest path[2]). For an MCF $F = \{f_i\}_{i \in I}$, let $p_{\max}(F) \triangleq \max_{i \in I} p_{\max}(f_i)$. Let $F^*_{\mathit{frac}}(R)$ (respectively, $F^*_{\mathit{int}}(R)$) denote a maximum throughput fractional (resp., integral) MCF with respect to the set of requests R. Similarly, let $F^*_{\mathit{frac}}(R \mid p_{\max} < d')$ (respectively, $F^*_{\mathit{int}}(R \mid p_{\max} < d')$) denote a maximum throughput fractional (resp., integral) MCF with respect to the set of requests R subject to the additional constraint that the maximum path length is at most d'.

▶ **Lemma 3.**

$$F^*_{\mathit{frac}}(R_d \mid p_{\max} \leq 2d) \geq \frac{c}{B+2c} \cdot F^*_{\mathit{frac}}(R_d),$$
$$F^*_{\mathit{int}}(R_d \mid p_{\max} \leq 2d) \geq \frac{c}{2(B+c)} \cdot F^*_{\mathit{int}}(R_d).$$

Proof. Partition the space-time grid into *slabs* S_j of "width" d. Slab S_j contains the vertices (v, k), where $k \in [(j-1) \cdot d, j \cdot d]$, $j \geq 1$. We refer to vertices of the form (v, jd) as the *boundary* of S_j. Note that if $v - u \leq d$, then the forward-only vertical path from (u, jd) to $(v, jd + (v-u))$ is contained in slab S_{j+1}.

[2] Without loss of generality, we may assume that each single commodity flow f_i is acyclic.

We begin with the fractional case. Let $f^* = F^*_{frac}(R_d)$ denote an optimal fractional solution for R_d. Consider request r_i and the corresponding single commodity flow f^*_i in f^*. Decompose f^*_i to flow-paths $\{p_\ell\}_\ell$. For each flow-path p_ℓ in f^*_i, let p'_ℓ denote the prefix of p_ℓ till it reaches the boundary of a slab. Note that $p'_\ell = p_\ell$ if p_ℓ is confined to a single slab. If $p'_\ell \subsetneq p_\ell$, then let (v, jd) denote the last vertex of p'_ℓ. Namely, the path p'_ℓ begins in $(a_i, t_i) \in S_j$ and ends in (v, jd). Let q''_ℓ denote the forward-only path from (v, jd) to $(b_i, jd + (b_i - v))$. (If $p'_\ell = p_\ell$, then q''_ℓ is an empty path.) Note that q''_ℓ is confined to the slab S_{j+1}. We refer to the vertex (v, jd) in the intersection of p'_ℓ and q''_ℓ as the *boundary* vertex. Let g_i denote the fractional single commodity flow for request r_i obtained by adding the concatenated flow-paths $q_\ell \triangleq p'_\ell \circ q''_\ell$ each with the flow amount of f^*_i along p_ℓ. Define the MCF g by $g(e) \triangleq \sum_{i \in I} g_i(e)$. For every edge e, part of the flow $g(e)$ is due to prefixes p'_ℓ, and the remaining flow is due to suffixes q''_ℓ. We denote the part due to prefixes by $g_{pre}(e)$ and refer to it as the prefix-flow. We denote the part due to suffixes by $g_{suf}(e)$ and refer to it as the suffix-flow. By definition, $g(e) = g_{pre}(e) + g_{suf}(e)$.

The support of g_i is contained in the union of two consecutive slabs. Hence, the diameter of the support of g_i is bounded by $2d$. Hence $p_{\max}(g) \leq 2d$.

Clearly, $|g_i| = |f^*_i|$ and hence $|g| = |f^*|$. Set $\rho = c/(B + 2c)$. To complete the proof, it suffices to prove that $\rho \cdot g$ satisfies the capacity constraints. Indeed, for a "store" edge $e = (v, t) \to (v, t + 1)$, we have $g_{suf}(e) = 0$ and $g_{pre}(e) \leq f^*(e) \leq B$. For a "forward" edge $e = (v, t) \to (v + 1, t + 1)$ we have: $g_{pre}(e) \leq f^*(e) \leq c$. On the other hand, $g_{suf}(e) \leq B + c$. The reason is as follows. All the suffix-flow along e starts in the same boundary vertex (u, jd) below e. The amount of flow forwarded by (u, jd) is bounded by the amount of incoming flow, which is bounded by $B + c$. This completes the proof of the fractional case.

We now prove the integral case. The proof is a variation of the proof for the fractional case in which the supports of pre-flows and suffix-flows are disjoint. Namely, one alternates between slabs that support prefix-flow and slabs that support suffix-flow.

In the integral case, each accepted request r_i is allocated a single path p_i, and the allocated paths satisfy the capacity constraints. As in the fractional case, let $q_i \triangleq p'_i \circ q''_i$, where p'_i is the prefix of p_i till a boundary vertex (v, jd), and q''_i is a forward-only path. We need to prove that there exists a subset of at least $c/(2(B + c))$ of the paths $\{q_i\}_i$ that satisfy the capacity constraints. This subset is constructed in two steps.

First, partition the requests into "even" and "odd" requests according to the parity of the slab that contains their origin (a_i, t_i). (The parity of request r_i is simply the parity of $\lceil t_i/d \rceil$.) Pick a part that has at least half of the accepted requests in $F^*_{int}(R_d)$; assume w.l.o.g. that such a part is the part of the even slabs. Then, we only keep accepted requests whose origin belong to even slabs.

In the second step, we consider all boundary vertices $(v, j \cdot d)$. For each boundary vertex, we keep up to c paths that traverse it, and delete the remaining paths if such paths exist. In the second step, again, at least a $c/(B + c)$ fraction of the paths survive. It follows that altogether at least $c/(2(B + c))$ of the paths survive.

We claim that the remaining paths satisfy the capacity constraints. Note that prefixes are restricted to even slabs, and suffixes are restricted to odd slabs. Thus, intersections, if any, are between two prefixes or two suffixes. Prefixes satisfy the capacity constraints because they are prefixes of $F^*_{int}(R_d)$. Suffixes satisfy the capacity constraints because if two suffixes intersect, then they start in the same boundary vertex. However, at most c paths emanating from every boundary vertex survive. Hence, the surviving paths satisfy the capacity constraints, as required. This completes the proof of the lemma. ◀

We note that if the ratio B/c is bounded by a constant, then Lemma 3 guarantees an optimal solution which is only a (different) constant away from the unrestricted optimal solution.

4.2 The Algorithm for Short Packets

Short requests are further partitioned into four classes, defined as follows. Consider four tilings each with side length $k \triangleq 4\ell_S$ and horizontal and vertical shifts in $\varphi_x, \varphi_y \in \{0, k/2\}$.[3] The four possible shifts define four classes: The packets of a certain class (shift) are the packets whose source nodes reside in the SW quadrants of the tiles according to a given shift. Observe that each packet request belongs to exactly one class. We say that a path p_i from (a_i, t_i) to the row of b_i is *confined to a tile* if p_i is contained in one tile. We bound from above the path lengths by $2\ell_S$ so

We claim that by exhaustive search, it is possible to efficiently compute a maximum throughput solution for each class, under the restriction that each path is of length at most $2\ell_S$. The algorithm computes an optimal (bounded path length) solution for each class, and returns a highest throughput solution among the four solutions.

The polynomial running time of the exhaustive search algorithm per class is based on the two following observations.

▶ **Observation 4.** *A path of length at most $k/2 = 2\ell_S$ that begins in the SW quadrant of tile T is confined to T.*

Proof. The tile side length equals $k = 4\ell_S$. If the origin of a request is in the SW quadrant of a tile and the path length is at most $2\ell_S = k/2$, then the end of the path belongs to the same tile. ◀

▶ **Observation 5** ([6, 9, 8])**.** *Partition the packets according to their source node. For each node v, order the packets with source node v in increasing order of their target point. There exists an optimal solution that does not include any packet with rank more than 2 in that ordering.*[4]

▶ **Lemma 6.** *If $B = c = 1$, an optimal solution for each class in which paths are confined to their origin tile is computable in time polynomial in n and M.*

Proof. Fix a tile T. Let X denote the set of short requests in T, having rank at most 2 according to the ordering defined in Observation 5. By Observation 5, an optimal solution can be computed out of the set of packets X. Let Y be the set of paths in T. We perform an exhaustive search to find the optimal solution.

It suffices for the exhaustive search to consider all possibly partial functions $f : X \to Y$, and for each such function check that (1) each packet in the domain of f can be served by the path associated with it, and (2) no two path in the image of f intersect. Then pick, among all functions that pass the checks, the one with the largest domain.

The size of X is at most $2(k/2)^2$ since there are at most $(k/2)^2$ possible source nodes. The size of Y is at most $(k/2)^2\binom{2k}{k}$ (there are $(k/2)^2$ possible source nodes, and a path is defined by the steps in which it goes horizontally, and those where it goes vertically).

Therefore the number of possibly partial functions per tile is at most $2^{|X|} \cdot |Y|^{|X|} < 2^{2(k/2)^2} \cdot \left((k/2)^2 2^{2k}\right)^{2(k/2)^2} \leq 2^{poly(k)}$. Furthermore, given a function f the two needed checks

[3] Recall that $\ell_M \triangleq 3\ln n$ and $\ell_S \triangleq 3 \cdot \ln(\ell_M) = 3 \cdot \ln(3\ln n)$.
[4] Rank $B + c$ for non unit capacities.

can be done in time at most $O(|X| + |X|^2 \cdot k^2) = O(|X|^2 \cdot k^2)$. Since $k = O(\log \log n)$ for the case of the short packets, and $|X| \leq 2(k/2)^2$, the total time of the exhaustive search per tile is $poly(n)$.

The number of tiles that contain a request is bounded by the number of requests M. Hence, the running time of the algorithm for short requests is polynomial in n and M. ◀

▶ **Theorem 7.** *The approximation ratio of the algorithm for short requests is* $\frac{1}{16}$.

Proof. The short requests are partitioned to 4 classes. Then, for each class and tile, the exhaustive algorithm computes a solution which with cardinality at least a 1/4 of the optimal one, by the integral version of Lemma 3. ◀

5 Approximation Algorithm for Medium & Long Requests

We use the same algorithm for the two classes of medium and long requests, the only difference being some parameters of the algorithm. As indicated above, we consider at this point the case of unit capacities ($B = c = 1$). We further note that the approximation ratio of the algorithm for these classes is with respect to the optimal *fractional* solution.

Notation. Let $R_{d_{\min},d_{\max}}$ denote the set of packet requests whose source-to-destination distance is greater than $d_{\min}$ and at most $d_{\max}$. Formally, $R_{d_{\min},d_{\max}} \triangleq \{r_i \mid d_{\min} < b_i - a_i \leq d_{\max}\}$.

Parametrization. When applied to medium requests we use the parameter $d_{\max} = \ell_M$ and $d_{\min} = \ell_S$. When applied to long requests the parameters are $d_{\max} = n$ and $d_{\min} = \ell_M$. Note that these parameters satisfy $d_{\min} = 3 \cdot \ln d_{\max}$.

5.1 The Algorithm for $R_{d_{\min},d_{\max}}$

The algorithm for $R_{d_{\min},d_{\max}}$ proceeds as follows. To simplify notation, we abbreviate $R_{d_{\min},d_{\max}}$ by R. The parameters $d_{\min}$ and $d_{\max}$ must satisfy that $d_{\min} = 3 \cdot \ln d_{\max}$. We use the randomized rounding procedure by Raghavan [12, 13]. The description of this randomized rounding procedure is deferred to the full version.

1. Reduce the packet requests in R to path requests R^{st} over the space-time graph G^{st}.
2. Compute a maximum throughput fractional MCF $F \triangleq \{f_i\}_{r_i \in R^{st}}$ with edge capacities $\tilde{c}(e) = \lambda$ (for $\lambda = 1/(\beta(3) \cdot 6)$) [5] and bounded diameter $p_{\max}(F) \leq 2d_{\max}$. We remark that this MCF can be computed in time polynomial in n - the number of nodes and M - the number of requests.[6]

[5] The function $\beta : (-1, \infty) \to \mathbb{R}$ is defined by $\beta(\varepsilon) \triangleq (1 + \varepsilon)\ln(1 + \varepsilon) - \varepsilon$.

[6] Since always $d_{\max} \leq n$, we can consider a space-time grid of size at most $n \times (M \cdot 2n)$, which can be constructed by going over the release times of all M requests, eliminating "unnecessary" time steps. One can then compute a maximum throughput fractional solution with bounded diameter on this grid using linear programming. This is true because the constraint $p_{\max}(f_i) \leq d'$ is a linear constraint and can be imposed by a polynomial number of inequalities (i.e, polynomial in n and d'). For example, one can construct a product network with $(d' + 1)$ layers, and solve the MCF problem over this product graph.

3. Partition R to 4 classes $\{R^j\}_{j=1}^4$ according to the shift that results in the source node being in the SW quadrant of a $k \times k$ tiling, where $k \triangleq 2d_{\min} = 6 \ln d_{\max}$ (see Section 2.3). Pick a class R^j such that the throughput of F restricted to R^j is at least a quarter of the throughput of F, i.e., $|F(R^j)| \geq |F|/4$.
4. For each request $r_i \in R^j$, apply randomized rounding independently to f_i. The outcome of randomized rounding per request $r_i \in R^j$ is either "reject" or a path p_i in G^{st}. Let $R_{rnd} \subseteq R^j$ denote the subset of requests r_i that are assigned a path p_i by the randomized rounding procedure.
5. Let $R_{\mathit{fltr}} \subseteq R_{rnd}$ denote the requests that remain after applying filtering (described in Section 5.2).
6. Let $R_{quad} \subseteq R_{\mathit{fltr}}$ denote the requests for which routing in first quadrant is successful (as described in Section 5.3).
7. Complete the path of each request in R_{quad} by applying crossbar routing (as described in Section 5.4).

5.2 Filtering

Notation. Let e denote an edge in the space-time grid G^{st}. Let e^s denote an edge in the sketch graph (see Section 2.3). We view e^s also as the set of edges in G^{st} that cross the tile boundary that corresponds to the sketch graph edge e^s. The path p_i is a random variable that denotes the path, if any, that is chosen for request r_i by the randomized rounding procedure. For a path p and an edge e let $\mathbb{1}_p(e)$ denote the 0-1 indicator function that equals 1 iff $e \in p$.

The set of filtered requests R_{fltr} is defined as follows (recall that $\lambda = 1/(\beta(3) \cdot 6)$).

▶ **Definition 8.** A request $r_i \in R_{\mathit{fltr}}$ if and only if r_i is accepted by the randomized rounding procedure, and for every sketch-edge e^s in the sketch-path p_i^s it holds that $\sum_i \mathbb{1}_{p_i^s}(e^s) \leq 4\lambda \cdot k$.

▶ **Claim 9.** $\mathbf{E}\left[|R_{\mathit{fltr}}|\right] \geq \left(1 - O(\frac{1}{k})\right) \cdot \mathbf{E}\left[|R_{rnd}|\right]$.

Proof. We begin by bounding the probability that at least $4\lambda k$ sketch paths cross a single sketch edge.

▶ **Lemma 10** (Chernoff Bound). *For every edge e^s in the sketch graph,*[7]

$$\mathbf{Pr}\left[\sum_i \mathbb{1}_{p_i^s}(e^s) > 4\lambda k\right] \leq e^{-k/6}. \tag{1}$$

Proof of lemma. Recall that the edge capacities in the MCF F are λ. The capacity constraint $\sum_i f_i(e) \leq \lambda$ implies that $f_i(e) \leq \lambda$. Each sketch edge e^s corresponds to the grid edges between adjacent tiles. Since the demand of each request is 1, it follows that $f_i(e^s) \leq 1$.

For every edge e and request r_i, we have $\mathbf{E}\left[\mathbb{1}_{p_i^s}(e^s)\right] = \mathbf{Pr}\left[\mathbb{1}_{p_i^s}(e^s) = 1\right] = f_i(e^s) \leq 1$. Fix a sketch edge e^s. The random variables $\{\mathbb{1}_{p_i^s}(e^s)\}_i$ are independent 0-1 variables. Moreover, $\sum_i \mathbf{E}\left[\mathbb{1}_{p_i^s}(e^s)\right] = \sum_i f_i(e^s) = \sum_{e \in e^s} \sum_i f_i(e) \leq \lambda \cdot k$. By Chernoff bound[8]

$$\mathbf{Pr}\left[\sum_i \mathbb{1}_{p_i^s}(e^s) > 4 \cdot \sum_i \mathbf{E}\left[\mathbb{1}_{p_i^s}(e^s)\right]\right] < e^{-\beta(3)\cdot\lambda k} = e^{-k/6}. \quad ◀$$

[7] The e in the RHS is the base of the natural logarithm.

[8] We use the following version of Chernoff Bound [12, 15]. Let $\{X_i\}_i$ denote a sequence of independent random variables attaining values in $[0,1]$. Assume that $\mathbf{E}[X_i] \leq \mu_i$. Let $X \triangleq \sum_i X_i$ and $\mu \triangleq \sum_i \mu_i$. Then, for $\varepsilon > 0$, $\mathbf{Pr}[X \geq (1+\varepsilon) \cdot \mu] \leq e^{-\beta(\varepsilon)\cdot\mu}$.

A request $r_i \in R_{rnd}$ is not in R_{fltr} iff at least one of the edges $e^s \in p_i^s$ has more than $2\lambda k$ paths on it. Hence, by a union bound,

$$\mathbf{Pr}\left[r_i \notin R_{\mathit{fltr}} \mid r_i \in R_{rnd}\right] \leq |p_i^s| \cdot e^{-k/6} \leq \left(\left\lceil \frac{2d_{\max}}{k} \right\rceil + 2\right) \cdot e^{-\ln d_{\max}} = O\left(\frac{1}{k}\right),$$

since $k = 6 \ln d_{\max}$. ◀

5.3 Routing in the First Quadrant

In this section, we deal with the problem of evicting as many requests as possible from their origin quadrant to the boundary of the origin quadrant.

▶ Remark. Because $k/2 \leq d_{\min}$ every request that starts in a SW quadrant of a tile must reach the boundary (i.e., top or right side) of the quadrant before it can reach its destination.

The maximum flow algorithm. Consider a tile T. Let X denote set of requests r_i whose source (a_i, t_i) is in the south-west quadrant of T. We say that a subset $X' \subseteq X$ is *quadrant feasible* (in short, feasible) if it satisfies the following condition: There exists a set of edge disjoint paths $\{q_i \mid r_i \in X'\}$, where each path q_i starts in the source (a_i, t_i) of r_i and ends in the top or right side of the SW quadrant of T.

We employ a maximum-flow algorithm to solve the following problem.
Input: A set of requests X whose source is in the SW quadrant of T.
Goal: Compute a maximum cardinality quadrant-feasible subset $X' \subseteq X$.

The algorithm is simply a maximum-flow algorithm over the following network, denoted by $N(X)$. Augment the quadrant with a super source $\tilde{s}$ and a super sink $\tilde{t}$. The super source $\tilde{s}$ is connected to every source (a_i, t_i) (of a request $r_i \in X$) with a unit capacity directed edge. (If γ requests share the same source, then the capacity of the edge is γ.) There is a unit capacity edge from every vertex in the top side and right side of the SW quadrant of T to the super sink $\tilde{t}$. All the grid edges are assigned unit capacities. Compute an integral maximum flow in the network. Decompose the flow to unit flow paths. These flow paths are the paths that are allocated to the requests in X'.

Analysis. Fix a tile T and let $R_T \subseteq R_{\mathit{fltr}}$ denote the set of requests in R_{fltr} whose source vertex is in the SW quadrant of T. Let $R'_T \subseteq R_T$ denote the quadrant-feasible subset of maximum cardinality computed by the max-flow algorithm. Let $R_{quad} = \bigcup_T R'_T$.

We now prove the following theorem that relates the *expected value* of $|R'_T|$ to the expected value of $|R_T|$. Observe that it is not always true that the same relation holds for any specific R_T that results from a specific random tape used by the randomized rounding procedure.

▶ **Theorem 11.** *[10, 11]* $\mathbf{E}_\tau\left[|R_{quad}|\right] \geq 0.93 \cdot \mathbf{E}_\tau\left[|R_{\mathit{fltr}}|\right]$, *where τ is the probability space induced by the randomized rounding procedure.*

Proof. By linearity of expectation, it suffices to prove that $\mathbf{E}_\tau\left[|R'_T|\right] \geq 0.93 \cdot \mathbf{E}_\tau\left[|R_T|\right]$, for any given tile T.

The proof goes along the following lines. We define a certain capacity constraint over rectangles; this definition makes use of the capacity of the boundary of the rectangles, and the number of requests within them. We define the set $\hat{R}_T \subseteq R_T$ to be a set of requests based on the capacity constraints of the rectangles containing the source of the requests. We prove that: (1) The set $\hat{R}_T$ thus defined is feasible, and (2) $\mathbf{E}_\tau\left[|\hat{R}_T|\right] \geq 0.93 \cdot \mathbf{E}_\tau\left[|R_T|\right]$. By the algorithm, R'_T is of maximum cardinality (maximum flow), therefore, $|R'_T| \geq |\hat{R}_T|$, and the theorem follows.

We now describe how the feasible subset $\hat{R}_T$ is defined. Consider a subset S of the vertices in the SW quadrant of T. Let $\mathsf{dem}(S)$ denote the number of requests in R_T whose origin is in S. Let $\mathsf{cap}(S)$ denote the capacity of the edges in the network $N(X)$ that emanate from S. By the min-cut max-flow theorem, a set of requests $X \subseteq R_T$ is feasible if and only if $\mathsf{dem}(S) \leq \mathsf{cap}(S)$ for every cut $S \cup \{\tilde{s}\}$ in the network $N(X)$.

In fact, it is not necessary to consider all the cuts. It suffices to consider only axis parallel rectangles contained in the quadrant T. The reason is that without loss of generality, the set S is connected in the underlying undirected graph of the grid (i.e., consider each connected components of S separately). Every "connected" set S can be replaced by the smallest rectangle $Z(S)$ that contains S. We claim that $\mathsf{cap}(S) \geq \mathsf{cap}(Z(S))$ and $\mathsf{dem}(S) \leq \mathsf{dem}(Z(S))$. Indeed, there is an injection from the edges in the cut of $Z(S)$ to the edges in the cut of S. For example, a vertical edge e in the cut of $Z(S)$ is mapped to the topmost edge e' in the cut of S that is in the column of e. Hence, $\mathsf{cap}(Z(S)) \leq \mathsf{cap}(S)$. On the other hand, as $S \subseteq Z(S)$, it follows that $\mathsf{dem}(S) \leq \mathsf{dem}(Z(S))$. Hence if $\mathsf{dem}(S) > \mathsf{cap}(S)$, then $\mathsf{dem}(Z(S)) > \mathsf{cap}(Z(S))$.

We say that a rectangle Z is *overloaded* if $\mathsf{dem}(Z) > \mathsf{cap}(Z)$. The set $\hat{R}_T$ is defined to be the set of requests $r_i \in R_T$ such that the source of r_i is not included in an overloaded rectangle. Namely, $\hat{R}_T \triangleq \{r_i \in R_T \mid \forall \text{ rectangles } Z : Z \text{ is overloaded } \Rightarrow (a_i, t_i) \notin Z\}$. Consider an $x \times y$ rectangle Z. We wish to bound from above the probability that $\mathsf{dem}(Z) > \mathsf{cap}(Z)$. Note that $\mathsf{cap}(Z) = x + y$. Since requests in R_T that start in Z must exit the quadrant, it follows that $\mathsf{dem}(Z)$ is bounded by the number of paths in R_T that cross the top or right side of Z (note that there might be additional paths that do not start in Z but cross Z.). The amount of flow that emanates from Z is bounded by $\lambda \cdot (x+y)$ (the initial capacities are λ and there are $x+y$ edges in the cut). By the randomized rounding procedure, $\mathbf{Pr}[e \in p_i] = f_i(e)$. Summing up over all the edges in the cut of Z and the requests in R_T, the expected number of paths in R_T that cross the cut of Z equals the flow of the request in R_T, which, in turn, is bounded by the capacity $\lambda \cdot (x+y)$. As the paths of the requests are independent random variables, we obtain:[9] $\mathbf{Pr}[\mathsf{dem}(Z) > \mathsf{cap}(Z)] \leq \mathbf{Pr}\left[\sum_{i \in R_T} |p_i \cap cut(Z)| > (x+y)\right] \leq (\lambda \cdot e)^{x+y}$.

For each x, y, each source (a_i, t_i) is contained in at most $x \cdot y$ rectangles with side lengths $x \times y$. By applying a union bound, the probability that (a_i, t_i) is contained in an overloaded rectangle is bounded from above by

$$\begin{aligned}\mathbf{Pr}[\exists \text{ overloaded rectangle } Z : (a_i, t_i) \in Z] &\leq \sum_{x=1}^{\infty}\sum_{y=1}^{\infty} xy \cdot (\lambda \cdot e)^{x+y} \\ &\leq \frac{(\lambda \cdot e)^2}{(1 - \lambda \cdot e)^4} \leq 0.07, \qquad (2)\end{aligned}$$

and the theorem follows. ◀

Routing within the first tile (see Section 5.4) requires however a stronger upper bound on the number of requests that emanate from each side of the quadrant, namely that at most $k/3$ paths reach each side of the quadrant. Using a simple procedure (e.g., taking the solution and greedily eliminating paths) one can have a solution for which this condition holds, and with cardinality only a constant fraction smaller.

[9] Using the following version of the Chernoff bound: $\mathbf{Pr}[X \geq \alpha \cdot \mu] \leq \left(\frac{e}{\alpha}\right)^{\alpha \cdot \mu}$.

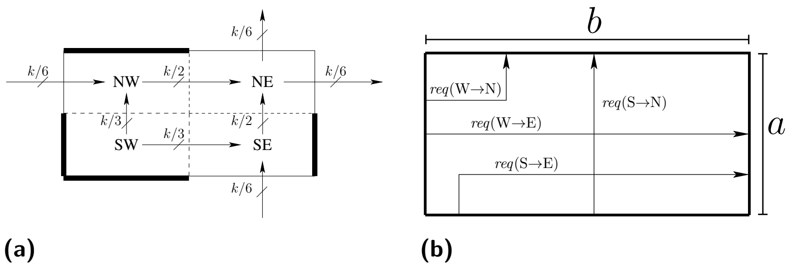


Figure 1 (a) Partitioning of a tile to quadrants [9]. Thick lines represent "walls" that cannot be crosses by paths. Sources may reside only in the SW quadrant of a tile. Maximum flow amounts crossing quadrant sides appears next to each side. Final destinations of paths are assumed (pessimistically) to be in the top row of the NE quadrant. (b) Crossbar routing: flow crossing an $a \times b$ grid [9].

▶ **Corollary 12.** *Let R'_{quad} be the set of quadrant-feasible paths such that at most $k/3$ paths reach each side of each quadrant. Then, $\mathbf{E}_\tau\left[|R'_{quad}|\right] \geq \Omega(1) \cdot \mathbf{E}_\tau\left[|R_{fltr}|\right]$, where τ is the probability space induced by the randomized rounding procedure.*

5.4 Detailed Routing

In this section we deal with computing paths for requests $r_i \in R_{quad}$ starting from the boundary of the SW quadrant that contains the source (a_i, t_i) till the destination row b_i. These paths are concatenated to the paths computed in the first quadrant to obtain the *final* paths of the accepted requests. Detailed routing is based on the following components: (1) The projections of the final path and the path p_i to the sketch graph must coincide. (2) Each tile is partitioned to quadrants and routing rules within a tile are defined. (3) Crossbar routing within each quadrant is applied to determine the final paths (except for routing in SW quadrants in which paths are already assigned).

Sketch paths and routing between tiles. Each path p_i computed by the randomized rounding procedure is projected to a sketch path p^s_i in the sketch graph. The final path $\hat{p}_i$ assigned to request r_i traverses the same sequence of tiles, namely, the projection of $\hat{p}_i$ is also p^s_i.

Routing rules within a tile [6]. Each tile is partitioned to quadrants as depicted in Figure 1a. The bold sides (i.e., "walls") of the quadrants indicate that final paths may not cross these walls. The classification of the requests ensures that source vertices of requests reside only in SW quadrants of tiles. Final paths may not enter the SW quadrants; they may only emanate from them. If the endpoint of a sketch path p^s_i ends in tile T, then the path $\hat{p}_i$ must reach a copy of its destination b_i in T. Reaching the destination is guaranteed by having $\hat{p}_i$ reach the top row of the NE quadrant of T (and thus it must reach the row of b_i along the way).

Crossbar routing. [9]. Routing in each quadrant is simply an instance of routing in a (uni-directional) 2D grid where requests enter from two adjacent sides and exit from the opposite sides. Figure 1b depicts such an instance in which requests arrive from the left and bottom sides and exit from the top and right side. The following claim characterizes when crossbar routing succeeds.

▶ **Claim 13** ([9]). *Consider a 2-dimensional directed $a \times b$ grid. A set of requests can be routed from the bottom and left boundaries of the grid to the opposite boundaries if and only if the number of requests that should exit each side is at most the length of the corresponding side.*

We conclude with the following claim.

▶ **Claim 14.** *Detailed routing succeeds in routing all the requests in R_{quad}.*

Proof sketch. The sketch graph is a directed acyclic graph. Sort the tiles in topological ordering. Within each tile, order the quadrants also in topological order: SW, NW, SE, NE. Prove by induction on the position of the quadrant in the topological ordering that detailed routing up to and including the quadrant succeeds. The claim for all SW quadrants follows because this routing is done along the path that result of the randomized rounding step of the requests in R_{quad}. Now note that filtering ensures that the number of paths between tiles is at most $2\lambda k < k/6$. Routing in the first quadrant ensures that the number of paths emanating from each side of a SW quadrant is at most $k/3$. The induction step follows by applying Claim 13. ◀

5.5 Approximation Ratio

▶ **Theorem 15.** *The approximation ratio of the algorithm for packet requests in $R_{d_{\min},d_{\max}}$, for $d_{\min} = 3 \cdot \ln d_{\max}$, is constant in expectation.*

Proof. We follow the algorithm, as defined in Section 5.1, stage by stage.

Stage 2 computes a fractional maximum multi-commodity flow on a network with reduced edge capacities, and with the requirement that all flows have bounded diameter. By Lemma 3, bounding path lengths in the MCF results in a solution of at least a 1/3 fraction of the unrestricted one, and the scaling of the capacities in the space-time grid results in a solution which is at least a $\lambda = \Omega(1)$ fraction of the latter.

Stage 3 classifies the requests into 4 classes and picks only the one for which the multi-commodity flow solution is the highest, hence resulting in a solution of at least a 1/4 fraction of the solution of the previous stage.

Stage 4 applies a randomized rounding procedure to the flows that are picked in stage 3. The expected size of the solution is equal to the total flow left from the previous stage (but the solution might not be feasible).

Stage 5 applies a filtering procedure to the solution of the previous stage, in order to get a feasible solution on the sketch graph. By Claim 9, the expected size of the feasible solution is at least a $1 - O(1/k)$ fraction of the solution given by stage 4. Observe that $1 - O(1/k) = O(1)$ (in fact $k = \Omega(\log \log n)$ in any relevant invocation of the algorithm).

Stage 6 further reduces the size of the solution when the algorithm selects a subset of the requests that have survived so far, using a maximum flow algorithm applied to each SW quadrant. This is done in order to allow for the solution to be feasible in the original space-time grid. By Corollary 12 the expected size of the solution after this stage is an $\Omega(1)$ fraction of the expected size before this stage.

Stage 7 gives the final routing without further reducing the size of the solution.

We conclude that the algorithm for medium and long requests is a randomized $O(1)$ approximation algorithm (in fact with respect to the fractional optimum). ◀

6 Extensions

6.1 Non-unit Capacities & Buffer Sizes

Our results extend to arbitrary values of B and c, with an (additional) multiplicative penalty in the approximation ratio of $O(B/c)$ in certain cases. In this section we outline the required changes in the algorithm when B/c does not depend on the input size, i.e., $B = \Theta(c)$. For this case, our results will give a randomized approximation algorithm with constant approximation ratio. We now outline the required modifications to handle this case, explaining the modifications in each component of the algorithm, and then the overall structure of the modified algorithm.

Adapting the algorithm for short packets (the exhaustive search algorithm). Exhaustive search for arbitrary B and c can be done in polynomial time provided that the distance of each packet request is at most $\ln(\ell_S)$ (see [11, Lemma 7]). This means that for general B and c there is an additional category of requests, called *very short* requests, on which the exhaustive search will be applied. The remaining packets are divided into short, medium, and long requests, as for the case of $B = c = 1$, and the distance of short requests is lower bounded by $\ln(\ell_S)$.

Adapting the Algorithm for $R_{d_{\min},d_{\max}}$ (for short, medium, and long packets). We adapt the algorithm to the case where $B = c = \gamma$, where $\gamma \in \mathbb{N}^{>0}$. Note that while we consider here non-unit capacities, the flow demands (packets) are not changed, i.e., they remain unit demands. Consider a 3-dimensional view of the grid where there are γ "floors", each floor for a single capacity slice out of γ. This view of the grid partitions the routing problem at hand to γ problems, where at each of them the capacities are unit. Now, apply the algorithm for $R_{d_{\min},d_{\max}}$ on each of these floors. The approximation ratio of the revised algorithm follows from linearity of expectation.

Putting things together. We now describe the general structure of the algorithm for arbitrary B and c. The packets are partitioned into four classes *very short, short, medium,* and *long.* As given above, we have a constant approximation algorithm for very short packets for arbitrary B and c.

For the other three categories, we set both the buffer sizes and link capacities to $\min\{B, c\}$, and apply the (modified) $R_{d_{\min},d_{\max}}$ for the case $B = c$. Observe that the *fractional* optimum incurs a penalty of at most a factor of $O(B/c)$ as a result of this capacity change. Since this algorithm gives a constant approximation compared to the *fractional* optimum, we get an $O(B/c)$-approximation algorithm compared to the optimum on the network with non-modified capacities.

We can now conclude with the following theorem.

▶ **Theorem 16.** *There exists a randomized algorithm for the Max-Pkt-Line problem, such that if $B = O(c)$, then its approximation ratio is a (different) constant.*

6.2 Supporting Soft Deadlines

In this section we argue that if packets have deadlines, we achieve a constant approximation ratio when the produced solution is allowed to miss deadlines by at most $O(\log n)$ time units. This is achieved, simply, by reducing a request with deadlines to a path request which its destination set is a copy of the destination vertex with a time index which is less than the time of the deadline. Since the tiling has "resolution" of at most $O(\log n)$, the detailed routing might "miss" the deadline by the tile's side length.

Note that the algorithm for short requests (or very short for non-unit capacities) can handle requests with deadlines, and achieve the same performance while respecting hard deadlines, because it employs exhaustive search[10].

References

1 Micah Adler, Arnold L. Rosenberg, Ramesh K. Sitaraman, and Walter Unger. Scheduling time-constrained communication in linear networks. *Theory Comput. Syst.*, 35(6):599–623, 2002. doi:10.1007/s00224-002-1001-6.

2 William Aiello, Eyal Kushilevitz, Rafail Ostrovsky, and Adi Rosén. Dynamic routing on networks with fixed-size buffers. In *SODA*, pages 771–780, 2003. doi:10.1145/644108.644236.

3 Stanislav Angelov, Sanjeev Khanna, and Keshav Kunal. The network as a storage device: Dynamic routing with bounded buffers. *Algorithmica*, 55(1):71–94, 2009. (Appeared in APPROX-05). doi:10.1007/s00453-007-9143-1.

4 Baruch Awerbuch, Yossi Azar, and Amos Fiat. Packet routing via min-cost circuit routing. In *ISTCS*, pages 37–42, 1996.

5 Yossi Azar and Rafi Zachut. Packet routing and information gathering in lines, rings and trees. In *ESA*, pages 484–495, 2005. (See also manuscript in http://www.cs.tau.ac.il/~azar/). doi:10.1007/11561071_44.

6 Guy Even and Moti Medina. An $O(\log n)$-Competitive Online Centralized Randomized Packet-Routing Algorithm for Lines. In *ICALP (2)*, pages 139–150, 2010. doi:10.1007/978-3-642-14162-1_12.

7 Guy Even and Moti Medina. Online packet-routing in grids with bounded buffers. In *Proc. 23rd Ann. ACM Symp. on Parallelism in Algorithms and Architectures (SPAA)*, pages 215–224, 2011. doi:10.1145/1989493.1989525.

8 Guy Even and Moti Medina. Online packet-routing in grids with bounded buffers. *CoRR*, abs/1407.4498, 2014.

9 Guy Even, Moti Medina, and Boaz Patt-Shamir. Better deterministic online packet routing on grids. In *Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2015, Portland, OR, USA, June 13-15, 2015*, pages 284–293, 2015. doi:10.1145/2755573.2755588.

10 Jon M Kleinberg. *Approximation algorithms for disjoint paths problems.* PhD thesis, Massachusetts Institute of Technology, 1996.

11 Harald Räcke and Adi Rosén. Approximation algorithms for time-constrained scheduling on line networks. In *SPAA*, pages 337–346, 2009. doi:10.1145/1583991.1584071.

12 Prabhakar Raghavan. Randomized rounding and discrete ham-sandwich theorems: provably good algorithms for routing and packing problems. In *Report UCB/CSD 87/312*. Computer Science Division, University of California Berkeley, 1986.

13 Prabhakar Raghavan and Clark D Tompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987.

14 Adi Rosén and Gabriel Scalosub. Rate vs. buffer size-greedy information gathering on the line. *ACM Transactions on Algorithms*, 7(3):32, 2011. doi:10.1145/1978782.1978787.

15 Neal E Young. Randomized rounding without solving the linear program. In *SODA*, volume 95, pages 170–178, 1995.

[10] Apply the exhaustive search outlined in [11, Lemma 7] for this case.

Distributed Signaling Games*

Moran Feldman[1], **Moshe Tennenholtz**[2], **and Omri Weinstein**[3]

1 **Department of Mathematics and Computer Science, The Open University of Israel, Ra'anana, Israel**
moranfe@openu.ac.il

2 **Faculty of Industrial Engineering and Management, Technion – Israel Institute of Technology, Haifa, Israel**
moshet@ie.technion.ac.il

3 **New York University, New York, USA**
oweinste@cs.princeton.edu

Abstract

The study of the algorithmic and computational complexity of designing efficient signaling schemes for mechanisms aiming to optimize social welfare or revenue is a recurring theme in recent computer science literature. In reality, however, information is typically not held by a central authority, but is distributed among multiple sources (third-party "mediators"), a fact that dramatically changes the strategic and combinatorial nature of the signaling problem.

In this paper we introduce *distributed signaling games*, while using display advertising as a canonical example for introducing this foundational framework. A distributed signaling game may be a pure coordination game (i.e., a distributed optimization task), or a non-cooperative game. In the context of pure coordination games, we show a wide gap between the computational complexity of the centralized and distributed signaling problems, proving that distributed coordination on revenue-optimal signaling is a much harder problem than its "centralized" counterpart.

In the context of non-cooperative games, the outcome generated by the mediators' signals may have different value to each. The reason for that is typically the desire of the auctioneer to align the incentives of the mediators with his own by a compensation relative to the marginal benefit from their signals. We design a mechanism for this problem via a novel application of Shapley's value, and show that it possesses a few interesting economical properties.

1998 ACM Subject Classification C.2.4 [Computer Systems Organization] Computer Communication Networks, Distributed Systems, F.2 [Theory of Computation] Analysis of Algorithms and Problem Complexity, J.4 [Computer Applications] Social and Behavioral Sciences, Economics

Keywords and phrases Signaling, display advertising, mechanism design, shapley value

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.41

1 Introduction

The topic of signaling has recently received much attention in the computer science literature on mechanism design [2, 4, 6, 5, 7, 12]. A recurring theme of this literature is that proper design of a signaling scheme is crucial for obtaining efficient outcomes, such as social welfare maximization or revenue maximization. In reality, however, sources of information are typically not held by a central authority, but are rather distributed among third party mediators/information providers, a fact which dramatically changes the setup to be studied, making it a game between information providers rather than a more classic mechanism

* This work was done while the authors were at Microsoft Research, Israel.

design problem. Such a game is in the spirit of work on the theory of teams in economics [14], whose computational complexity remained largely unexplored. The goal of this paper is to initiate an algorithmic study of such games, which we term *distributed signaling games*, via what we view as a canonical example: Bayesian auctions; and more specifically, display advertising in the presence of third party external mediators (information providers).

Consider a web-site owner that auctions each user's visit to its site, a.k.a. impression. The impression types are assumed to arrive from a commonly known distribution. The bidders are advertisers who know that distribution, but only the web site owner knows the impression type instantiation, consisting of identifiers such as age, origin, gender and salary of the web-site visitor. As is the practice in existing ad exchanges, we assume the auction is a second price auction. The web-site owner decides on the information (i.e., signal) about the instantiation to be provided to the bidders, which then bid their expected valuations for the impression given the information provided. The selection of the proper signaling by the web-site is a central mechanism design problem. Assume, for example, an impression associated with two attributes: whether the user is male or female on one side, and whether he is located in the US or out of the US on the other side. This gives 4 types of possible users. Assume for simplicity that the probability of arrival of each user type is 1/4, and that there are four advertisers each one of them has value of \$100 for a distinguished user type and \$0 for the other types, where these values are common-knowledge. One can verify that an auctioneer who reveals no information receives an expected payoff of \$25, an auctioneer who reveals all information gets no payoff, while partitioning the impression types into two pairs, revealing only the pair of the impression which was materialized (rather than the exact instantiation) will yield a payoff of \$50, which is much higher revenue.

While the above example illustrates some of the potential benefits of signaling and its natural fit to mechanism design, its major drawback is in the unrealistic manner in which information is manipulated: while some information about the auctioned item is typically published by the ad network [18] (such information is modeled here as a public prior), and despite the advertisers' effort to perform "behavioral targeting" by clever data analysis (e.g., utilizing the browsing history of a specific user to infer her interests), the quantity of available contextual information and market expertise is often way beyond the capabilities of both advertisers and auctioneers. This reality gave rise to "third-party" companies which develop technologies for collecting data and online statistics used to infer the contexts of auctioned impressions (see, e.g., [15] and references therein). Consequently, a new *distributed* ecosystem has emerged, in which many third-party companies operate within the market aiming at maximizing their own utility, while significantly increasing the effectiveness of display advertising, as suggested by the following article recently published by Facebook:

> *"Many businesses today work with third parties such as Acxiom, Datalogix, and Epsilon to help manage and understand their marketing efforts. For example, an auto dealer may want to customize an offer to people who are likely to be in the market for a new car. The dealer also might want to send offers, like discounts for service, to customers that have purchased a car from them. To do this, the auto dealer works with a third-party company to identify and reach those customers with the right offer."*
> (www.facebook.com, "Advertising and our Third-Party Partners", April 10, 2013.)

Hence, in reality sources of information are distributed. Typically, the information is distributed among several mediators or information providers/brokers, and is not held (or mostly not held) by a central authority. In the display advertising example, one information source may know the gender and another may know the location of the web-site visitor, while the web-site itself often lacks the capability to track such information. The information sources need to decide on the communicated information. In this case the information sources become players in a game. To make the situation clearer, assume (as above) that the value of each impression type for each bidder is public-knowledge (as is typically the case in repeated interactions through ad exchanges which share their logs with the participants), and the only unknown entity is the instantiation of the impression type; given the information learned from the information sources each bidder will bid his true expected valuation; hence, the results of this game are determined solely by the information providers. Notice that if, in the aforementioned example, the information provider who knows the gender reveals it while the other reveals nothing, then the auctioneer receives a revenue of \$50 as in the centralized case, while the cases in which both information providers reveal their information or none of them do so result in lower revenues. This shows the subtlety of the situation.

The above suggests that a major issue to tackle is the study of *distributed signaling games*, going beyond the realm of classical mechanism design. We use a model of the above display advertising setting, due to its centrality, as a tool to introduce this novel foundational topic. The distributed signaling games may be pure coordination games (a.k.a. distributed optimization), or non-cooperative games. In the context of pure coordination games each information source has the same utility from the output created by their joint signal. Namely, in the above example if the web-site owner pays each information source proportionally to the revenue obtained by the web-site owner then the aims of the information sources are identical. The main aim of the third parties/mediators is to choose their signals based on their privately observed information in a distributed manner in order to optimize their own payoffs. Notice that in a typical embodiment, which we adapt, due to both technical and legal considerations, the auctioneer does not synthesize reported signals into new ones nor the information providers are allowed to explicitly communicate among them about the signals, but can only broadcast information they individually gathered. The study of the computational complexity of this highly fundamental problem is the major technical challenge tackled in this paper. Interestingly, we show a wide gap between the computational complexity of the centralized and distributed setups, proving that coordinating on optimal signaling is a much harder problem than the one discussed in the context of centralized mechanism design. On the other hand we also show a natural restriction on the way information is distributed among information providers, which allows for an efficient constant approximation scheme.

In the context of non-cooperative games the outcome generated by the information sources' reports may result in a different value for each of them. The reason for that is typically the desire of the auctioneer to align the incentives of the mediators with his own by a compensation relative to the marginal benefit from their signals. In the above example one may compare the revenue obtained without the additional information sources, to what is obtained through their help, and compensate relatively to the Shapley values of their contributions, which is a standard (and rigorously justified) tool to fully divide a gain yielded by the cooperation of several parties. Here we apply such division to distributed signaling games, and show that it possesses some interesting properties: in particular the corresponding game has a pure strategy equilibrium, a property of the Shapley value which is shown for the first time for signaling settings (and is vastly different from previous studies of Shapley mechanisms in non-cooperative settings such as cost-sharing games [16]).

1.1 Model

Our model is a generalization of the one defined in [11]. There is a ground set $I = [n]$ of potential items (contexts) to be sold and a set $B = [k]$ of bidders. The value of item j for bidder i is given as v_{ij}. Following the above discussion (and the previous line of work, e.g., [8, 11]), we assume the valuation matrix $V = \{v_{i,j}\}$ is publicly known. An auctioneer is selling a single random item j_R, distributed according to some publicly known prior distribution μ over I, using a second price auction (a more detailed description of the auction follows). There is an additional set $M = [m]$ of "third-party" mediators. Following standard practice in game theoretic information models [1, 9, 7], we assume each mediator $t \in M$ is equipped with a *partition* (signal-set) $\mathcal{P}_t \in \Omega(I)$[1]. Intuitively, $\mathcal{P}_t$ captures the extra information t has about the item which is about to be sold – he knows the set $S \in \mathcal{P}_t$ to which the item j_R belongs, but has no further knowledge about which item of S it is (except for the a priori distribution μ) – in other words, the distribution t has in mind is $\mu|_S$. For example, if the signal-set partition $\mathcal{P}_t$ partitions the items of I into pairs, then mediator t knows to which pair $\{j_1, j_2\} \in \mathcal{P}_t$ the item j_R belongs, but he has no information whether it is j_1 or j_2, and therefore, from her point of view, $\Pr[j_R = j_1] = \mu(j_1)/\mu(\{j_1, j_2\})$.

Mediators can signal some (or all) of the information they own to the network. Formally, this is represented by allowing each mediator t to report *any* super-partition $\mathcal{P}'_t$, which is obtained by merging partitions in her signal-set partition $\mathcal{P}_t$ (in other words $\mathcal{P}_t$ must be a refinement of $\mathcal{P}'_t$). In other words, a mediator may report any partition $\mathcal{P}'_t$ for which there exists a set $\mathcal{Q}'_t \in \Omega(\mathcal{P}_t)$ such that $\mathcal{P}'_t = \{\cup_{S \in A} S \mid A \in \mathcal{Q}'_t\}$. In particular, a mediator can always report $\{I\}$, in which case we say that he remains silent since he does not contribute any information. The signals $\mathcal{P}'_1, \mathcal{P}'_2, \ldots, \mathcal{P}'_m$ reported by the mediators are *broadcasted*[2] to the bidders, inducing a combined partition $\mathcal{P} \triangleq \times_{t=1}^{m} \mathcal{P}'_t = \{\cap_{i \in M} A_i \mid A_i \in \mathcal{P}'_i\}$, which we call the *joint partition* (or *joint signal*). $\mathcal{P}$ splits the auction into separate "restricted" auctions. For each bundle $S \in \mathcal{P}$, the item j_R belongs to S with probability $\mu(S) = \sum_{j \in S} \mu(j)$, in which case S is signaled to the bidders and a second-price auction is performed over $\mu|_S$. Notice that if the signaled bundle is $S \subseteq I$, then the (expected) value of bidder i for $j_R \sim \mu|_S$ is $v_{i,S} = \frac{1}{\mu(S)} \sum_{j \in S} (\mu(j) \cdot v_{ij})$, and the truthfulness of the second price auction implies that this will also be bidder i's bid for the restricted auction. The winner of the auction is the bidder with the maximum bid $\max_{i \in B} v_{i,S}$, and he is charged the second highest valuation for that bundle $\max^{(2)}_{i \in B} v_{i,S}$. Therefore, the auctioneer's revenue with respect to $\mathcal{P}$ is the expectation (over $S \in_R \mathcal{P}$) of the price paid by the winning bidder: $R(\mathcal{P}) = \sum_{S \in \mathcal{P}} [\mu(S) \cdot \max^{(2)}_{i \in B}(v_{i,S})]$.

The joint partition $\mathcal{P}$ signaled by the mediators can dramatically affect the revenue of the auctioneer. Consider, for example, the case where V is the 4×4 identity matrix, μ is the uniform distribution, and M consists of two mediators associated with the partitions $\mathcal{P}_1 = \{\{1, 2\}\ ,\ \{3, 4\}\}$ and $\mathcal{P}_2 = \{\{1, 3\}\ ,\ \{2, 4\}\}$. If both mediators remain silent, the revenue is $R(\{I\}) = 1/4$ (as this is the average value of all 4 bidders for a random item). However, observe that $\mathcal{P}_1 \times \mathcal{P}_2 = \{\{1\}, \{2\}, \{3\}, \{4\}\}$, and the second highest value in every column of V is 0, thus, if both report their partitions, the revenue drops to $R(\mathcal{P}_1 \times \mathcal{P}_2) = 0$. Finally, if mediator 1 reports $\mathcal{P}_1$, while meditor 2 keeps silent, the revenue increases from 1/4 to $R(\mathcal{P}_1) = 1/2$, as the value of each pair of items is 1/2 for two different bidders (thus, the second highest price for each pair is 1/2). This example can be easily generalized to show

[1] For a set S, $\Omega(S) \triangleq \{\mathcal{A} \subseteq 2^S \mid \bigcup_{A \in \mathcal{A}} A = S, \forall_{A,B \in \mathcal{A}} A \cap B = \varnothing\}$ is the collection of all partitions of S.

[2] By saying that a mediator reports $\mathcal{P}'_t$, we mean that he reports the bundle $S \in \mathcal{P}'_t$ for which $j_R \in S$. The reader may wonder why our model is a broadcast model, and does not allow the mediators to report their information to the auctioneer through private channels, in which case the ad network will be able to manipulate and publish whichever information that best serves its interest. The primary reason for the broadcast assumption is that online advertising is a highly dynamic marketplace in which mediators often "come and go", so implementing "private contracts" is infeasible. The second reason is that real-time bidding environments cannot afford the latency incurred by such a two-phase procedure in which the auctioneer first collects the information, and then selectively publishes it. The auction process is usually treated as a "black box", and modifying it harms the modularity of the system.

that in general the intervention of mediators can increase the revenue by a factor of $n/2$!

Indeed, the purpose of this paper is to understand how mediators' (distributed) signals affect the revenue of the auctioneer. We explore the following two aspects of this question:

1. (Computational) Suppose the auctioneer has control over the signals reported by the mediators. We study the computational complexity of the following problem. *Given a $k \times n$ matrix V of valuations and mediators' partitions $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_m$, what is the revenue maximizing joint partition $\mathcal{P} = \mathcal{P}'_1 \times \ldots \times \mathcal{P}'_m$?* We call this problem the *Distributed Signaling Problem*, and denote it by $\mathbf{DSP}(n, k, m)$.
   We note that the problem studied in [11] is a special case of **DSP**, in which there is a *single* mediator ($m = 1$) with *perfect knowledge* about the item sold and can report any desirable signal (partition).[3]
2. (Strategic) What if the auctioneer cannot control the signals reported by the mediators (as the reality of the problem usually entails)? *Can the auctioneer introduce compensations that will incentivize mediators to report signals leading to increased revenue in the auction, when each mediator is acting selfishly?*
   This is a mechanism design problem: Here the auctioneer's goal is to design a payment rule (i.e., a mechanism) for allocating (part of) his profit from the auction among the mediators, based on their reported signals and the auction's outcome, so that global efficiency (i.e., maximum revenue) emerges from their signals.

Section 1.2 summarizes our findings regarding the two above problems.

1.2 Our Results

Ghosh et al. [11] showed that computing the revenue-maximizing signal in their "perfect-knowledge" setup is NP-hard, but present an efficient algorithm for computing a 2-approximation of the optimal signal (partition). We show that when information is distributed, the problem becomes much harder. More specifically, we present a gap-preserving reduction from the *Maximum Independent Set* problem to **DSP**.

▶ **Theorem 1.1** (Hardness of approximating **DSP**). *If there exists an $O(m^{1/2-\varepsilon})$ approximation (for some constant $\varepsilon > 0$) for instances of $\mathbf{DSP}(2m, m+1, m)$, then there exists a $O(N^{1-2\varepsilon})$ approximation for Maximum Independent Set (MIS$_N$), where N is the number of nodes in the underlying graph of the MIS instance.*

Since the Maximum Independent Set problem is NP-hard to approximate to within a factor of $n^{1-\rho}$ for any fixed $\rho > 0$ [13], Theorem 1.1 indicates that approximating the revenue-maximizing signal, even within a multiplicative factor of $O((\min\{n, k, m\})^{1/2-\varepsilon})$, is NP-hard. In other words, one cannot expect a reasonable approximation ratio for $\mathbf{DSP}(n, k, m)$ when the three parameters of the problem are all "large". The next theorem shows that a "small" value for either one of the parameters n or k indeed implies a better approximation ratio.

▶ **Theorem 1.2** (Approximation algorithm for small n or k). *For $k \geq 2$, there is a polynomial time $\min\{n, k-1\}$-approximation algorithm for $\mathbf{DSP}(n, k, m)$.*[4]

We leave open the problem of determining whether one can get an improved approximation ratio when the parameter m is "small". For $m = 1$, the result of [11] implies immediately

[3] In other words, $\mathcal{P}_1$ is the partition of I into singletons.
[4] For $k = 1$, any algorithm is optimal since the use of a second price auction implies that the revenue of any strategy profile is 0 when there is only one bidder.

a 2-approximation algorithm. However, even for the case of $m = 2$ we are unable to find an algorithm having a non-trivial approximation ratio. We mitigate the above results by proving that for a natural (and realistic) class of mediators called *local experts* (defined in Section 3), there exists a polynomial time 5-approximation algorithm for **DSP**.

▶ **Theorem 1.3** (A 5-approximation algorithm for Local Expert mediators)**.** *If mediators are local experts, there exists a polynomial time* 5*-approximation algorithm for* **DSP***.*

In the strategic setup, we design a fair (symmetric) payment rule $\mathcal{S} : (\mathcal{P}'_1, \mathcal{P}'_2, ..., \mathcal{P}'_m) \to \mathbb{R}^m_+$ for incentivizing mediators to report useful information they own, and refrain from reporting information with negative impact on the revenue. This mechanism is inspired by the *Shapley Value* – it distributes part of the auctioneer's surplus among the mediators according to their expected relative marginal contribution to the revenue, when ordered randomly.[5] We first show that this mechanism always admits a *pure* Nash equilibrium, a property we discovered to hold for arbitrary games where the value of the game is distributed among players according to Shapley's value function.

▶ **Theorem 1.4.** *Let $\mathcal{G}_m$ be a non-cooperative m-player game in which the payoff of each player is set according to $\mathcal{S}$. Then $\mathcal{G}_m$ admits a pure Nash equilibrium. Moreover, best response dynamics are guaranteed to converge to such an equilibrium.*

We then turn to analyze the revenue guarantees of our mechanism $\mathcal{S}$. Our first theorem shows that using the mechanism $\mathcal{S}$ never decreases the revenue of the auctioneer compared to the initial state (i.e., when all mediators are silent).

▶ **Theorem 1.5.** *For every Nash equilibrium $(\mathcal{P}'_1, \mathcal{P}'_2, \ldots, \mathcal{P}'_m)$ of $\mathcal{S}$, $R(\times_{t\in M}\mathcal{P}'_t) \geq R(\{I\})$.*

The next two theorems provide tight bounds on the *price of anarchy* and *price of stability* of $\mathcal{S}$.[6] Unlike in the computational setup, even restricting the mediators to be local experts does not enable us to get improved results here.

▶ **Theorem 1.6.** *For $k \geq 2$, the price of anarchy of $\mathcal{S}$ under any instance* $\mathbf{DSP}(n, k, m)$ *is no more than* $\min\{k - 1, n\}$*.*

▶ **Theorem 1.7.** *For every $n \geq 1$, there is a* $\mathbf{DSP}(3n + 1, n + 2, 2)$ *instance for which the price of stability of $\mathcal{S}$ is at least n. Moreover, all the mediators in this instance are local experts.*

Interestingly, an adaptation of Shapley's uniqueness theorem [17] to our non-cooperative setting asserts that the price of anarchy of our mechanism is inevitable if one insists on a few natural requirements – essentially anonymity and efficiency[7] of the payment rule – and assuming the auctioneer alone can introduce payments. We discuss this further in the full version of this paper [10].

[5] Shapley's value was originally introduced in the context of cooperative games, where there is a well defined notion of a coalition's value. In order to apply this notation to a non-cooperative game, we assume the game has some underlying global function ($v(\cdot)$) assigning a value to every strategy profile of the players, and the Shapley value of each player is defined with respect to $v(\cdot)$. In this setting, a "central planner" (the auctioneer in our case) is the one making the utility transfer to the "coalised" players. For the formal axiomatic definition of a value function and Shapley's value function, see [17].

[6] The price of anarchy (stability) is the ratio between the revenue of the optimum and the worst (best) Nash equilibrium.

[7] I.e., the sum of payments is equal to the total surplus of the auctioneer.

1.3 Additional Related Work

The formal study of internet auctions with contexts was introduced by [8] where the authors studied the impact of contexts in the related Sponsored Search model, and showed that *bundling contexts* may have a significant impact on the revenue of the auctioneer. The subsequent work of Ghosh et al. [11] considered the computational algorithmic problem of computing the revenue maximizing partition of items into bundles, under a second price auction in the full information setting. Recently, Emek et al. [7] and Bro Miltersen and Sheffet [2] studied signaling (which generalizes bundling) in the context of display advertising. They explore the computational complexity of computing a signaling scheme that maximizes the auctioneer's revenue in a Bayesian setting. On the other hand, Guo and Deligkas [12] studied a special case of bundling where only "natural" bundles are allowed. Unlike our distributed setup, all the above models are *centralized*, in the sense that the auctioneer has full control over the bundling process (which in our terms corresponds to having a single mediator with a perfect knowledge about the item sold).

A different model with knowledgeable third parties was recently considered by Cavallo et al. [3]. However, the focus of this model is completely different then ours. More specifically, third parties in this model use their information to estimate the clicks-per-impression ratio, and then use this estimate to bridge between advertisers who would like to pay-by-click and ad networks which use a pay-by-impression payment scheme.

2 Preliminaries

Throughout the paper we use capital letters for sets and calligraphic letters for set families. For example, the partition $\mathcal{P}_t$ representing the knowledge of mediator t is a set of sets, and therefore, should indeed be calligraphic according to this notation. A mechanism $\mathcal{M}$ is a tuple of payment functions $(\Pi_1, \Pi_2, \ldots, \Pi_m)$ determining the compensation of every mediator given a strategy profile (i.e., $\Pi_t : \Omega(\mathcal{P}_1) \times \Omega(\mathcal{P}_2) \times \ldots \times \Omega(\mathcal{P}_m) \longrightarrow \mathbb{R}^+$). Every mechanism $\mathcal{M}$ induces the following game between mediators.

▶ **Definition 2.1** (**DSP** game). Given a mechanism $\mathcal{M} = (\Pi_1, \Pi_2, \ldots, \Pi_m)$ and an instance $\mathbf{DSP}(n, k, m)$, the $\mathbf{DSP}_{\mathcal{M}}(n, k, m)$ game is defined as follows. Every mediator $t \in M$ is a player whose strategy space consists of all partitions $\mathcal{P}'_t$ for which $\mathcal{P}_t$ is a refinement. Given a strategy profile $\mathcal{P}'_1, \mathcal{P}'_2, \ldots, \mathcal{P}'_m$, the payoff of mediator t is $\Pi_t(\mathcal{P}'_1, \mathcal{P}'_2, \ldots, \mathcal{P}'_m)$.

Given a **DSP** instance and a set $S \subseteq I$, we use the shorthand $v(S) := \max^{(2)}_{i \in B}(v_{i,S})$ to denote the second highest bid in the restricted auction $\mu|_S$. Using this notation, the expected revenue $R(\mathcal{P})$ of the auctioneer under the (joint) partition $\mathcal{P}$ of the mediators can be restated as $R(\mathcal{P}) = \sum_{S \in \mathcal{P}} \mu(S) \cdot v(S)$.

For a $\mathbf{DSP}_{\mathcal{M}}$ game, let $\mathcal{E}(\mathcal{M})$ denote the set of Nash equilibria of this game and let $\mathcal{P}^*$ be a maximum revenue strategy profile. The *Price of Anarchy* and *Price of Stability* of $\mathbf{DSP}_{\mathcal{M}}$ are defined as:

$$\mathsf{PoA} := \max_{\mathcal{P} \in \mathcal{E}(\mathcal{M})} \frac{R(\mathcal{P}^*)}{R(\mathcal{P})} \ , \qquad \text{and} \qquad \mathsf{PoS} := \min_{\mathcal{P} \in \mathcal{E}(\mathcal{M})} \frac{R(\mathcal{P}^*)}{R(\mathcal{P})} \ ,$$

respectively. Notice that our definition of the price of anarchy and price of stability differs from the standard one by using revenue instead of social welfare.

Paper Organization. The proofs of our results for the computational and strategic setups are given in Sections 3 and 4, respectively. Unfortunately, due to space constraints, many proof are omitted from these sections, and are deferred to the full version of this paper [10]. Section 5 summarizes our contributions and discusses possible avenues for future research.

3 The Computational Complexity of Distributed Signaling (DSP)

This section explores **DSP** from a pure combinatorial optimization viewpoint. In other words, we assume the auctioneer can control the signals produced by each mediator. The objective of the auctioneer is then to choose a distributed strategy profile $\mathcal{P}'_1, \mathcal{P}'_2, \ldots, \mathcal{P}'_m$ whose combination $\times_t \mathcal{P}'_t$ yields maximum revenue in the resulting auction. In light of Theorem 1.1, an efficient algorithm with a reasonable approximation guarantee for general **DSP** is unlikely to exist when the three parameters of the problem are all "large". Subsection 3.1 gives a trivial algorithm which has a good approximation guarantee when either n or k is small. A more interesting result is given in Subsection 3.2, which proves a 5-approximation algorithm for **DSP** under the assumption that the mediators are *local experts*. Due to space constraints, the proof of our negative result (i.e., Theorem 1.1) is omitted from this extended abstract.

3.1 A Simple $\min\{n, k-1\}$-Approximation Algorithm for DSP

In this section we prove the following theorem:

▶ **Theorem 1.2.** *For $k \geq 2$, there is a polynomial time* $\min\{n, k-1\}$*-approximation algorithm for* $\mathbf{DSP}(n, k, m)$.

Proof. We show that the algorithm that simply returns the partition $\{I\}$, the joint partition corresponding to the case where all mediators are silent, has the promised approximation guarantee. For that purpose we analyze the revenue of $\{I\}$ in two different ways:

- Let $\mathcal{P}' = (\mathcal{P}'_1, \mathcal{P}'_2, \ldots, \mathcal{P}'_m)$ be an arbitrary strategy profile of the instance in question. The revenue of $\mathcal{P}'$ is:

$$\begin{aligned} R(\times_{t=1}^m \mathcal{P}'_t) &= \sum_{S \in \times_{t=1}^m \mathcal{P}'_t} \mu(S) \cdot v(S) \leq |\times_{t=1}^m \mathcal{P}'_t| \cdot \max_{S \in \times_{t=1}^m \mathcal{P}'_t} \mu(S) \cdot v(S) \\ &\leq n \cdot \max_{S \in \times_{t=1}^m \mathcal{P}'_t} \mu(S) \cdot v(S) \leq n \cdot R(\{I\}) \ , \end{aligned}$$

  where the last inequality holds since, for every set S, $R(\{I\}) = v(I) \geq v(S) \cdot \mu(S)$. This shows that the approximation ratio of the trivial strategy profile $\{I\}$ provides an n-approximation to the optimal revenue.

- Let $\mathcal{P}' = (\mathcal{P}'_1, \mathcal{P}'_2, \ldots, \mathcal{P}'_m)$ be an arbitrary strategy profile of the instance in question. The revenue of $\mathcal{P}'$ is:

$$\begin{aligned} R(\times_{t=1}^m \mathcal{P}'_t) &= \sum_{S \in \times_{t=1}^m \mathcal{P}'_t} \mu(S) \cdot v(S) = \sum_{S \in \times_{t=1}^m \mathcal{P}'_t} \mu(S) \cdot \left(\max{}^{(2)}_{i \in B} \frac{\sum_{j \in S} \mu(j) \cdot v_{i,j}}{\mu(S)} \right) \\ &= \sum_{S \in \times_{t=1}^m \mathcal{P}'_t} \left(\max{}^{(2)}_{i \in B} \sum_{j \in S} \mu(j) \cdot v_{i,j} \right) \ . \end{aligned}$$

  For every bidder $i \in B$, let $\Sigma_i = \sum_{j \in I} \mu(j) \cdot v_{ij}$. It is easy to see that $v(I) = \max^{(2)}_{i \in B} \Sigma_i$ (in other words, the second highest Σ_i value is $v(I)$). Let $i^* \in B$ be the index maximizing Σ_{i^*} (breaking ties arbitrary). Consider a set $S \in \times_{t=1}^m \mathcal{P}'_t$. The elements of S contribute at least $\max^{(2)}_{i \in B} \sum_{j \in S} \mu(j) \cdot v_{i,j}$ to at least two of the values: $\Sigma_1, \ldots, \Sigma_n$. Thus, they contribute at least the same quantity to the sum $\sum_{i \in B \setminus \{i^*\}} \Sigma_i$. This means that at least one of the values $\{\Sigma_i\}_{i \in B \setminus \{i^*\}}$ must be at least:

$$\frac{\sum_{S \in \times_{t=1}^m \mathcal{P}'_t} \left(\max^{(2)}_{i \in B} \sum_{j \in S} \mu(j) \cdot v_{i,j} \right)}{k-1} = \frac{R(\times_{t=1}^m \mathcal{P}'_t)}{k-1} .$$

By definition Σ_{i^*} must also be at least that large, and therefore, $R(\{I\}) = v(I) \geq R(\times_{t=1}^m \mathcal{P}'_t)/(k-1)$. ◀

3.2 A 5-Approximation Algorithm for Local Expert Mediators

In this subsection we consider an interesting special case of **DSP** which is henceforth shown to admit a constant factor approximation.

▶ **Definition 3.1** (Local Expert mediators). A mediator t in a **DSP** instance is a *local expert* if there exists a set $I_t \subseteq I$ such that: $\mathcal{P}_t = \{\{j\} \mid j \in I_t\} \cup \{I \setminus I_t\}$.

Informally, a local expert mediator has perfect knowledge about a single set I_t – if the item belongs to I_t, he can tell exactly which item it is. In other words, a local expert mediator specializes in some kind of items to the extent that it knows everything about this kind of items, and nothing at all about other kinds of items. Our objective in the rest of the section is to prove Theorem 1.3, i.e., to describe a 5-approximation algorithm for instances of **DSP** consisting of only local expert mediators.

We begin the proof with an upper bound on the revenue of the optimal joint strategy, which we denote by $\mathcal{P}^*$. To describe this bound, we need some notation. We use $\hat{I}$ to denote the set of items that are within the experty field of some mediator (formally, $\hat{I} = \bigcup_{t \in M} I_t$). Additionally, for every item $j \in I$, h_j and s_j denote $\mu(j)$ times the largest value and second largest value, respectively, of j for any bidder (more formally, $h_j = \mu(j) \cdot \max_{i \in B} v_{i,j}$ and $s_j = \mu(j) \cdot \max^{(2)}_{i \in B} v_{i,j}$).

Next, we need to partition the items into multiple sets. The optimal joint partition $\mathcal{P}^*$ is obtained from partitions $\{\mathcal{P}^*_t\}_{t \in M}$, where $\mathcal{P}^*_t$ is a possible partition for mediator t. Each part of $\mathcal{P}^*$ is the intersection of $|M|$ parts, one from each partition in $\{\mathcal{P}^*_t\}_{t \in M}$. On the other hand, each part of $\mathcal{P}^*_t$ is a subset of I_t, except for maybe a single part. Hence, there exists at most a single part $I_0 \in \mathcal{P}^*$ such that $I_0 \not\subseteq I_t$ for any $t \in M$. For ease of notation, if there is no such part (which can happen when $\hat{I} = I$) we denote $I_0 = \varnothing$. To partition the items of $I \setminus I_0$, we associate each part $S \in \mathcal{P}^* \setminus \{I_0\}$ with an arbitrary mediator t such that $S \subseteq I_t$, and denote by A_t the set of items of all the parts associated with mediator t. Observe that the construction of A_t guarantees that $A_t \subseteq I_t$. Additionally, $\{I_0\} \cup \{A_t\}_{t \in M}$ is a disjoint partition of I.

A different partition of the items partitions them according to the bidder that values them the most. In other words, for every $1 \leq i \leq k$, H_i is the set of items for which bidder i has the largest value. If multiple bidders have the same largest value for an item, we assign it to the set H_i of an arbitrary one of these bidders. Notice that the construction of H_i guarantees that the sets $\{H_i\}_{i \in B}$ are disjoint.

Finally, for every set $S \subseteq I$, we use $\phi(S)$ to denote the sum of the $|B| - 1$ smaller values in $\{\sum_{j \in H_i \cap S} h_j\}_{i \in B}$, i.e., the sum of all the values except the largest one. In other words, we calculate for every bidder i the sum of its values for items in $H_i \cap S$, and then add up the $|B| - 1$ smaller sums. Using all the above notation we can now state our promised upper bound on $R(\mathcal{P}^*)$.

▶ **Lemma 3.2.** $R(\mathcal{P}^*) \leq \mu(I_0) \cdot v(I_0) + \sum_{j \in \hat{I}} s_j + \sum_{t \in M} \phi(A_t)$.

Proof. Fix an arbitrary mediator $t \in M$, and let i be the bidder whose term is not counted by $\phi(A_t)$. For every part $S \in \mathcal{P}^*$ associated with t, let i' be a bidder other than i that has one of the two largest bids for S. By definition:

$$\mu(S) \cdot v(S) = \max^{(2)}_{i'' \in B} \sum_{j \in S} \mu(j) \cdot v_{i'',j} \leq \sum_{j \in S} \mu(j) \cdot v_{i',j} \leq \sum_{j \in S \cap H_i} s_j + \sum_{j \in S \setminus H_i} h_j \ .$$

Summing over all parts associated with t, we get:

$$\sum_{\substack{S \in \mathcal{P}^* \\ S \subseteq A_t}} \mu(S) \cdot v(S) \leq \sum_{j \in A_t \cap H_i} s_j + \sum_{j \in A_t \setminus H_i} h_j \leq \sum_{j \in A_t} s_j + \phi(A_t) \ .$$

Summing over all mediators, we get:

$$R(\mathcal{P}^*) - \mu(I_0) \cdot v(I_0) \leq \sum_{t \in M} \left(\sum_{j \in A_t} s_j + \phi(A_t) \right) \leq \sum_{j \in \hat{I}} s_j + \sum_{t \in M} \phi(A_t) \ . \qquad ◀$$

Our next step is to describe joint partitions that can be found efficiently and upper bound the different terms in the bound given by Lemma 3.2 (up to a constant factor). Finding such partitions for the first two terms is quite straightforward.

▶ **Observation 3.3.** *The joint partitions where all mediators are silent $\{I\} = \times_{i \in B}\{I\}$ obeys: $R(\{I\}) \geq \mu(I_0) \cdot v(I_0)$.*

Proof.

$$R(\{I\}) = \max^{(2)}_{i \in B} \left(\sum_{j \in I} \mu(j) \cdot v_{i,j} \right) \geq \max^{(2)}_{i \in B} \left(\sum_{j \in I_0} \mu(j) \cdot v_{i,j} \right) = \mu(I_0) \cdot v(I_0) \ . \qquad ◀$$

▶ **Observation 3.4.** *The joint partitions $\mathcal{P}_S = \times_{t \in M} \mathcal{P}_t$ where every mediator reports all his information obeys:*

$$R(\mathcal{P}_S) = R(\{\{j\}_{j \in \hat{I}}\} \cup \{I \setminus \hat{I}\}) \geq \sum_{j \in \hat{I}} \mu(j) \cdot \max^{(2)}_{i \in B} v_{i,j} = \sum_{j \in \hat{I}} s_j \ .$$

It remains to find a joint partition that upper bounds, up to a constant factor, the third term in the bound given by Lemma 3.2. If one knows the sets $\{A_t\}_{t \in M}$, then one can easily get such a partition using the method of Ghosh et al. [11]. In this method, one partitions every set A_t into the parts $\{A_t \cap H_i\}_{i=1}^t$ and sort these parts according to the value of $\sum_{j \in A_t \cap H_i} h_j$. Then, with probability $1/2$ every even part is united with the part that appears after it in the above order, and with probability $1/2$ it is united with the part that appears before it in this order. It is not difficult to verify that if the part of bidder i is not the first in the order, then with probability $1/2$ it is unified with the part that appears before it in the order, and then it contributes $\sum_{j \in A_t \cap H_i} h_j$ to the revenue. Hence, the expected contribution to the revenue of the parts produced from A_t is at least $1/2 \cdot \phi(A_t)$.

Algorithm 1 can find a partition that is competitive against $\sum_{t \in M} \phi(A_t)$ without knowing the sets $\{A_t\}_{t \in M}$. The algorithm uses the notation of a *cover*. We say that a set S_j is a cover of an element $j \in I_t \cap H_i$ if $S_j \subseteq I_t \cap H_{i'}$ for some $i \neq i'$.

Notice that the definition of cover guarantees that a part containing both j and S_j contributes to the revenue at least $\min\{h_j, \sum_{j' \in S_j h_{j'}}\}$. Using this observation, each iteration of Algorithm 1 can be viewed as trying to extract revenue from element j. Additionally, observe that the partition $\mathcal{P}$ produced by Algorithm 1 can be presented as a joint partition since every part in it, except for $I \setminus \hat{I}$, contains only items that belong to a single set I_t (for some mediator $t \in M$).

Algorithm 1: Local Experts - Auxiliary Algorithm

```
1 Let $I' \leftarrow \hat{I}$ and $\mathcal{P} \leftarrow \{I \setminus \hat{I}\}$.
2 while $I' \neq \varnothing$ do
3     Let $j$ be the element maximizing $h_j$ in $I'$.
4     Find a cover $S_j \subseteq I'$ of $j$ obeying $h_j \leq \sum_{j' \in S_j} h_{j'} \leq 2h_j$, or maximizing $\sum_{j' \in S_j} h_{j'}$
       if no cover of $j$ makes this expression at least $h_j$.
5     Add the part $S_j \cup \{j\}$ to $\mathcal{P}$, and remove the elements of $S_j \cup \{j\}$ from $I'$.
6 return $\mathcal{P}$
```

▶ **Observation 3.5.** *Algorithm 1 can be implemented in polynomial time.*

Proof. One can find a cover S_j maximizing $\sum_{j' \in S_j} h_{j'}$ in line 4 of the algorithm by considering the set $I_t \cap H_{i'} \cap I'$ for every mediators t and bidder i' obeying $j \in I_t$ and $j \notin H_{i'}$. Moreover, if this cover is of size larger than $2h_j$, then by removing elements from this cover one by one the algorithm must find a cover S'_j obeying $h_j \leq \sum_{j' \in S'_j} h_{j'} \leq 2h_j$ because j is the element maximizing h_j in I'. ◀

The following lemma relates the revenue of the set produced by Algorithm 1 to the sum $\sum_{t \in M} \phi(A_t)$.

▶ **Lemma 3.6.** *No iteration of the loop of Algorithm 1 decreases the value of the expression* $R(\mathcal{P}) + 1/3 \cdot \sum_{t \in M} \phi(A_t \cap I')$.[8]

Proof. Fix an arbitrary iteration. There are two cases to consider. First, assume $h_j \leq \sum_{j' \in S_j} h_{j'} \leq 2h_j$. In this case the increase in $R(\mathcal{P})$ during this iteration is:

$$\mu(S_j \cup \{j\}) \cdot v(S_j \cup \{j\}) \geq \min\left\{h_j, \sum_{j' \in S_j} h_{j'}\right\} = h_j \quad .$$

On the other hand, one can observe that, when removing an element j' from S, the value of $\phi(S)$ can decrease by at most $h_{j'}$. Hence, the decrease in $\sum_{t \in M} \phi(A_t \cap I')$ during this iteration can be upper bounded by: $h_j + \sum_{j' \in S_j} h_{j'} \leq 3h_j$.

Consider now the case $\sum_{j' \in S_j} h_{j'} < h_j$. In this case the increase in $R(\mathcal{P})$ during the iteration is:

$$\mu(S_j \cup \{j\}) \cdot v(S_j \cup \{j\}) \geq \min\left\{h_j, \sum_{j' \in S_j} h_{j'}\right\} = \sum_{j' \in S_j} h_{j'} \quad .$$

If j does not belong to A_t for any mediator t, then by the above argument we can bound the decrease in $\sum_{t \in M} \phi(A_t \cap I')$ by $\sum_{j' \in S_j} h_{j'}$. Hence, assume from now on that there exists a mediator t' and a bidder i such that $j \in A_{t'} \cap H_i$. Let $i' \neq i$ be a bidder maximizing $\sum_{j' \in H_{i'} \cap A_{t'} \cap I'} h_{j'}$. Clearly, the removal of a single element from I' can decrease $\phi(A_{t'} \cap I')$ by no more than $\sum_{j' \in H_{i'} \cap A_{t'} \cap I'} h_{j'}$. Hence, the decrease in $\sum_{t \in M} \phi(A_t \cap I')$ during the iteration of the algorithm can be upper bounded by:

$$\sum_{j' \in H_{i'} \cap A_{t'} \cap I'} h_{j'} + \sum_{j' \in S_j} h_{j'} \quad .$$

[8] Before the algorithm terminates $\mathcal{P}$ is a partial partition in the sense that some items do not belong to any part in it. However, the definition of $R(\mathcal{P})$ naturally extends to such partial partitions.

On the other hand, $H_{i'} \cap A_{t'} \cap I'$ is a possible cover for j, and thus, by the optimality of S_j:

$$\sum_{j' \in H_{i'} \cap A_{t'} \cap I'} h_{j'} \leq \sum_{j' \in S_j} h_{j'} \ . \qquad \blacktriangleleft$$

▶ **Corollary 3.7.** *$R(\mathcal{P}_A) \geq 1/3 \cdot \sum_{t \in M} \phi(A_t)$, where $\mathcal{P}_A$ is the partition produced by Algorithm 1.*

Proof. After the initialization step of Algorithm 1 we have:

$$R(\mathcal{P}) + 1/3 \cdot \sum_{t \in M} \phi(A_t \cap I') \geq 1/3 \cdot \sum_{t \in M} \phi(A_t) \ .$$

On the other hand, when the algorithm terminates:

$$R(\mathcal{P}) + 1/3 \cdot \sum_{t \in M} \phi(A_t \cap I') = R(\mathcal{P}_A)$$

because $I' = \varnothing$. The corollary now follows from Lemma 3.6. ◀

We are now ready to prove Theorem 1.3.

▶ **Theorem 1.3.** *If mediators are local experts, there exists a polynomial time 5-approximation algorithm for* **DSP**.

Proof. Consider an algorithm that outputs the best solution out of $\{I\}$, $\mathcal{P}_S$ and $\mathcal{P}_A$. The following inequality shows that at least one of these joint partitions has a revenue of $R(\mathcal{P}^*)/5$:

$$R(\{I\}) + R(\mathcal{P}_S) + 3R(\mathcal{P}_A) \geq \sum_{j \in \hat{I}} s_j + \mu(I_0) \cdot v(I_0) + \sum_{t \in M} \phi(A_t) \geq R(\mathcal{P}^*) \ ,$$

where the first inequality holds by Observations 3.3 and 3.4 and Corollary 3.7; and the second inequality uses the upper bound on $R(\mathcal{P}^*)$ proved by Lemma 3.2. ◀

4 The Strategic Problem

This section explores the **DSP** problem from a strategic viewpoint, in which the auctioneer *cannot* control the signals produced by each mediator, and is, therefore, trying to solicit information from the mediators that would yield a maximal revenue in the auction. In other words, the objective of the auctioneer is to design a mechanism $\mathcal{M}$ whose equilibria (i.e., the signals $\mathcal{P}'_1, \mathcal{P}'_2, \ldots, \mathcal{P}'_m$ which are now chosen strategically by the mediators) induce maximum revenue. Due to space constraints we are only able to present in this extended abstract only a few of our contributions for the strategic settings. Namely, we introduce the *Shapley mechanism* $\mathcal{S}$ and prove some interesting properties of it (Theorems 1.4 and 1.5).

Our mechanism $\mathcal{S}$ aims to incentivize mediators to report useful information, with the hope that global efficiency emerges despite selfish behavior of each mediator. For the sake of generality, we describe $\mathcal{S}$ for a game generalizing **DSP**. Consider a game $\mathcal{G}_m$ of m players where each player t has a finite set A_t of possible strategies, one of which $\varnothing_t \in A_t$ is called the null strategy of t. The value of a strategy profile in the game $\mathcal{G}_m$ is determined by a value function $v : A_1 \times A_2 \times \ldots \times A_m \to \mathbb{R}$. A mechanism $M = (\Pi_1, \Pi_2, \ldots, \Pi_m)$ for $\mathcal{G}_m$ is a set of payments rules. In other words, if the players choose strategies $a_1 \in A_1, a_2 \in A_2, \ldots, a_m \in A_m$, then the payment to player t under mechanism M is $\Pi_t(v, a_1, a_2, \ldots, a_m)$. Notice that **DSP** fits the definition of $\mathcal{G}_m$ when $A_t = \Omega(\mathcal{P}_t)$ is the set of partitions that t can report for every mediator t, and $\varnothing_t$ is the silence strategy $\{I\}$. The appropriate value function v for **DSP** is the function $R(\times_{t=1}^{m}\mathcal{P}'_t)$, where $\mathcal{P}'_t \in A_t$ is the strategy of mediator t. In other words, the value function v of a **DSP** game is equal to the revenue of the auctioneer.

Given a strategy profile $a = (a_1, a_2, \ldots, a_m)$, and subset $J \in [m]$ of players, we write a_J to denote a strategy profiles where the players of J play their strategy in a, and the other players play their null strategies. We abuse notation and denote by $\varnothing$ the strategy profile $a_\varnothing$ where all players play their null strategies. Additionally, we write (a'_t, a_{-t}) to denote a strategy profile where player t plays a'_t and the rest of the players follow the strategy profile a. The mechanism $\mathcal{S}$ we propose distributes the increase in the value of the game (compared to $v(\varnothing)$) among the players according to their Shapley value: it pays each player his expected marginal contribution to the value according to a uniformly random ordering of the m player. Formally, the payoff for player t given a strategy profile a is

$$\Pi_t(a) = \frac{1}{m!} \cdot \sum_{\sigma \in S_m} \left[v\left(a_{\{\sigma^{-1}(j) | 1 \le j \le \sigma(t)\}}\right) - v\left(a_{\{\sigma^{-1}(j) | 1 \le j < \sigma(t)\}}\right) \right] \quad , \tag{1}$$

which can alternatively be written as

$$\Pi_t(a) = \sum_{J \subseteq [m] \setminus \{t\}} \gamma_J \left(v(a_{J \cup \{t\}}) - v(a_J) \right) \quad , \tag{2}$$

where $\gamma_J = \frac{|J|!(m-|J|-1)!}{m!}$ is the probability that the players of J appear before player t when the players are ordered according to a uniformly random permutation $\sigma \in_R S_m$. We use both definitions (1) and (2) interchangeably, as each one is more convenient in some cases than the other. We remark that the above payoffs can be implemented efficiently.[9]

Clearly, the mechanism $\mathcal{S}$ is anonymous (symmetric). The main feature of the Shapley mechanism is that it is efficient. In other words, the sum of the payoffs is exactly equal to the total increase in value (in the case of **DSP**, the surplus revenue of the auctioneer compared to the initial state).[10]

▶ **Proposition 4.1** (Efficiency property). *For every strategy profile $a = (a_1, a_2, \ldots, a_m)$, $v(a) - v(\varnothing) = \sum_{t=1}^{m} \Pi_t(a)$.*

Proof. Recall that the payoff of mediator t is:

$$\frac{1}{m!} \cdot \sum_{\sigma \in S_m} \left[v\left(a_{\{\sigma^{-1}(j) | 1 \le j \le \sigma(t)\}}\right) - v\left(a_{\{\sigma^{-1}(j) | 1 \le j < \sigma(t)\}}\right) \right] \quad .$$

Summing over all mediators, we get:

$$\sum_{t=1}^{m} \Pi_t(\mathcal{P}'_t, \mathcal{P}'_{-t}) = \sum_{t=1}^{m} \left\{ \frac{1}{m!} \cdot \sum_{\sigma \in S_m} \left[v\left(a_{\{\sigma^{-1}(j) | 1 \le j \le \sigma(t)\}}\right) - v\left(a_{\{\sigma^{-1}(j) | 1 \le j < \sigma(t)\}}\right) \right] \right\}$$

[9] Assuming value queries, we can calculate a payoff for every player by drawing a random permutation σ and paying $v\left(a_{\{\sigma^{-1}(j) | 1 \le j \le \sigma(t)\}}\right) - v\left(a_{\{\sigma^{-1}(j) | 1 \le j < \sigma(t)\}}\right)$ for each mediator t. Clearly this procedure produce the payoffs of our mechanism *in expectation*. Alternatively, the *expected* payoff of each player can be approximated using sampling.

[10] One natural alternative for the Shapley mechanism is a VCG-based mechanism. The main disadvantage of this alternative mechanism is that it is not necessarily efficient. In fact, one can easily design instances where a VCG-based mechanism induces a total payoff which is significantly larger than the increase in the value.

$$= \frac{1}{m!} \cdot \sum_{\sigma \in S_m} \sum_{t=1}^{m} \left[v\left(a_{\{\sigma^{-1}(j) | 1 \leq j \leq \sigma(t)\}}\right) - v\left(a_{\{\sigma^{-1}(j) | 1 \leq j < \sigma(t)\}}\right) \right]$$

$$= \frac{1}{m!} \cdot \sum_{\sigma \in S_m} \left[v\left(a_{\{\sigma^{-1}(j) | 1 \leq j \leq m\}}\right) - v\left(a_{\varnothing}\right) \right] = v(a) - v(\varnothing) \ . \qquad ◀$$

Proposition 4.1 implies the following theorem. Notice that Theorem 1.5 is in fact a restriction of this theorem to the game $\mathbf{DSP}_{\mathcal{S}}$.

▶ **Theorem 4.2.** *For every Nash equilibrium a, $v(a) \geq v(\varnothing)$.*

Proof. A player always has the option of playing his null strategy, which results in a zero payoff for him. Thus, the payoff of a player in a Nash equilibrium can never be negative. Hence, by Proposition 4.1: $v(a) \geq v(\varnothing) + \sum_{i=1}^{m} \Pi_t(a) \geq v(\varnothing)$. ◀

Next, let us prove Theorem 1.4. For convenience, we restate it below.

▶ **Theorem 1.4.** *Let $\mathcal{G}_m$ be a non-cooperative m-player game in which the payoff of each player is set according to $\mathcal{S}$. Then $\mathcal{G}_m$ admits a pure Nash equilibrium. Moreover, best response dynamics are guaranteed to converge to such an equilibrium.*

Proof. We prove the theorem by showing that $\mathcal{G}_m$ is an exact potential game, which in turn implies all the conclusions of the theorem. Recall that an exact potential game is a game for which there exists a potential function $\Phi\colon A_1 \times A_2 \times \cdots \times A_t \to \mathbb{R}$ such that every strategy profile a and possible deviation $a'_t \in A_t$ of a player t obey:

$$\Pi_t(a'_t, a_{-t}) - \Pi_t(a) = \Phi(a'_t, a_{-t}) - \Phi(a) \ . \tag{3}$$

In our case the potential function is $\Phi(a) = \sum_{J \subseteq [m]} \beta_J \cdot v(a_J)$, where $\beta_J = \frac{(|J|-1)!(m-|J|)!}{m!}$. Let us prove that this function obeys (3). It is useful to denote by a' the strategy profile (a'_t, a_{-t}). By definition:

$$\Pi_t(a') - \Pi_t(a) = \sum_{J \subseteq [m] \setminus \{i\}} \gamma_J \left[v(a_{J \cup \{i\}}) - v(a_J) \right] - \sum_{J \subseteq [m] \setminus \{i\}} \gamma_J \left[v(a'_{J \cup \{i\}}) - v(a'_J) \right] \ . \tag{4}$$

For $J \subseteq [m] \setminus \{i\}$, we have $a_J = a'_J$. Plugging this observation into (4), and rearranging, we get:

$$\Pi_t(a') - \Pi_t(a) = \sum_{J \subseteq [m] \setminus \{i\}} \gamma_J \left[v(a_{J \cup \{i\}}) - v(a'_{J \cup \{i\}}) \right] \ . \tag{5}$$

For every J containing i we get: $\alpha_{J \setminus \{i\}} = \beta_J$. Using this observation and the previous observation that $a_J = a'_J$ for $J \subseteq [m] \setminus \{i\}$, (5) can be replaced by:

$$\Pi_t(a') - \Pi_t(a) = \sum_{J \subseteq [m]} \beta_J (v(a'_J) - v(a_J)) = \Phi(a') - \Phi(a) \ . \qquad ◀$$

Before concluding this section, a few remarks regarding the use of $\mathcal{S}$ to **DSP** are in order:

1. The reader may wonder why the auctioneer cannot impose on the mediators any desired outcome $\times_{t \in M} \mathcal{P}'_t$ by offering mediator t a negligible payment if he signals $\mathcal{P}'_t$, and no payment otherwise. However, implementing such a mechanism requires the auctioneer to know the information sets $\mathcal{P}_t$ of each mediator *in advance.* In contrast, our mechanism requires access only to the outputs of the mediators.

2. Proposition 4.1 implies that the auctioneer distributes the entire surplus among the mediators, which seems to defeat the purpose of the mechanism. However, in the target application she can scale the revenue by a factor $\alpha \in (0, 1]$ and only distribute the corresponding fraction of the surplus. As all of our results are invariant under scaling, this trick can be applied in a black box fashion. Thus, we assume, without loss of generality, $\alpha = 1$.
3. We assume mediators never report a signal which is inconsistent with the true identity of the sold element j_R. The main justification for this assumption is that the mediators' signals must be consistent with one another (as they refer to a single element j_R). Thus, given that sufficiently many mediators are honest, "cheaters" can be easily detected.
4. Note that for a particular ordering of the mediators $\sigma \in \mathcal{S}_m$ and a particular joint strategy profile, the marginal payoff of a mediator may be negative (if she is out of luck and contributes negatively to the revenue according to σ). However, we stress that the expected value (over σ) of each mediator is never negative in any equilibrium strategy (by Theorem 4.2). Since in realistic applications the process is assumed to be repeated over time, the probability that a mediator has overall negative payoff is negligible.

5 Discussion

In this paper we have considered computational and strategic aspects of auctions involving third party information mediators. Our main result for the computational point of view shows that it is NP-hard to get a reasonable approximation ratio when the three parameters of the problem are all "large". For the parameters n and k this is tight in the sense that there exists an algorithm whose approximation ratio is good when either one of these parameters is "small". However, we do not know whether a small value for the parameter m allows for a good approximation ratio. More specifically, even understanding the approximation ratio achievable in the case $m = 2$ is an interesting open problem. Observe that the case $m = 2$ already captures (asymptotically) the largest possible price of stability and price of anarchy in the strategic setup,[11] and thus, it is tempting to assume that this case also captures all the complexity of the computational setup.

Unfortunately, most of our results, for both the computational and strategic setups, are quite negative. The class of local experts we describe is a natural mediators class allowing us to bypass one of these negative result and get a constant approximation ratio algorithm for the computational setup. An intriguing potential avenue for future research is finding additional natural classes of mediators that allow for improved results, either under the computational or the strategic setup.

Another possible direction for future research is to study an extension of our distributed setup where bundling is replaced with randomized signaling (similarly to the works of [2] and [7] which introduced randomized signaling into the centralized model of [11]). In the centralized model it turned out that finding the optimal randomized signaling is easier then finding the optimal bundling [2, 7], which is counterintuitive since randomized signaling generalize bundling. Hence, one can hope that randomized signaling might also mitigate some of our inapproximability results.

[11] By Theorem 1.7 the price of stability can be as large as $O(\min\{k, n\})$ even for two mediators, and Theorem 1.6 shows that the price of anarchy cannot be larger than that for any number of mediators.

References

1 Robert J. Aumann. Agreeing to Disagree. *The Annals of Statistics*, 4(6):1236–1239, 1976.

2 Peter Bro Miltersen and Or Sheffet. Send mixed signals: Earn more, work less. In *EC*, pages 234–247, New York, NY, USA, 2012. ACM. doi:10.1145/2229012.2229033.

3 Ruggiero Cavallo, R. Preston McAfee, and Sergei Vassilvitskii. Display advertising auctions with arbitrage. *ACM Trans. Economics and Comput.*, 3(3):15, 2015. doi:10.1145/2668033.

4 Yu Cheng, Ho Yee Cheung, Shaddin Dughmi, and Shang-Hua Teng. Signaling in quasipolynomial time. *CoRR*, abs/1410.3033, 2014. URL: http://arxiv.org/abs/1410.3033.

5 Shaddin Dughmi. On the hardness of signaling. In *55th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 354–363, 2014. doi:10.1109/FOCS.2014.45.

6 Shaddin Dughmi, Nicole Immorlica, and Aaron Roth. Constrained signaling in auction design. In *Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1341–1357, 2014. doi:10.1137/1.9781611973402.99.

7 Yuval Emek, Michal Feldman, Iftah Gamzu, Renato Paes Leme, and Moshe Tennenholtz. Signaling schemes for revenue maximization. *ACM Trans. Economics and Comput.*, 2(2):5, 2014. doi:10.1145/2594564.

8 Eyal Even-Dar, Michael J. Kearns, and Jennifer Wortman. Sponsored search with contexts. In *Internet and Network Economics, Third International Workshop (WINE)*, pages 312–317, 2007. doi:10.1007/978-3-540-77105-0_32.

9 Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Vardi. *Reasoning About Knowledge.* MIT Press, 1995.

10 Moran Feldman, Moshe Tennenholtz, and Omri Weinstein. Distributed signaling games. *CoRR*, abs/1404.2861v2, 2015. URL: http://arxiv.org/abs/1404.2861v2.

11 Arpita Ghosh, Hamid Nazerzadeh, and Mukund Sundararajan. Computing optimal bundles for sponsored search. In *Internet and Network Economics, Third International Workshop (WINE)*, pages 576–583, 2007. doi:10.1007/978-3-540-77105-0_63.

12 Mingyu Guo and Argyrios Deligkas. Revenue maximization via hiding item attributes. In *IJCAI*, 2013. URL: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6909.

13 Johan Håstad. Clique is hard to approximate within $1-\varepsilon$. *Acta Mathematica*, 182(1):105–142, 1999. doi:10.1007/BF02392825.

14 Jacob Marschak and Roy Radner. *Economic theory of teams.* Cowles foundation for research in economics at Yale University. Yale University Press, New Haven and London, 1972.

15 Jonathan R. Mayer and John C. Mitchell. Third-party web tracking: Policy and technology. In *IEEE Symposium on Security and Privacy (SP)*, pages 413–427, 2012. doi:10.1109/SP.2012.47.

16 Tim Roughgarden and Mukund Sundararajan. New trade-offs in cost-sharing mechanisms. In *The Thirty-Eighth Annual ACM Symposium on Theory of Computing (STOC)*, pages 79–88, 2006. doi:10.1145/1132516.1132528.

17 L. S. Shapley. A value for n-person games. *Contributions to the theory of games*, 2:307–317, 1953.

18 Shuai Yuan, Jun Wang, and Xiaoxue Zhao. Real-time bidding for online advertising: Measurement and analysis. In *The Seventh International Workshop on Data Mining for Online Advertising (ADKDD)*, pages 3:1–3:8, 2013.

New Algorithms for Maximum Disjoint Paths Based on Tree-Likeness*

Krzysztof Fleszar[1], Matthias Mnich[2], and Joachim Spoerhase[3]

1 **Universität Würzburg, Würzburg, Germany**
krzysztof.fleszar@uni-wuerzburg.de
2 **Universität Bonn, Bonn, Germany**
mmnich@uni-bonn.de
3 **Universität Würzburg, Würzburg, Germany**
joachim.spoerhase@uni-wuerzburg.de

Abstract

We study the classical NP-hard problems of finding maximum-size subsets from given sets of k terminal pairs that can be routed via edge-disjoint paths (MaxEDP) or node-disjoint paths (MaxNDP) in a given graph. The approximability of MaxEDP/NDP is currently not well understood; the best known lower bound is $\Omega(\log^{1/2-\varepsilon} n)$, assuming $\mathsf{NP} \not\subseteq \mathsf{ZPTIME}(n^{\mathrm{poly}\log n})$. This constitutes a significant gap to the best known approximation upper bound of $\mathcal{O}(\sqrt{n})$ due to Chekuri et al. (2006) and closing this gap is currently one of the big open problems in approximation algorithms. In their seminal paper, Raghavan and Thompson (Combinatorica, 1987) introduce the technique of randomized rounding for LPs; their technique gives an $\mathcal{O}(1)$-approximation when edges (or nodes) may be used by $\mathcal{O}\left(\frac{\log n}{\log\log n}\right)$ paths.

In this paper, we strengthen the above fundamental results. We provide new bounds formulated in terms of the *feedback vertex set number* r of a graph, which measures its vertex deletion distance to a forest. In particular, we obtain the following.

- For MaxEDP, we give an $\mathcal{O}(\sqrt{r}\cdot\log^{1.5} kr)$-approximation algorithm. As $r \leq n$, up to logarithmic factors, our result strengthens the best known ratio $\mathcal{O}(\sqrt{n})$ due to Chekuri et al.
- Further, we show how to route $\Omega(\mathrm{OPT})$ pairs with congestion $\mathcal{O}\left(\frac{\log kr}{\log\log kr}\right)$, strengthening the bound obtained by the classic approach of Raghavan and Thompson.
- For MaxNDP, we give an algorithm that gives the optimal answer in time $(k+r)^{\mathcal{O}(r)}\cdot n$. This is a substantial improvement on the run time of $2^k r^{\mathcal{O}(r)}\cdot n$, which can be obtained via an algorithm by Scheffler.

We complement these positive results by proving that MaxEDP is NP-hard even for $r=1$, and MaxNDP is W[1]-hard for parameter r. This shows that neither problem is fixed-parameter tractable in r unless $\mathsf{FPT} = \mathsf{W}[1]$ and that our approximability results are relevant even for very small constant values of r.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases disjoint paths, approximation algorithms, feedback vertex set

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.42

1 Introduction

In this paper, we study disjoint paths routing problems. In this setting, we are given an undirected graph G and a collection of source-destination pairs $\mathcal{M} = \{(s_1,t_1),\ldots,(s_k,t_k)\}$.

* This research was supported by ERC Starting Grant 306465 (BeyondWorstCase).

The goal is to select a maximum-sized subset $\mathcal{M}' \subseteq \mathcal{M}$ of the pairs that can be *routed*, where a routing of $\mathcal{M}'$ is a collection $\mathcal{P}$ of paths such that, for each pair $(s_i, t_i) \in \mathcal{M}'$, there is a path in $\mathcal{P}$ connecting s_i to t_i. In the Maximum Edge Disjoint Paths (MaxEDP) problem, a routing $\mathcal{P}$ is *feasible* if its paths are pairwise edge-disjoint, and in the Maximum Node Disjoint Paths (MaxNDP) problem the paths in $\mathcal{P}$ must be pairwise vertex-disjoint.

Disjoint paths problems are fundamental problems with a long history and significant connections to optimization and structural graph theory. The decision versions EDP of MaxEDP and NDP of MaxNDP ask whether all of the pairs can be routed. Karp [27] showed that, when the number of pairs is part of the input, the decision problem is NP-complete. In undirected graphs, MaxEDP and MaxNDP are solvable in polynomial time when the number of pairs is a fixed constant; this is a very deep result of Robertson and Seymour [40] that builds on several fundamental results in structural graph theory from their graph minors project.

In this paper, we consider the optimization problems MaxEDP and MaxNDP when the number of pairs are part of the input. In this setting, the best approximation ratio for MaxEDP is achieved by an $\mathcal{O}(\sqrt{n})$-approximation algorithm [12, 33], where n is the number of nodes, whereas the best hardness for undirected graphs is only $\Omega(\log^{1/2-\varepsilon} n)$ [3]. Bridging this gap is a fundamental open problem that seems quite challenging at the moment.

Most of the results for routing on disjoint paths use a natural multi-commodity flow relaxation as a starting point. A well-known integrality gap instance due to Garg et al. [24] shows that this relaxation has an integrality gap of $\Omega(\sqrt{n})$, and this is the main obstacle for improving the $\mathcal{O}(\sqrt{n})$-approximation ratio in general graphs. The integrality instance on an $n \times n$ grid (of treewidth $\Theta(\sqrt{n})$) exploits a topological obstruction in the plane that prevents a large integral routing; see Fig. 1. This led Chekuri et al. [15] to studying the approximability of MaxEDP with respect to the *tree-width* of the underlying graph. In particular, they pose the following conjecture:

▶ **Conjecture 1** ([13]). *The integrality gap of the standard multi-commodity flow relaxation for* MaxEDP *is $\Theta(w)$, where w is the treewidth of the graph.*

Recently, Ene et al. [21] showed that MaxEDP admits an $\mathcal{O}(w^3)$-approximation algorithm on graphs of treewidth at most w. Theirs is the best known approximation ratio in terms of w, improving on an earlier $\mathcal{O}(w \cdot 3^w)$-approximation algorithm due to Chekuri et al. This shows that the problem seems more amenable on "tree-like" graphs.

However, for $w = \omega(n^{1/6})$, the bound is weaker than the bound of $\mathcal{O}(\sqrt{n})$. In fact, EDP remains NP-hard even for graphs of *constant* treewidth, namely treewidth $w = 2$ [37]. This further rules out the existence of a fixed-parameter algorithm for MaxEDP parameterized by w, assuming P $\neq$ NP. Therefore, to obtain fixed-parameter tractability results as well as better approximation guarantees, one needs to resort to parameters stronger than treewidth.

Another route to bridge the large gap between approximation lower and upper bounds for MaxEDP is to allow the paths to have *low congestion* c: that is, instead of requiring the routed paths to be pairwise disjoint, at most c paths can use an edge. In their groundbreaking work, Raghavan and Thompson [38] introduced the technique of randomized rounding of LPs to obtain polynomial-time approximation algorithms for combinatorial problems. Their approach allows to route $\Omega(\text{OPT})$ pairs of paths with congestion $\mathcal{O}\left(\frac{\log n}{\log\log n}\right)$. This extensive line of research [2, 18, 28] has culminated in a $\log^{\mathcal{O}(1)} k$-approximation algorithm with congestion 2 for MaxEDP [20]. A slightly weaker result also holds for MaxNDP [11].

1.1 Motivation and Contribution

The goal of this work is to study disjoint paths problems under another natural measure for how "far" a graph is from being a tree. In particular, we propose to examine MaxEDP and MaxNDP under the *feedback vertex set number*, which for a graph G denotes the smallest size r of a set R of G for which $G-R$ is a forest. Note that the treewidth of G is at most $r+1$. Therefore, given the NP-hardness of EDP for $w=2$ and the current gap between the best known upper bound $\mathcal{O}(w^3)$ and the linear upper bound suggested by Conjecture 1, it is interesting to study the stronger restriction of bounding the feedback vertex set number r of the input graph. Our approach is further motivated by the fact that MaxEDP is efficiently solvable on trees by means of the algorithm of Garg, Vazirani and Yannakakis [24]. Similarly, MaxNDP is easy on trees (see Theorem 4).

Our main insight is that one can in fact obtain bounds in terms of r that either strengthen the best known bounds or are almost tight (see Table 1). It therefore seems that parameter r correlates quite well with the "difficulty" of disjoint paths problems.

Our first result allows the paths to have small congestion: in this setting, we strengthen the result, obtained by the classic randomized LP-rounding approach of Raghavan and Thompson [38], that one can always route $\Omega(\mathrm{OPT})$ pairs with congestion $\mathcal{O}\left(\frac{\log n}{\log\log n}\right)$ with constant probability.

▶ **Theorem 2.** *For any instance* $(G,\mathcal{M})$ *of* MaxEDP, *one can efficiently find a routing of* $\Omega(\mathrm{OPT})$ *pairs with congestion* $\mathcal{O}\left(\frac{\log kr}{\log\log kr}\right)$ *with constant probability; in other words, there is an efficient* $\mathcal{O}(1)$*-approximation algorithm for* MaxEDP *with congestion* $\mathcal{O}\left(\frac{\log kr}{\log\log kr}\right)$.

Our second main result builds upon Theorem 2 and uses it as a subroutine. We show how to use a routing for MaxEDP with low congestion to obtain a polynomial-time approximation algorithm for MaxEDP *without congestion* that performs well in terms of r.

▶ **Theorem 3.** *The integrality gap of the multi-commodity flow relaxation for* MaxEDP *with* k *terminal pairs is* $\mathcal{O}(\sqrt{r}\cdot\log^{1.5} rk)$ *for graphs with feedback vertex set number* r. *Moreover, there is a polynomial time algorithm that, given a fractional solution to the relaxation of value* opt, *it constructs an integral routing of size* $\mathsf{opt}/\mathcal{O}(\sqrt{r}\cdot\log^{1.5} rk)$.

In particular, our algorithm strengthens the best known approximation algorithm for MaxEDP on general graphs [12] as always $r\leq n$, and indeed it matches that algorithm's performance up to polylogarithmic factors. Substantially improving upon our bounds would also improve the current state of the art of MaxEDP. Conversely, the result implies that it suffices to study graphs with close to linear feedback vertex set number in order to improve the currently best upper bound of $\mathcal{O}(\sqrt{n})$ on the approximation ratio [12].

Our algorithmic approaches harness the forest structure of $G-R$ for any feedback vertex set R. However, the technical challenge comes from the fact that the edge set running between $G-R$ and R is unrestricted. Therefore, the "interaction" between R and $G-R$ is non-trivial, and flow paths may run between the two parts in an arbitrary manner and multiple times. In fact, we show that MaxEDP is already NP-hard if R consists of a *single node* (Theorem 6); this contrasts the efficient solvability on forests [24].

In order to overcome the technical hurdles we propose several new concepts, which we believe could be of interest in future studies of disjoint paths or routing problems.

In the randomized rounding approach of Raghavan and Thompson [38], it is shown that the probability that the congestion on any fixed edge is larger than $c\frac{\log n}{\log\log n}$ for some constant c is at most $1/n^{O(1)}$. Combining this with the fact that there are at most n^2 edges, yields that every edge has bounded congestion w.h.p. The number of edges in the graph may, however, be unbounded in terms of r and k. Hence, in order to to prove Theorem 2, we propose a non-trivial *pre-processing step* of the optimum LP solution that is applied prior to the randomized rounding. In this step, we aggregate the flow paths by a careful rerouting so that the flow "concentrates" in $O(kr^2)$ nodes (so-called *hot spots*) in the sense that if all edges incident on hot spots have low congestion then so have all edges in the graph. Unfortunately, for any such hot spot the number of incident edges carrying flow may still be unbounded in terms of k and r. We are, however, able to give a refined probabilistic analysis that suitably relates the probability that the congestion bound is exceeded to the amount of flow on that edge. Since the total amount of flow on each hot spot is bounded in terms of k, the probability that *all* edges incident on the same hot spot have bounded congestion is inverse polynomial in r and k.

The known $\mathcal{O}(\sqrt{n})$-approximation algorithm for MaxEDP by Chekuri et al. [12] employs a clever LP-rounding approach. If there are many long paths then there must be a single node carrying a significant fraction of the total flow and a good fraction of this flow can be realized by integral paths by solving a single-source flow problem. If the LP solution contains many short flow paths then greedily routing these short paths yields the bound since each such path blocks a bounded amount of flow. In order to prove Theorem 3, it is natural to consider the case where there are many paths visiting a large number of nodes in R. In this case, we reduce to a single-source flow problem, similarly to the approach of Chekuri et al. The case where a majority of the flow paths visit only a few nodes in R turns out more challenging, since any such path may still visit an unbounded number of edges in terms of k and r. We use two main ingredients to overcome these difficulties. First, we apply our Theorem 2 as a building block to obtain a solution with logarithmic congestion while losing only a constant factor in the approximation ratio. Second, we introduce the concept of *irreducible routings with low congestion* which allows us exploit the structural properties of the graph and the congestion property to identify a sufficiently large number of flow paths blocking only a small amount of flow.

Note that the natural greedy approach of always routing the shortest conflict-free path gives only $\mathcal{O}(\sqrt{m})$ for MaxEDP. We believe that it is non-trivial to obtain our bounds via a more direct or purely combinatorial approach.

Our third result is a fixed-parameter algorithm for MaxNDP in $k + r$.

▶ **Theorem 4.** MaxNDP *can be solved in time* $(8k + 8r)^{2r+2} \cdot \mathcal{O}(n)$ *on graphs with feedback vertex set number* r *and* k *terminal pairs.*

This run time is polynomial for constant r. We also note that for small r, our algorithm is asymptotically significantly faster than the fastest known algorithm for NDP, by Kawarabayashi and Wollan [29], which requires time at least *quadruple-exponential* in k [1]. Namely, if r is at most triple-exponential in k, our algorithm is asymptotically faster than theirs. We achieve this result by the idea of so-called *essential pairs* and *realizations*, which characterizes the "interaction" between the feedback vertex set R and the paths in an optimum solution. Note that in our algorithm of Theorem 4, parameter k does not appear in the exponent of the run time at all. Hence, for small values of r, our algorithm is also faster than reducing MaxNDP to NDP by guessing the subset of pairs to be routed (at an expense of 2^k in the run time) and using Scheffler's [41] algorithm for NDP with run time $2^{O(r \log r)} \cdot \mathcal{O}(n)$.

Once a fixed-parameter algorithm for a problem has been obtained, the existence of a polynomial-size kernel comes up. Here we note that MaxNDP does not admit a polynomial kernel for parameter $k + r$, unless $\mathsf{NP} \subseteq \mathsf{coNP}/poly$ [8].

Table 1 Summary of results obtained in this paper.

const.	param.	EDP	MaxEDP	NDP	MaxNDP
$r = 0$		poly [24]	poly [24]	poly [41]	poly (Thm. 4)
$r = 1$		*open*	NP-hard (Thm. 6)	poly [41]	poly (Thm. 4)
$r \geq 2$		NP-hard (Thm. 6)	NP-hard (Thm. 6)	poly [41]	poly (Thm. 4)
	r	para-NP-hard (Thm. 6)		FPT [41]	W[1]-hard (Thm. 5)
		$\mathcal{O}(\sqrt{r} \cdot \log^{1.5} kr)$-approx (Thm. 3)			exact $(k + r)^{\mathcal{O}(r)}$ (Thm. 4)
		$\mathcal{O}(1)$-approx. w.cg. $\mathcal{O}\left(\frac{\log kr}{\log\log kr}\right)$ (Thm. 2)			

Another natural question is whether the run time $f(k, r) \cdot n$ in Theorem 4 can be improved to $f(r) \cdot n^{\mathcal{O}(1)}$. We answer this question in the negative, ruling out the existence of a fixed-parameter algorithm for MaxNDP parameterized by r (assuming FPT $\neq$ W[1]):

▶ **Theorem 5.** MaxNDP *in unit-capacity graphs is* W[1]*-hard parameterized by* r*.*

This contrasts the known result that NDP is fixed-parameter tractable in r [41]—which further stresses the relevance of understanding this parameter. We prove Theorem 5 in the full version of the paper [22].

For MaxEDP, we prove that the situation is, in a sense, even worse:

▶ **Theorem 6.** MaxEDP *is* NP*-hard for unit-capacity graphs with* $r = 1$ *and* EDP *is* NP*-hard for unit-capacity graphs with* $r = 2$*.*

This theorem also shows that our algorithms are relevant for small values of r, and they nicely complement the NP-hardness for MaxEDP in capacitated trees [24].

Our results are summarized in Table 1.

Related Work. Our study of the feedback vertex set number is in line with the general attempt to obtain bounds for MaxEDP (or related problems) that are independent of the input size. Besides the above-mentioned works that provide bounds in terms of the *tree-width* of the input graph, Günlük [25] and Chekuri et al. [17] give bounds on the *flow-cut gap* for the closely related integer multicommodity flow problem that are logarithmic with respect to the *vertex cover number* of a graph. This improved upon earlier bounds of $\mathcal{O}(\log n)$ [34] and $\mathcal{O}(\log k)$ [5, 35]. As every feedback vertex set is in particular a vertex cover of a graph, our results generalize earlier work for disjoint path problems on graphs with bounded vertex cover number. Bodlaender et al. [8] showed that NDP does not admit a polynomial kernel parameterized by vertex cover number *and* the number k of terminal pairs, unless NP $\subseteq$ coNP/*poly* ; therefore, NDP is unlikely to admit a polynomial kernel in $r + k$ either. Ene et al. [21] showed that MaxNDP is W[1]-hard parameterized by treedepth, which is another restriction of treewidth that is incomparable to the feedback vertex set number.

The basic gap in understanding the approximability of MaxEDP has led to several improved results for special graph classes, and also our results can be seen in this light. For example, polylogarithmic approximation algorithms are known for graphs whose global minimum cut value is $\Omega(\log^5 n)$ [39], for bounded-degree expanders [10, 9, 30, 34, 23], and for Eulerian planar or 4-connected planar graphs [28]. Constant factor approximation algorithms are known for capacitated trees [24, 14], grids and grid-like graphs [4, 6, 31, 32]. For planar graphs, there is a constant-factor approximation algorithm with congestion 2 [42]. Very

(MaxEDP LP)

$$\begin{array}{lll} \max & \displaystyle\sum_{i=1}^{k} x_i & \\ \text{s.t.} & \displaystyle\sum_{P \in \mathcal{P}(s_i,t_i)} f(P) = x_i \leq 1 & i = 1, \dots, k, \\ & \displaystyle\sum_{P\colon e \in P} f(P) \leq 1 & e \in E(G) \\ & f(P) \geq 0 & P \in \mathcal{P} \end{array}$$

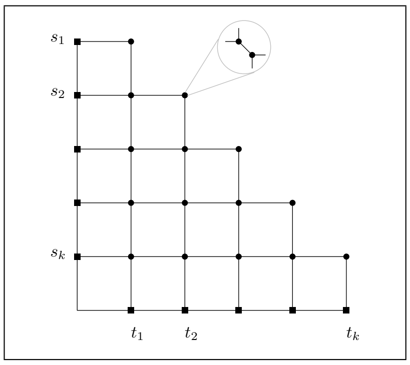


Figure 1 Multi-commodity flow relaxation for MaxEDP. Right: $\Omega(\sqrt{n})$ integrality gap for MaxEDP [24]: any integral routing routes at most one pair, whereas a multi-commodity flow can send 1/2 unit of flow for each pair (s_i, t_i) along the canonical path from s_i to t_i in the grid.

recently, Chuzhoy et al. [19] gave a $\tilde{\mathcal{O}}(n^{9/19})$-approximation algorithm for MaxNDP on *planar* graphs. However, improving the $\mathcal{O}(\sqrt{n})$-approximation algorithm for MaxEDP remains elusive even for *planar* graphs.

2 Preliminaries

We use standard graph theoretic notation. For a graph G, let $V(G)$ denote its vertex set and $E(G)$ its edge set. Let G be a graph. A *feedback vertex set* of G is a set $R \subseteq V(G)$ such that $G - R$ is a forest. A *minor* of G is a graph H that is obtained by successively contracting edges from a subgraph of G (and deleting any occurring loops). A class $\mathcal{G}$ of graphs is *minor-closed* if for any graph in $\mathcal{G}$ also all its minors belong to $\mathcal{G}$.

For an instance $(G, \mathcal{M})$ of MaxEDP/MaxNDP, we refer to the vertices participating in the pairs $\mathcal{M}$ as *terminals*. It is convenient to assume that $\mathcal{M}$ forms a matching on the terminals; this can be ensured by making several copies of a terminal and attaching them as leaves.

Multi-commodity flow relaxation. We use the following standard multi-commodity flow relaxation for MaxEDP (there is an analogous relaxation for MaxNDP). We use $\mathcal{P}(u, v)$ to denote the set of all paths in G from u to v, for each pair (u, v) of nodes. Since the pairs $\mathcal{M}$ form a matching, the sets $\mathcal{P}(s_i, t_i)$ are pairwise disjoint. Let $\mathcal{P} = \bigcup_{i=1}^{k} \mathcal{P}(s_i, t_i)$. The LP has a variable $f(P)$ for each path $P \in \mathcal{P}$ representing the amount of flow on P. For each pair $(s_i, t_i) \in \mathcal{M}$, the LP has a variable x_i denoting the total amount of flow routed for the pair (in the corresponding IP, x_i denotes whether the pair is routed or not). The LP imposes the constraint that there is a flow from s_i to t_i of value x_i. Additionally, the LP has constraints that ensure that the total amount of flow on paths using a given edge (resp. node for MaxNDP) is at most 1.

It is well-known that the relaxation MaxEDP LP can be solved in polynomial time, since there is an efficient separation oracle for the dual LP (alternatively, one can write a compact relaxation). We use $(f, \mathbf{x})$ to denote a feasible solution to MaxEDP LP for an instance $(G, \mathcal{M})$ of MaxEDP. For each terminal v, let $x(v)$ denote the total amount of flow routed for v and we refer to $x(v)$ as the *marginal value* of v in the multi-commodity flow f.

We will use the following result by Chekuri et al. [12, Sect. 3.1]; see also Proposition 3.3 of Chekuri et al. [16].

▶ **Proposition 7.** *Let $(f, \mathbf{x})$ be a fractional solution to the LP relaxation of a* MaxEDP *instance $(G, \mathcal{M})$. If some node v is contained in all flow paths of f, then we can find an integral routing of size at least $\frac{1}{12}\sum_i x_i$ in polynomial time.*

3 Bi-Criteria Approximation for MaxEDP with Low Congestion

We present a randomized rounding algorithm that will lead to the proof of Theorem 2.

3.1 Algorithm

Consider an instance $(G, \mathcal{M})$ of MaxEDP. Let R be a 2-approximate minimum feedback vertex set of G and let $r = |R|$; note that such a set R can be obtained in polynomial time [7].

For the sake of easier presentation, we will assume in this section that the feedback vertex set R contains all terminal nodes from $\mathcal{M}$. This can be achieved by temporarily adding the set of terminals to the feedback vertex set R. Also note that this assumption increases the bound of Theorem 2 by at most a constant factor.

First, solve the corresponding MaxEDP LP. We obtain an optimal extreme point solution $(f, \mathbf{x})$. For each $(s_i, t_i) \in \mathcal{M}$, this gives us a set $\mathcal{P}'(s_i, t_i) = \{P \in \mathcal{P}(s_i, t_i) \mid f(P) > 0\}$ of positive weighted paths that satisfy the LP constraints.

Since we have an extreme point solution, the number of tight constraints is not smaller then the number of variables. As the number of constraints that are not of type $f(P) \geq 0$ is polynomially bounded in the input size, the same holds for the cardinality of the set $\mathcal{P}' = \bigcup_{i=1}^{k} \mathcal{P}'(s_i, t_i)$. In what follows, we will modify $\mathcal{P}'$ and then select an (unweighted) subset $\mathcal{S}$ of $\mathcal{P}'$ that will form our integral solution.

Each $P \in \mathcal{P}'$ has the form $(r_1, \dots, r_2, \dots, r_\ell)$ where $r_1, \dots, r_\ell$ are the nodes in R that are traversed by P in this order. The paths $(r_j, \dots, r_{j+1})$ with $j = 1, \dots, \ell - 1$ are called *subpaths of* P. For every subpath P' of P, we set $f(P') = f(P)$. Let $\mathcal{J}$ be the multi-set of all subpaths of all paths in $\mathcal{P}'$. Let $F = G - R$ be the forest obtained by removing R.

We now modify some paths in $\mathcal{P}'$, one by one, and at the same time construct a subset H_0 of nodes that we will call "hot spots". At the end, every subpath in $\mathcal{J}$ will contain at least one hot spot.

Initially, let $H_0 = \emptyset$. Consider any tree T in F and fix any of its nodes as a root. Then let $\mathcal{J}_T$ be the multi-set of all subpaths in $\mathcal{J}$ that, excluding the endpoints, are contained in T. For each subpath $P \in \mathcal{J}_T$, define its *highest node* $h(P)$ as the node on P closest to the root. Note that $P \cap T = P \cap F$ is a path. Now, pick a subpath $P \in \mathcal{J}_T$ that does not contain any node in H_0 and whose highest node $h(P)$ is *farthest away* from the root. Consider the multi-set $\mathcal{J}[P]$ of all subpaths in $\mathcal{J}_T$ that are identical to P (but may be subpaths of different flow paths in $\mathcal{P}'$). Note that the weight $f(\mathcal{J}[P]) := \sum_{P \in \mathcal{J}[P]} f(P)$ of $\mathcal{J}[P]$ is at most 1 by the constraints of the LP. Let $u, v \in R$ be the endpoints of P. We define $\mathcal{J}_{uv}$ as the set of all subpaths in $\mathcal{J} \setminus \mathcal{J}[P]$ that have u and v as their endpoints and that do not contain any node in H_0.

Intuitively speaking, we now aggregate flow on P by rerouting as much flow as possible from $\mathcal{J}_{uv}$ to P. To this end, we repeatedly perform the following operation as long as $f(\mathcal{J}[P]) < 1$ and $\mathcal{J}_{uv} \neq \emptyset$. We pick a path P' in $\mathcal{J}$ that contains a subpath in $\mathcal{J}_{uv}$. We reroute flow from P' by creating a new path P'' that arises from P' by replacing its subpath between u and v with P, and assign it the weight $f(P'') = \min\{f(P'), 1 - f(\mathcal{J}[P])\}$. Then we set the weight of (the original path) P' to $\max\{0, f(P') + f(\mathcal{J}[P]) - 1\}$. We update the sets $\mathcal{P}'$, $\mathcal{P}'(s_i, t_i)$, $\mathcal{J}$, $\mathcal{J}_T$, $\mathcal{J}[P]$ and $\mathcal{J}_{uv}$ accordingly.

As soon as $f(\mathcal{J}[P]) = 1$ or $\mathcal{J}_{uv} = \emptyset$, we add $h(P)$ to H_0. Then, we proceed with the next $P \in \mathcal{J}_T$ not containing a hot spot and whose highest node $h(P)$ is farthest away from the root. If no such P is left, we consider the next tree T in F.

At the end, we create our solution $\mathcal{S}$ by randomized rounding: We route every terminal pair (s_i, t_i) with probability x_i. In case (s_i, t_i) is routed, we randomly select a path from $\mathcal{P}'(s_i, t_i)$ and add it to $\mathcal{S}$ where the probability that path P is taken is $f(P)/x_i$.

3.2 Analysis

First, observe that $\mathbf{x}$ did not change during our modifications of the paths, as the total flow between any terminal pair did not change. Thus, the expected number of pairs routed in our solution is $\sum_{i=1}^{k} x_i \geq \mathrm{OPT}$. Using the Chernoff bound, the probability that we route less than $\mathrm{OPT}/2$ pairs is at most $e^{-1/8\,\mathrm{OPT}} < 1/2$, assuming that $\mathrm{OPT} > 8$. Secondly, we bound the congestion of our solution—our second criterion.

▶ **Lemma 8.** *The congestion of flow f is at most 2.*

Proof. In our algorithm, we increase the flow only along flow subpaths that are pairwise edge-disjoint. To see this, consider two distinct flow subpaths P and P' on which we increase the flow. Assume, without loss of generality, that P was considered before P' by the algorithm. If there was an edge e lying on P and P', then both subpaths traverse the same tree in forest F. Hence, the path from e to $h(P')$ would visit $h(P)$, and $h(P)$ would be an internal node of P'. This yields a contradiction, as $h(P)$ was already marked as a hot spot when P' was considered. This shows that we increased the flow along any edge by at most one unit, and, hence, f has congestion at most 2. ◀

We now bound the congestion of the integral solution obtained by randomized rounding. In the algorithm, we constructed a set H_0 of hot spots. As a part of the analysis, we will now extend this set to a set H as follows. Initially, $H = H_0$. We build a sub-forest F' of F consisting of all edges of F that lie on a path connecting two hot spots. Then we add to H all nodes that have degree at least 3 in F'. Since the number of nodes of degree 3 in any forest is at most its number of leaves and since every leaf of F' is a hot spot, it follows that this can at most double the size of H to $2|H_0|$. Finally, we add the set R of all feedback vertex nodes to H. In the following, all nodes in H are called hot spots.

▶ **Lemma 9.** *The number $|H|$ of hot spots is $\mathcal{O}(kr^2)$.*

Proof. It suffices to show that $|H_0| \in \mathcal{O}(kr^2)$. To this end, fix two nodes $u, v \in R$ and consider the set of flow subpaths P with end nodes u and v for which we added $h(P)$ to H_0. Due to the aggregation of flows in our algorithm, all except possibly one of the subpaths are saturated, that is, they carry precisely one unit of flow. Since no two of these subpaths are contained in a same flow path of f and since the flow value of f is bounded from above by k, we added only $\mathcal{O}(k)$ hot spots for the pair u, v. Since there are at most r^2 pairs in R, the claim follows. ◀

▶ **Definition 10.** A hot spot $u \in H$ is *good* if the congestion on any edge incident on u is bounded by $c \cdot \frac{\log kr}{\log\log kr}$, where c is a sufficiently large constant; otherwise, u is *bad*.

▶ **Lemma 11.** *Let $u \in H$ be a hot spot. Then the probability that u is bad is at most $1/(k^2r^3)$.*

Proof. Let $e_1 = uv_1, \ldots, e_\ell = uv_\ell$ be the edges incident on u and let f_i be the total flow on edge uv_i for $i = 1, \ldots, \ell$. By Lemma 8, we have that $f_i \leq 2$. Since any flow path visits at most two of the edges incident on u, the total flow $\sum_{i=1}^{\ell} f_i$ on the edges incident on u is at most $2k$.

For any $i = 1, \dots, \ell$, we have that $f_i = \sum_{P\colon P \ni e_i} f(P)$, where P runs over the set of all paths connecting some terminal pair and containing e_i. Let $f_{ij} = \sum_{P \in \mathcal{P}(s_j,t_j)\colon P \ni e_i} f(P)$ be the total amount of flow sent across e_i by terminal pair (s_j, t_j). Recall that x_j is the total flow sent for terminal pair (s_j, t_j). The probability that the randomized rounding procedure picks path P with $P \in \mathcal{P}(s_j, t_j)$ is precisely $x_j \cdot \frac{f(P)}{x_j} = f(P)$. Given the disjointness of the respective events, the probability that pair (s_j, t_j) routes a path across e_i is precisely f_{ij}. Let X_{ij} be the binary random variable indicating whether pair (s_j, t_j) routes a path across e_i. Then $\mathbb{P}[X_{ij} = 1] = f_{ij}$. Let $X_i = \sum_j X_{ij}$ be the number of paths routed across e_i by the algorithm. By linearity of expectation, we have that $\mathbb{E}[X_i] = \sum_j \mathbb{E}[X_{ij}] = \sum_j f_{ij} = f_i$.

Fix any edge e_i. Set $\delta = c \cdot \frac{\log kr}{\log\log kr}$ and $\delta' = 2\frac{\delta}{f_i} - 1$. Note that for fixed i, the variables X_{ij} are independent. Hence, by the Chernoff bound, we have that

$$\mathbb{P}\left[X_i \geq c \cdot \frac{\log kr}{\log\log kr}\right] \leq \mathbb{P}\left[X_i \geq (1+\delta')f_i\right] < \left(\frac{e^{\delta'}}{(1+\delta')^{1+\delta'}}\right)^{f_i}$$
$$\leq \left(\frac{f_i}{2}\right)^{2\delta} \cdot \left(\frac{\delta}{e}\right)^{-2\delta} \leq f_i e^{-c' \log\log kr \cdot \frac{\log kr}{\log\log kr}} \leq \frac{f_i}{2k^3 r^3}\,.$$

Here, we use that $f_i \leq 2$ for the second last inequality and for the last inequality we pick c' sufficiently large by making c and k sufficiently large. (Note that MaxEDP can be solved efficiently for constant k.)

Now, using the union bound, we can infer that the probability that any of the edges incident on u carries more than δ paths is at most $\sum_i f_i/(2k^3r^3) \leq (2k)/(2k^3r^3) = 1/(k^2r^3)$. ◀

▶ **Lemma 12.** *Assume that every hot spot is good. Then the congestion on any edge is bounded by* $2c\frac{\log kr}{\log\log kr}$.

Proof. Consider an arbitrary edge $e = uv$ that is not incident on any hot spot. In particular, this means that e lies in the forest $F = G - R$. A hot spot z in F is called *direct* to e if the path in F from z to e excluding e does not contain any hot spot other than z.

We claim that there can be at most two distinct hot spots z, z' direct to e. If there were a third hot spot z'' direct to e, then consider the unique node $z_0 \in V(F)$ such that no two of the hot spots z, z', z'' are connected in $F - z_0$. Such a node z_0 exists since z, z', z'' cannot lie on a common path in F since they are all direct to e. The node z_0, however, would be added as a hot spot at the latest when H was built. Now, this is a contradiction, because then one of the paths connecting z, z' or z'' to e would contain z_0 and thus one of these hot spots would not be direct to e.

Now let P be an arbitrary path that is routed by our algorithm and that traverses e, and let $P' \in \mathcal{J}$ be the subpath of P in F visiting e. Moreover, let P_z, $P_{z'}$ be the paths in F connecting z, z' to e excluding e, and let e_z, $e_{z'}$ be the edges on these paths incident on z, z', respectively. By our construction, P' must visit a hot spot in F. If P' visited neither z nor z', then P' would contain a hot spot direct to u or to v that is distinct from z and z'—a contradiction. Hence P' and thus also P visit e_z or $e_{z'}$. The claim now follows from the fact that this holds for any path traversing e, that z and z' are good, and that therefore altogether at most $2c\frac{\log kr}{\log\log kr}$ paths visit e_z or e'_z. ◀

▶ **Theorem 13.** *The algorithm from Sect. 3.1 produces—with constant probability—a routing with* $\Omega(\mathrm{OPT})$ *paths, such that the congestion is* $\mathcal{O}\left(\frac{\log kr}{\log\log kr}\right)$.

Proof. As argued above, we route less than OPT $/2$ paths with probability at most $1/2$. By Lemma 9, there are $\mathcal{O}(kr^2)$ hotspots. The probability that at least one of these hot spots is bad is $\mathcal{O}(kr^2/(k^2r^3)) = \mathcal{O}(1/(kr))$, by Lemma 11. Hence, with constant probability, we route at least OPT $/2$ pairs with congestion at most $2c\frac{\log kr}{\log\log kr}$, by Lemma 12. ◀

4 Refined Approximation Bound for MaxEDP

In this section, we provide an improved approximation guarantee for MaxEDP *without* congestion, thereby proving Theorem 3. (In contrast to the previous section, we do not assume here that all terminals are contained in the feedback vertex set.)

4.1 Irreducible Routings with Low Congestion

We first develop the concept of *irreducible routings with low congestion*, which is (besides Theorem 2) a key ingredient of our strengthened bound on the approximability of MaxEDP based on the feedback vertex number.

Consider any multigraph G and any set $\mathcal{P}$ of (not necessarily simple) paths in G with congestion c. We say that an edge e is *redundant in* $\mathcal{P}$ if there is an edge $e' \neq e$ such that the set of paths in $\mathcal{P}$ *covering* (containing) e is a subset of the set of paths in $\mathcal{P}$ covering e'. Thus, any edge that is not covered by any path in $\mathcal{P}$ is redundant in $\mathcal{P}$ if G contains at least two edges.

▶ **Definition 14.** Set $\mathcal{P}$ is called an *irreducible routing with congestion* c if each edge belongs to at most c paths of $\mathcal{P}$ and there is no edge redundant in $\mathcal{P}$.

In contrast to a feasible routing of an MaxEDP instance, we do not require an irreducible routing to connect a set of terminal pairs. If there is an edge e redundant in $\mathcal{P}$, we can apply the following *reduction rule*: We contract e in G and we contract e in every path of $\mathcal{P}$ that covers e. By this, we obtain a minor G' of G and a set $\mathcal{P}'$ of paths that consists of all the contracted paths and of all paths in $\mathcal{P}$ that were not contracted. Thus, there is a one-to-one correspondence between the paths in $\mathcal{P}$ and $\mathcal{P}'$.

We make the following observation about $\mathcal{P}$ and $\mathcal{P}'$.

▶ **Observation 15.** *A subset of paths in $\mathcal{P}'$ is edge-disjoint in G' if and only if the corresponding subset of paths in $\mathcal{P}$ is edge-disjoint in G.*

As applying the reduction rule strictly decreases the number of redundant edges, an iterative application of this rule yields an irreducible routing on a minor of the original graph.

▶ **Theorem 16.** *Let $\mathcal{G}$ be a minor-closed class of multigraphs and let $p_{\mathcal{G}} > 0$. If for each graph $G \in \mathcal{G}$ and every non-empty irreducible routing $\mathcal{S}$ of G with congestion c there exists a path in $\mathcal{S}$ of length at most $p_{\mathcal{G}}$, then the average length of the paths in $\mathcal{S}$ is at most $c \cdot p_{\mathcal{G}}$.*

Proof. Take a path P_0 of length at most $p_{\mathcal{G}}$. Contract all edges of P_0 in G and obtain a minor $G' \in \mathcal{G}$ of G. For each path in $\mathcal{S}$ contract all edges shared with P_0 to obtain a set $\mathcal{S}'$ of paths. Remove P_0 along with all degenerated paths from $\mathcal{S}'$, thus $|\mathcal{S}'| < |\mathcal{S}|$. Note that $\mathcal{S}'$ is an irreducible routing of G' with congestion c. We repeat this reduction procedure recursively on G' and S' until S' is empty which happens after at most $|\mathcal{S}|$ steps. At each step we decrease the total path length by at most $c \cdot p_{\mathcal{G}}$. Hence, the total length of paths in $\mathcal{S}$ is at most $|\mathcal{S}| \cdot c \cdot p_{\mathcal{G}}$. ◀

As a consequence of Theorem 16, we get the following result for forests.

▶ **Lemma 17.** *Let F be a forest and let $\mathcal{S}$ be a non-empty irreducible routing of F with congestion c. Then the average path length in $\mathcal{S}$ is at most $2c$.*

Proof. We show that $\mathcal{S}$ contains a path of length as most 2. The lemma follows immediately by applying Theorem 16.

Take any tree in F, root it with any node and consider a leaf v of maximum depth. If v is adjacent to the root, then the tree is a star and every path in the tree has length at most 2. Otherwise, let e_1 and e_2 be the first two edges on the path from v to the root. By definition of irreducible routing, the set of all paths covering e_1 is not a subset of the paths covering e_2; hence, e_1 is covered by a path which does not cover e_2. Since all other edges incident to e_1 end in a leaf, this path has length at most 2. ◀

Note that the bound provided in Lemma 17 is actually tight up to a constant. Let $c \geq 1$ be an arbitrary integer. Consider a graph that is a path of length $c-1$ with a star of $c-1$ leafs attached to one of its end points. The $c-1$ paths of length c together with the $2c-2$ paths of length 1 form an irreducible routing with congestion c. The average path length is $((c-1)c + (2c-2))/(3c-3) = (c+2)/3$.

4.2 Approximation Algorithm

Consider an instance $(G, \mathcal{M})$ of MaxEDP, and let r be the size of a feedback vertex set R in G. Using our result of Sect. 3, we can efficiently compute a routing $\mathcal{P}$ with congestion $c := \mathcal{O}\left(\frac{\log kr}{\log\log kr}\right)$ containing $\Omega(\mathrm{OPT})$ paths.

Below we argue how to use the routing $\mathcal{P}$ to obtain a feasible routing of cardinality $\Omega\left(|\mathcal{P}|/(c^{1.5}\sqrt{r})\right)$, which yields an overall approximation ratio of $\mathcal{O}\left(\sqrt{r} \cdot \log^{1.5} rk\right)$; that will prove Theorem 3.

Let $r' = \sqrt{r/c}$. We distinguish the following cases.

Case 1: At least half of the paths in $\mathcal{P}$ visit at most r' nodes of the feedback vertex set R. Let $\overline{\mathcal{P}}$ be the subset of these paths. Initialize $\mathcal{P}'$ with $\overline{\mathcal{P}}$. As long as there is an edge e not adjacent to R that is redundant in $\mathcal{P}'$, we iteratively apply the reduction rule from Sect. 4.1 on e. Let G' be the obtained minor of G with forest $F' = G' - R$. The obtained set $\mathcal{P}'$ is a set of (not necessarily simple) paths in G' corresponding to $\overline{\mathcal{P}}$. By (iterated application of) Observation 15 to path sets $\overline{\mathcal{P}}$ and $\mathcal{P}'$, it suffices to show that there is a subset $\mathcal{P}'_0 \subseteq \mathcal{P}'$ of pairwise edge-disjoint paths of size $|\mathcal{P}'_0| = \Omega\left(|\mathcal{P}|/(cr')\right)$ in order to obtain a feasible routing for $(G, \mathcal{M})$ of size $\Omega\left(|\mathcal{P}|/(cr')\right)$.

To obtain $\mathcal{P}'_0$, we first bound the total path length in $\mathcal{P}'$. Removing R from G' “decomposes” the set P' into a set $\mathcal{S} := \{S \text{ is a connected component of } P \cap F \mid P \in \mathcal{P}'\}$ of subpaths lying in F'. Observe that $\mathcal{S}$ is an irreducible set of F' with congestion c, as the reduction rule is not applicable anymore. (Note that a single path in $\mathcal{P}'$ may lead to many paths in the cover $\mathcal{S}$ which are considered distinct.) Thus, by Lemma 17, the average path length in $\mathcal{S}$ is at most $2c$.

Let P be an arbitrary path in $\mathcal{P}'$. Each edge on P that is *not* in a subpath in $\mathcal{S}$ is incident on a node in R, and each node in R is incident on at most two edges in P. Together with the fact that P visits at most r' nodes in R and that the average length of the subpaths in $\mathcal{S}$ is at most $2c$, we can upper bound the total path length $\sum_{P \in \mathcal{P}'} |P|$ by $|\mathcal{P}'| r'(2c+2)$. Let $\mathcal{P}''$ be the set of the $|\mathcal{P}'|/2$ shortest paths in $\mathcal{P}'$. Hence, each path in $\mathcal{P}''$ has length at most $4r'(c+1)$.

We greedily construct a feasible solution $\mathcal{P}'_0$ by iteratively picking an arbitrary path P from $\mathcal{P}''$ adding it to $\mathcal{P}'_0$ and removing all paths from $\mathcal{P}''$ that share some edge with P (including P itself). We stop when $\mathcal{P}''$ is empty. As $\mathcal{P}''$ has congestion c, we remove at most $4r'c(c+1)$ paths from $\mathcal{P}''$ per iteration. Thus, $|\mathcal{P}'_0| \geq |\mathcal{P}''|/(4r'c(c+1)) = \Omega\left(|\mathcal{P}|/(c^{1.5}\sqrt{r}\right)$.

Case 2: At least half of the paths in $\mathcal{P}$ visit at least r' nodes of the feedback vertex set R. Let $\mathcal{P}'$ be the subset of these paths. Consider each path in $\mathcal{P}'$ as a flow of value $1/c$ and let f be the sum of all these flows. Note that f provides a feasible solution to the MaxEDP LP relaxation for (G, M) of value at least $|\mathcal{P}|/(2c)$. Note that each such flow path contributes $1/c$ unit of flow to each of the r' nodes in R it visits. Since every flow path in f has length at least r', the total inflow of the nodes in R is at least $|f|r'$. By averaging, there must be a node $v \in R$ of inflow at least $r'|f|/r = |f|/r'$. Let f' be the subflow of f consisting of all flow paths visiting v. This subflow corresponds to a feasible solution $(f', \mathbf{x}')$ of the LP relaxation of value at least $|f|/r' \geq |\mathcal{P}|/(2cr')$. Using Proposition 7, we can recover an integral feasible routing of size at least $\frac{1}{12}\sum_i x'_i \geq |\mathcal{P}|/(24cr') = \Omega\left(|\mathcal{P}|/(c^{1.5}\sqrt{r}\right)$.

This completes the proof of Theorem 3. ◀

5 Fixed-Parameter Algorithm for MaxNDP

We give a fixed-parameter algorithm for MaxNDP with run time $(k+r)^{\mathcal{O}(r)} \cdot n$, where r is the size of a minimum feedback vertex set in the given instance $(G, \mathcal{M})$. A feedback vertex set R of size r can be computed in time $2^{O(r)} \cdot \mathcal{O}(n)$ [36]. By the matching assumption, each terminal in $\mathcal{M}$ is a leaf. We can thus assume that none of the terminals is contained in R.

Consider an optimal routing $\mathcal{P}$ of the given MaxNDP instance. Let $\mathcal{M}_R \subseteq \mathcal{M}$ be the set of terminal pairs that are connected via $\mathcal{P}$ by a path that visits at least one node in R. Let $P \in \mathcal{P}$ be a path connecting a terminal pair $(s_i, t_i) \in \mathcal{M}_R$. This path has the form $(s_i, \dots, r_1, \dots, r_2, \dots, r_\ell, \dots, t_i)$, where $r_1, \dots, r_\ell$ are the nodes in R that are traversed by P in this order. The pairs $(s_i, r_1), (r_\ell, t_i)$ and (r_j, r_{j+1}) with $j = 1, \dots, \ell-1$ are called *essential* pairs for P. A node pair is called *essential* if it is essential for some path in $\mathcal{P}$. Let $\mathcal{M}_e$ be the set of essential pairs.

Let F be the forest that arises when deleting R from the input graph G. Let (u, v) be an essential pair. A u-v path P in G is said to *realize* (u, v) if all internal nodes of P lie in F. A set $\mathcal{P}'$ of paths is said to *realize* $\mathcal{M}_e$ if every pair in $\mathcal{M}_e$ is realized by some path in $\mathcal{P}'$ and if two paths in $\mathcal{P}'$ can only intersect at their end nodes. Note that the optimal routing $\mathcal{P}$ induces a natural realization of $\mathcal{M}_e$, by considering all maximal subpaths of paths in $\mathcal{P}$ whose internal nodes all lie in F. Conversely, for any realization $\mathcal{P}'$ of $\mathcal{M}_e$, we can concatenate paths in $\mathcal{P}'$ to obtain a feasible routing that connects all terminal pairs in $\mathcal{M}_R$. Therefore, we consider $\mathcal{P}'$ (slightly abusing notation) also as a feasible routing for $\mathcal{M}_R$.

In our algorithm, we first guess the set $\mathcal{M}_e$ (and thus $\mathcal{M}_R$). Then, by a dynamic program, we construct two sets of paths, $\mathcal{P}_e$ and $\mathcal{P}_F$ where $\mathcal{P}_e$ realizes $\mathcal{M}_e$ and $\mathcal{P}_F$ connects in F a subset of $\overline{\mathcal{M}}_R := \mathcal{M} \setminus \mathcal{M}_R$. In our algorithm, the set $\mathcal{P}_e \cup \mathcal{P}_F$ forms a feasible routing that maximizes $|\mathcal{P}_F|$ and routes all pairs in $\mathcal{M}_R$. (Recall that we consider the realization $\mathcal{P}_e$ of $\mathcal{M}_e$ as a feasible routing for $\mathcal{M}_R$.)

Now assume that we know set $\mathcal{M}_e$. We will describe below a dynamic program that computes an optimum routing in time $2^{\mathcal{O}(r)}(k+r)^{\mathcal{O}(1)}n$. For the sake of easier presentation, we only describe how to compute the cardinality of such a routing.

We make several technical assumptions that help to simplify the presentation. First, we modify the input instance as follows. We subdivide every edge incident on a node in R by introducing a single new node on this edge. Note that this yields an instance equivalent to the input instance. As a result, every neighbor of a node in R that lies in F, that is, every node in $N_G(R)$, is a leaf in F. Moreover, the set R is an independent set in G. Also recall that we assumed that every terminal is a leaf. Therefore, we may assume that R does not contain any terminal. We also assume that forest F is a rooted tree, by introducing a dummy node (which plays the role of the root) and arbitrarily connecting this node to every connected component of F by an edge. In our dynamic program, we will take care that no path visits this root node. We also assume that F is an ordered tree by introducing an arbitrary order among the children of each node.

For any node v, let F_v be the subtree of F rooted at v. Let $c_v := \deg_F(v) - 1$ be the number of children of v and let $v_1, \ldots v_{c_v}$ be the (ordered) children of v. Then, for $i = 1, \ldots, c_v$, let F_v^i denote the subtree of F_v induced by the union of v with the subtrees $F_{v_1}, \ldots, F_{v_i}$. For leaves v, we define F_v^0 as $F_v = v$.

We introduce a dynamic programming table T. It contains an entry for every F_v^i and every subset $\mathcal{M}'_e$ of $\mathcal{M}_e$. Roughly speaking, the value of such an entry is the solution to the subproblem, where we restrict the forest to F_v^i, and the set of essential pairs to $\mathcal{M}'_e$. More precisely, table T contains five parameters. Parameters v and i describing F_v^i, parameter $\mathcal{M}'_e$, and two more parameters u and b. Parameter u is either a terminal, or a node in R, and b is in one of the three states: *free*, *to-be-used*, or *blocked*. The value $T[v, i, \mathcal{M}'_e, u, b]$ is the maximum cardinality of a set $\mathcal{P}_F$ of paths with the following properties:

1. $\mathcal{P}_F$ is a feasible routing of some subset of $\overline{\mathcal{M}}_R$.
2. $\mathcal{P}_F$ is completely contained in F_v^i.
3. There is an additional set $\mathcal{P}_e$ of paths with the following properties:
   a. $\mathcal{P}_e$ is completely contained in $F_v^i \cup R$ and node-disjoint from the paths in $\mathcal{P}_F$.
   b. $\mathcal{P}_e$ is a realization of $\mathcal{M}'_e \cup \{(u, v)\}$ if $b =$ *to-be-used*. Else, it is a realization of $\mathcal{M}'_e$.
   c. There is no path in $\mathcal{P}_e \cup \mathcal{P}_F$ visiting v if $b =$ *free*.

If no such set $\mathcal{P}_F$ exists then $T[v, i, \mathcal{M}'_e, u, b]$ is $-\infty$.

Note that the parameter u is only relevant when $b =$ *to-be-used* (otherwise, it can just be ignored). Observe that $T[v, i, \mathcal{M}'_e, u, \textit{blocked}] \geq T[v, i, \mathcal{M}'_e, u, \textit{free}] \geq T[v, i, \mathcal{M}'_e, u, \textit{to-be-used}]$. Below, we describe how to compute the entries of T in a bottom-up manner.

In the base case v is a leaf. We set $T[v, 0, \emptyset, u, \textit{free}] = 0$. Then we set $T[v, 0, \mathcal{M}'_e, u, \textit{blocked}] = 0$ if $\mathcal{M}'_e$ is either empty, consists of a single pair of nodes in $R \cap N_G(v)$, or consists of a single pair where one node is v and the other one is in $R \cap N_G(v)$. Finally, we set $T[v, 0, \emptyset, u, \textit{to-be-used}] = 0$ if $u = v$ or u is in $R \cap N_G(v)$. For all other cases where v is a leaf, we set $T[v, i, \mathcal{M}'_e, u, b] = -\infty$.

For the inductive step, we consider the two cases $i = 1$ and $i > 1$. Let $i = 1$. It holds that $T[v, 1, \mathcal{M}'_e, u, \textit{to-be-used}] = T[v_1, c_v, \mathcal{M}'_e, u, \textit{to-be-used}]$ since the path in $\mathcal{P}_e$ realizing (u, v) has to start at a leaf node of F_{v_1}. It also holds that $T[v, 1, \mathcal{M}'_e, u, \textit{blocked}]$ and $T[v, 1, \mathcal{M}'_e, u, \textit{free}]$ are equal to $T[v_1, c_v, \mathcal{M}'_e, u, \textit{blocked}]$.

Now, let $i > 1$. In a high level view, we guess which part of $\mathcal{M}'_e$ is realized in $F_v^{i-1} \cup R$ and which part is realized in $F_{v_i} \cup R$. For this, we consider every tuple $(\mathcal{M}'_{e1}, \mathcal{M}'_{e2})$ such that $\mathcal{M}'_{e1} \uplus \mathcal{M}'_{e2}$ is a partition of $\mathcal{M}'_e$. By our dynamic programming table, we find a tuple that maximizes our objective. In the following, we assume that we guessed $(\mathcal{M}'_{e1}, \mathcal{M}'_{e2})$ correctly. Let us consider the different cases of b in more detail.

For $b =$ *free*, node v is not allowed to be visited by any path, especially by any path in $F_v^{i-1} \cup R$. Hence, $T[v, i, \mathcal{M}'_e, u, \textit{free}]$ is equal to

$$T[v, i-1, \mathcal{M}'_{e1}, u, \textit{free}] + T[v_i, c_{v_i}, \mathcal{M}'_{e2}, u, \textit{blocked}] \ .$$

In the case of $b = \textit{to-be-used}$, we have to realize (u, v) in $F_v^i \cup R$. For this, there are two possibilities: rither (u, v) is realized by a path in $F_v^{i-1} \cup R$, or there is a realizing path that first goes through $F_{v_i} \cup R$ and then reaches v via the edge (v_i, v). Hence, for the first case, we consider

$$T[v, i-1, \mathcal{M}'_{e1}, u, \textit{to-be-used}] + T[v_i, c_{v_i}, \mathcal{M}'_{e2}, u, \textit{blocked}],$$

for the second case, we consider

$$T[v, i-1, \mathcal{M}'_{e1}, u, \textit{free}] + T[v_i, c_{v_i}, \mathcal{M}'_{e2}, u, \textit{to-be-used}] \ .$$

Maximizing over both, we obtain $T[v, i, \mathcal{M}'_e, u, \textit{to-be-used}]$.

For the case of $b = \textit{blocked}$, we will consider two subcases. In the first subcase, there is no path in $\mathcal{P}_e \cup \mathcal{P}_F$ going through edge (v_i, v), hence, we get

$$T[v, i-1, \mathcal{M}'_{e1}, u, \textit{blocked}] + T[v_i, c_{v_i}, \mathcal{M}'_{e2}, u, \textit{blocked}] \ .$$

In the second subcase, there is a path P in $\mathcal{P}_e \cup \mathcal{P}_F$ going through edge (v_i, v). Since P is connecting two leafs in F_v^i, a part of P is in $F_v^{i-1} \cup R$ and the other part is in $F_{v_i} \cup R$. If $P \in \mathcal{P}_e$, then it is realizing a pair of $\mathcal{M}'_e$. Hence, for every pair $(u_1, u_2) \in \mathcal{M}'_e$, we have to consider the term

$$T[v, i-1, \mathcal{M}'_{e1} - (u_1, u_2), u_1, \textit{to-be-used}] + T[v_i, c_{v_i}, \mathcal{M}'_{e2} - (u_1, u_2), u_2, \textit{to-be-used}]$$

and the symmetric term where we swap u_1 and u_2. If $P \in \mathcal{P}_F$, then it is realizing a terminal pair of $\overline{\mathcal{M}}_R$. Hence, for every pair $(u_1, u_2) \in \overline{\mathcal{M}}_R$ we get the term

$$1 + T[v, i-1, \mathcal{M}'_{e1}, u_1, \textit{to-be-used}] + T[v_i, c_{v_i}, \mathcal{M}'_{e2}, u_2, \textit{to-be-used}]$$

and the symmetric term where we swap u_1 and u_2. Note that we count the path realizing (u_1, u_2) in our objective. Maximizing over all the terms of the two subcases, we obtain $T[v, i, \mathcal{M}'_e, u, \textit{to-be-used}]$.

Let us analyze the run time of algorithm described in Sect. 5. In order to guess $\mathcal{M}_e$, we enumerate all potential sets of essential pairs. There are at most $(2k + r + 1)^{2r}$ candidate sets to consider, since each pair contains a node in R, and each node in R is paired with at most two other nodes each of which is either a terminal or another node in R. For each particular guess $\mathcal{M}_e$, we run the above dynamic program. The number of entries in T—as specified by the five parameters v, i, $\mathcal{M}'_e, u$ and b—for each fixed $\mathcal{M}_e$ is at most $(\sum_{v \in V(F)} \deg_F(v)) \times 2^{2r} \times (2k + r) \times 3$. For the computation of each such entry, we consider all combinations of at most 2^{2r} partitions of $\mathcal{M}'_e$ with either at most r essential pairs in $\mathcal{M}'_e$, or with at most k terminal pairs in $\overline{\mathcal{M}}_R$. Altogether, this gives a run time of $(8k + 8r)^{2r+2} \cdot \mathcal{O}(n)$. This finishes the proof of Theorem 4.

6 Hardness of Edge-Disjoint Paths in Almost-Forests

In this section we show that EDP (and hence MaxEDP) is NP-hard already in graphs that are forests after deleting two nodes. That is, we prove Theorem 6.

Proof of Theorem 6. We first show NP-hardness of EDP for $r = 2$. We reduce from the problem Edge 3-Coloring in cubic graphs, which is NP-hard [26]. Given a cubic graph H, we construct a complete bipartite graph G, where one of the two partite classes of $V(G)$ consists of three nodes $\{v_1, v_2, v_3\}$, and the other partite class consists of $V(H)$. As terminal pairs, we create the set $\mathcal{M} = \{(s,t) \mid \{s,t\} \in E(H)\}$; in words, we want to connect a pair of nodes by a path in G if and only if they are connected by an edge in H. This completes the construction of the instance $(G,\mathcal{M})$ of MaxEDP. Notice that G has a feedback vertex set of size $r = 2$, since removing any size-2 subset of $\{v_1, v_2, v_3\}$ from G yields a forest.

Regarding correctness of the reduction, we show that H is 3-edge-colorable if and only if *all* pairs in $\mathcal{M}$ can be routed in G.

In the forward direction, suppose that H is 3-edge-colorable. Let $\varphi : E(H) \to \{1,2,3\}$ be a proper 3-edge-coloring of H. For $c = 1,2,3$, let $E_c \subseteq E(H)$ be the set of edges that receive color c under φ. Then there is a routing in G that routes all terminal pairs $\{(s,t) \in \mathcal{M} \mid \{s,t\} \in E_c\}$ exclusively via the node v_c (and thus via paths of length 2). Notice that this routing indeed yields edge-disjoint paths, for if there are distinct vertices $s, t_1, t_2 \in V(H)$ and edges $e_1 = \{s,t_1\}, e_2 = \{s,t_2\} \in E(H)$, then e_1, e_2 receive distinct colors under φ (as φ is proper), and so the two terminal pairs $\{s,t_1\}, \{s,t_2\}$ are routed via distinct nodes $c_1, c_2 \in \{v_1, v_2, v_3\}$, and thus also via edge-disjoint paths.

In the backward direction, suppose that all terminal pairs in $\mathcal{M}$ can be routed in G. Since H is cubic, any node $s \in V(H)$ is contained in three terminal pairs. Therefore, no path of the routing can have a node in $V(H)$ as an internal node and thus all paths in the routing have length 2. Then this routing naturally corresponds to a proper 3-edge-coloring φ of H, where any terminal pair $\{s,t\}$ routed via c means that we color the edge $\{s,t\} \in E(H)$ with color c under φ.

In order to show NP-hardness of MaxEDP for $r = 1$, we also reduce from Edge 3-Coloring in cubic graphs and perform a similar construction as described above: This time, we construct a bipartite graph G with one subset of the partition being $\{v_1, v_2\}$, the other being $V(H)$, and the set $\mathcal{M}$ of terminal pairs being again specified by the edges of H. This completes the reduction. The resulting graph G has a feedback vertex set of size $r = 1$.

We claim that H is 3-colorable if and only if we can route $n = |V(H)|$ pairs in G.

In the forward direction, suppose that H is 3-edge-colorable. Let $\varphi : E(H) \to \{1,2,3\}$ be a proper 3-edge-coloring of H. For $c = 1,2,3$, let $E_c \subseteq E(H)$ be the set of edges that receive color c under φ. Then there is a routing in G that routes all f $\{(s,t) \in \mathcal{M} \mid \{s,t\} \in E_c\}$ exclusively via the node v_c (and thus via paths of length 2) for the colors $c = 1,2$. (The terminals corresponding to edges receiving color 3 remain unrouted.)

The reasoning that the resulting routing is feasible is analogous to the case of $r = 2$. Since for each of the n terminals exactly two of the three terminal pairs are routed, this means that precisely n terminal pairs are routed overall.

In the backward direction, suppose that n terminal pairs in $\mathcal{M}$ can be routed in G. Since any terminal v in G is a node in $V(H)$ has therefore has degree two in G, this means that at most two paths can be routed for v. As n terminal pairs are realized, this also means that *exactly* two paths are routed for each terminal. Hence, none of the paths in the routing has length more than two. Otherwise, it would contain an internal node in $V(H)$, which then could not be part of two other paths in the routing. Then this routing naturally corresponds to a partial edge-coloring of H, where any terminal pair $\{s,t\}$ routed via c means that we color the edge $\{s,t\} \in E(H)$ with color c. Since each terminal v in $V(H)$ is involved in exactly two paths in the routing, exactly one terminal pair for v remains unrouted. Hence, exactly one edge incident on v in H remains uncolored in the partial coloring. We color all uncolored edges in H by color 3 to obtain a proper 3-coloring. ◀

Thus, we almost close the complexity gap for EDP with respect to the size of a minimum feedback vertex set, only leaving the complexity of the case $r = 1$ open.

References

1 Isolde Adler, Stavros G. Kolliopoulos, Philipp Klaus Krause, Daniel Lokshtanov, Saket Saurabh, and Dimitrios Thilikos. Tight bounds for linkages in planar graphs. In *Proc. ICALP 2011*, volume 6755 of *Lecture Notes Comput. Sci.*, pages 110–121, 2011.

2 Matthew Andrews. Approximation algorithms for the edge-disjoint paths problem via Räcke decompositions. In *Proc. FOCS 2010*, pages 277–286, 2010.

3 Matthew Andrews, Julia Chuzhoy, Venkatesan Guruswami, Sanjeev Khanna, Kunal Talwar, and Lisa Zhang. Inapproximability of edge-disjoint paths and low congestion routing on undirected graphs. *Combinatorica*, 30(5):485–520, 2010.

4 Yonatan Aumann and Yuval Rabani. Improved bounds for all optical routing. In *Proc. SODA 1995*, pages 567–576, 1995.

5 Yonatan Aumann and Yuval Rabani. An $O(\log k)$ approximate min-cut max-flow theorem and approximation algorithm. *SIAM J. Comput.*, 27(1):291–301, 1998.

6 Baruch Awerbuch, Rainer Gawlick, Tom Leighton, and Yuval Rabani. On-line admission control and circuit routing for high performance computing and communication. In *Proc. FOCS 1994*, pages 412–423, 1994.

7 Vineet Bafna, Piotr Berman, and Toshihiro Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM J. Discrete Math.*, 12(3):289–297 (electronic), 1999.

8 Hans L. Bodlaender, Stéphan Thomassé, and Anders Yeo. Kernel bounds for disjoint cycles and disjoint paths. *Theoret. Comput. Sci.*, 412(35):4570–4578, 2011. `doi:10.1016/j.tcs.2011.04.039`.

9 Andrei Z. Broder, Alan M. Frieze, Stephen Suen, and Eli Upfal. Optimal construction of edge-disjoint paths in random graphs. *SIAM J. Comput.*, 28(2):541–573 (electronic), 1999.

10 Andrei Z. Broder, Alan M. Frieze, and Eli Upfal. Existence and construction of edge-disjoint paths on expander graphs. *SIAM J. Comput.*, 23(5):976–989, 1994.

11 Chandra Chekuri and Alina Ene. Poly-logarithmic approximation for maximum node disjoint paths with constant congestion. In *Proc. SODA 2013*, pages 326–341, 2013.

12 Chandra Chekuri, Sanjeev Khanna, and F. Bruce Shepherd. An $\mathcal{O}(\sqrt{n})$ approximation and integrality gap for disjoint paths and unsplittable flow. *Theory Comput.*, 2:137–146, 2006. `doi:10.4086/toc.2006.v002a007`.

13 Chandra Chekuri, Sanjeev Khanna, and F Bruce Shepherd. A note on multiflows and treewidth. *Algorithmica*, 54(3):400–412, 2009.

14 Chandra Chekuri, Marcelo Mydlarz, and F. Bruce Shepherd. Multicommodity demand flow in a tree and packing integer programs. *ACM Trans. Algorithms*, 3(3):Art. 27, 23, 2007.

15 Chandra Chekuri, Guyslain Naves, and F. Bruce Shepherd. Maximum edge-disjoint paths in k-sums of graphs. In *Proc. ICALP 2013*, volume 7965 of *Lecture Notes Comput. Sci.*, pages 328–339, 2013.

16 Chandra Chekuri, Guyslain Naves, and F. Bruce Shepherd. Maximum edge-disjoint paths in k-sums of graphs, 2013. URL: `http://arxiv.org/abs/1303.4897`.

17 Chandra Chekuri, F. Bruce Shepherd, and Christophe Weibel. Flow-cut gaps for integer and fractional multiflows. *J. Comb. Theory, Ser. B*, 103(2):248–273, 2013.

18 Julia Chuzhoy. Routing in undirected graphs with constant congestion. In *Proc. STOC 2012*, pages 855–874, 2012.

19 Julia Chuzhoy, David H. K. Kim, and Shi Li. Improved approximation for node-disjoint paths in planar graphs. In *Proc. STOC 2016*, 2016. to appear.

20 Julia Chuzhoy and Shi Li. A polylogarithmic approximation algorithm for edge-disjoint paths with congestion 2. In *Proc. FOCS 2012*, pages 233–242, 2012.

21 Alina Ene, Matthias Mnich, Marcin Pilipczuk, and Andrej Risteski. On routing disjoint paths in bounded treewidth graphs. In *Proc. SWAT 2016*, LIPIcs, 2016. to appear.

22 Krzysztof Fleszar, Matthias Mnich, and Joachim Spoerhase. New algorithms for maximum disjoint paths based on tree-likeness, 2016. URL: `http://arxiv.org/abs/1603.01740`.

23 Alan M. Frieze. Edge-disjoint paths in expander graphs. *SIAM J. Comput.*, 30(6):1790–1801 (electronic), 2001.

24 Naveen Garg, Vijay V. Vazirani, and Mihalis Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997. `doi:10.1007/BF02523685`.

25 Oktay Günlük. A new min-cut max-flow ratio for multicommodity flows. *SIAM J. Discrete Math.*, 21(1):1–15, 2007.

26 Ian Holyer. The NP-completeness of edge-coloring. *SIAM J. Comput.*, 10(4):718–720, 1981.

27 Richard M. Karp. On the computational complexity of combinatorial problems. *Networks*, 5:45–68, 1975.

28 Ken-ichi Kawarabayashi and Yusuke Kobayashi. Breaking $\mathcal{O}(n^{1/2})$-approximation algorithms for the edge-disjoint paths problem with congestion two. In *Proc. STOC 2011*, pages 81–88, 2011.

29 Ken-ichi Kawarabayashi and Paul Wollan. A shorter proof of the graph minor algorithm: the unique linkage theorem. In *Proc. STOC 2010*, pages 687–694, 2010.

30 Jon Kleinberg and Ronitt Rubinfeld. Short paths in expander graphs. In *Proc. FOCS 1996*, pages 86–95, 1996.

31 Jon Kleinberg and Éva Tardos. Disjoint paths in densely embedded graphs. In *Proc. FOCS 1995*, pages 52–61, 1995.

32 Jon Kleinberg and Éva Tardos. Approximations for the disjoint paths problem in high-diameter planar networks. *J. Comput. System Sci.*, 57(1):61–73, 1998.

33 Stavros G. Kolliopoulos and Clifford Stein. Approximating disjoint-path problems using packing integer programs. *Math. Program.*, 99(1):63–87, 2004.

34 Tom Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46(6):787–832, 1999.

35 Nathan Linial, Eran London, and Yuri Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, 1995.

36 Daniel Lokshtanov, M. S. Ramanujan, and Saket Saurabh. Linear time parameterized algorithms for subset feedback vertex set. In *Proc. ICALP 2015*, pages 935–946, 2015.

37 Takao Nishizeki, Jens Vygen, and Xiao Zhou. The edge-disjoint paths problem is NP-complete for series-parallel graphs. *Discrete Appl. Math.*, 115(1-3):177–186, 2001. `doi:10.1016/S0166-218X(01)00223-2`.

38 Prabhakar Raghavan and Clark D. Tompson. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987.

39 Satish Rao and Shuheng Zhou. Edge disjoint paths in moderately connected graphs. *SIAM J. Comput.*, 39(5):1856–1887, 2010.

40 Neil Robertson and P. D. Seymour. Graph minors. XIII. The disjoint paths problem. *J. Combin. Theory Ser. B*, 63(1):65–110, 1995. `doi:10.1006/jctb.1995.1006`.

41 Petra Scheffler. A practical linear time algorithm for disjoint paths in graphs with bounded tree-width. Technical Report TR 396/1994, FU Berlin, Fachbereich 3 Mathematik, 1994.

42 Loïc Séguin-Charbonneau and F. Bruce Shepherd. Maximum edge-disjoint paths in planar graphs with congestion 2. In *Proc. FOCS 2011*, pages 200–209, 2011.

Streaming Property Testing of Visibly Pushdown Languages*

Nathanaël François[1], Frédéric Magniez[2], Michel de Rougemont[3], and Olivier Serre[4]

1 **Fakultät für Informatik, TU Dortmund, Germany**
2 **CNRS, IRIF, Univ Paris Diderot, Sorbonne Paris-Cité, France**
3 **University of Paris II and IRIF, CNRS, France**
4 **CNRS, IRIF, Univ Paris Diderot, Sorbonne Paris-Cité, France**

Abstract

In the context of formal language recognition, we demonstrate the superiority of streaming property testers against streaming algorithms and property testers, when they are not combined. Initiated by Feigenbaum *et al.*, a streaming property tester is a streaming algorithm recognizing a language under the property testing approximation: it must distinguish inputs of the language from those that are ε-far from it, while using the smallest possible memory (rather than limiting its number of input queries). Our main result is a streaming ε-property tester for visibly pushdown languages (VPL) with memory space $\mathrm{poly}((\log n)/\varepsilon)$.

Our construction is done in three steps. First, we simulate a visibly pushdown automaton in one pass using a stack of small height but whose items can be of linear size. In a second step, those items are replaced by small sketches. Those sketches rely on a notion of suffix-sampling we introduce. This sampling is the key idea for taking benefit of both streaming algorithms and property testers in the third step. Indeed, the last step relies on a (non-streaming) property tester for weighted regular languages based on a previous tester by Alon *et al.* This tester can directly be used for streaming testing special cases of instances of VPL that are already hard for both streaming algorithms and property testers. We then use it to decide the correctness of completed items, given their sketches, before removing them from the stack.

1998 ACM Subject Classification F.1.2 Modes of Computation, F.4.3 Formal Languages

Keywords and phrases Streaming Algorithm, Property Testing, Visibly Pushdown Languages

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.43

1 Introduction

We focus on streams representing data with both a linear ordering and a hierarchically nested matching of items. Data with such dual linear-hierarchical structure arise in various context, *e.g.* in semi-structured data management when handling HTML/XML documents or in program analysis when considering executions of recursive programs. Regular languages, as recognized by finite state automata, revealed a natural and successful tool to express properties of streams but lack the ability to handle the hierarchical structure. Context-free languages easily capture the latter but turn out to be too expressive hence, quickly lead to intractable complexity. In contrast, visibly pushdown languages (VPL) [6] while encompassing regular languages, enjoy most of its good properties and permit to handle

* Partially supported by the French ANR projects ANR-12-BS02-005 (RDAM) and ANR-14-CE25-0017 (AGREG). A full version of this paper is available at `http://arxiv.org/abs/1505.03334`.

data with both a linear and a hierarchical structure. In the context of semi-structured documents, they are closely related with regular languages of unranked trees as captured by hedge automata: indeed, a well-known result [3] states that, when the tree is given by its depth-first traversal, such automata correspond to visibly pushdown automata (Vpa) (see *e.g.* [18] for an overview on automata and logic for unranked trees). In databases, this word encoding of XML document is known as SAX representation: the document is a linear sequence of text characters, along with a hierarchically nested matching of open-tags with closing tags. Numerous popular subclasses of XML documents (*e.g.* those satisfying a given DTD specifications) are subclasses of Vpl. In program analysis, Vpa permit to capture natural properties of execution traces of recursive finite-state programs. For such programs, desirable specifications are expressed on the call-stack (*e.g.* "a module A should be invoked only if the module B belongs to the call-stack"): such properties can be expressed in the temporal logic of calls and returns (CaRet) [5, 4] that itself is captured by Vpa. Hence, the analysis of execution traces boils down to check membership in a Vpl.

Therefore, the study of Vpl is central to understand how massive semi-structured data (*e.g.* large semi-structured documents or execution traces) can be analyzed by sublinear algorithms, such as streaming algorithms and property testers.

Historically, Vpl got several names such as input-driven languages or, more recently, languages of nested words. Intuitively, a Vpa is a pushdown automaton whose actions on stack (push, pop or nothing) are solely decided by the currently read symbol. As a consequence, symbols can be partitioned into three groups: push, pop and neutral symbols. The complexity of Vpl recognition has been addressed in various computational models. The first results go back to the design of logarithmic space algorithms [11] as well as NC^1-circuits [13]. Later on, other models motivated by the context of massive data were considered, such as streaming algorithms and property testers (described below).

Streaming algorithms (see *e.g.* [22]) have only a sequential access to their input, on which they can perform a single pass, or sometimes a small number of additional passes. The size of their internal (random access) memory is the crucial complexity parameter, which should be sublinear in the input size, and even polylogarithmic if possible. The area of streaming algorithms has experienced tremendous growth in many applications since the late 1990s. The analysis of Internet traffic [2], in which traffic logs are queried, was one of their first applications. Nowadays, they have found applications with big data, notably to test graphs properties, and more recently in language recognition on very large inputs. The streaming complexity of language recognition has been firstly considered for languages that arise in the context of memory checking [8, 12], of databases [28, 27], and later on for formal languages [20, 7]. However, even for simple Vpl, any randomized streaming algorithm with p passes requires memory $\Omega(n/p)$, where n is the input size [17].

As opposed to streaming algorithms, (standard) property testers [9, 10, 16] have random access to their input but in the query model. They must query each piece of the input they need to access. They should sample only a sublinear fraction of their input, and ideally make a constant number of queries. In order to make the task of verification possible, decision problems need to be approximated as follows. Given a distance on words, an ε-tester for a language L distinguishes with high probability the words in L from those ε-far from L, using as few queries as possible. Property testing of regular languages was first considered for the Hamming distance [1]. When the distance allows sufficient modifications of the input, such as moves of arbitrarily large factors, it has been shown that any context-free language becomes testable with a constant number of queries [19, 15]. However, for more realistic distances, property testers for simple languages require a large number of queries, especially if they have one-sided error only. For example the complexity of an ε-tester for well-parenthesized expressions with two types of parentheses is between $\Omega(n^{1/11})$ and $O(n^{2/3})$ [25], and it becomes linear, even for one type of parentheses, if we require one-sided error [1]. The difficulty of testing regular tree languages was also addressed when the tester can directly query the tree structure [23, 24].

Faced by the intrinsic hardness of VPL in both streaming and property testing, we study the complexity of *streaming property testers* of formal languages, a model of algorithms combining both approaches. Such testers were historically introduced for testing specific problems (groupedness) [14] relevant for network data. They were later studied in the context of testing the insert/extract-sequence of a priority-queue structure [12]. We extend these studies to classes of problems. A streaming property tester is a streaming algorithm recognizing a language under the property testing approximation: it must distinguish inputs of the language from those that are ε-far from it, while using the smallest possible memory (rather than limiting its number of input queries). Such an algorithm can simulate any standard non-adaptive property tester. Moreover, we will see that, using its full scan of the input, it can construct better sketches than in the query model.

In this paper, we consider a natural notion of distance for VPL, the *balanced-edit distance*, which refines the edit distance on *balanced words* (where for each push symbol there is a matching pop symbol at the same height of the stack, and conversely). It can be interpreted as the edit distance on trees when trees are encoded as balanced words. Neutral symbols can be deleted/inserted, but any push symbol can only be deleted/inserted together with its matching pop symbol. Since our distance is larger than the standard edit distance, our testers are also valid for the edit distance.

In Section 3, we first design an exact algorithm that maintains a small stack but whose items can be of linear size as opposed to the standard simulation of a pushdown automaton which usually has a stack of possible linear size but with constant size items. In our algorithm, stack items are prefixes of some peaks (which we call unfinished peaks), where a *peak* is a balanced factor whose push symbols appear all before the first pop symbol. Our algorithm compresses an unfinished peak $u = u_+v_-$ when it is followed by a long enough sequence. More precisely, the compression applies to the peak v_+v_- obtained by disregarding part of the prefix of push sequence u_+. Those peaks are then inductively replaced, and therefore compressed, by the state-transition relation they define on the given automaton. The relation is then considered as a single symbol whose weight is the size of the peak it represents. In addition, to maintain a stack of logarithmic depth, one of the crucial properties of our algorithm (**Proposition 6**) is rewriting the input word as a peak formed by potentially a linear number of intermediate peaks, but with only a logarithmic number of nested peaks.

In Section 4, for the case of a single peak, we show how to sketch the current unfinished peak of our algorithm. The simplicity of those instances will let us highlight our first idea. Moreover, they are already expressive enough in order to demonstrate the superiority of streaming testers against streaming algorithms and property testers, when they are not combined. We first reduce the problem of streaming testing such instances to the problem of testing regular languages in the standard model of property testing (**Theorem 16**). Since our reduction induces weights on the letters of the new input word, we need a tester for weighted regular languages. Such a property tester has previously been devised in [24] extending constructions for unweighted regular languages [1, 23]. However, we consider a slightly simpler construction that could be of independent interest. As a consequence we get a streaming property tester with polylogarithmic memory for recognizing peak instances of any given VPL (**Theorem 17**), a task already hard for streaming algorithms and property testers (**Fact 8**).

In Section 5, we construct our main tester for a VPL L given by some VPA. For this we introduce a more involved notion of sketches made of a polylogarithmic number of samples. They are based on a new notion of suffix sampling (**Definition 18**). This sampling consists in a decomposition of the string into an increasing sequence of suffixes, whose weights increase geometrically. Such a decomposition can be computed online on a data stream, and one can maintain samples in each suffix of the decomposition using a standard reservoir sampling. This suffix decomposition will allow us to simulate an appropriate sampling on the peaks we compress, even if we do not yet know where they start. Our sampling can be used to perform an approximate computation of the compressed relation by our new property tester of weighted regular languages which we also used for single peaks. We first establish a result of stability which basically states that we can assume that our algorithm knows in advance where the peak it will compress starts (**Lemma 22**). Then we prove the robustness of our algorithm: words that are ε-far from L are rejected with high probability (**Lemma 23**). As a consequence, we get a one-pass streaming ε-tester for L with one-sided error η and memory space $\mathrm{O}(m^5 2^{3m^2}(\log n)^6(\log 1/\eta)/\varepsilon^4)$, where m is the number of states of a VPA recognizing L (**Theorem 20**).

2 Definitions and Preliminaries

Let $\mathbb{N}^*$ be the set of positive integers, and for any $n \in \mathbb{N}^*$, let $[n] = \{1, 2, \dots, n\}$. A t-subset of a set S is any subset of S of size t. For a finite alphabet Σ we denote the set of finite words over Σ by Σ^*. We denote by $u \cdot v$ (or simply uv) the word obtained by concatenating u and v. For a word $u = u(1)u(2)\cdots u(n)$, we call n the *length* of u, and $u(i)$ the ith letter in u. A *factor* of u is a word $u[i, j] = u(i)u(i+1)\cdots u(j)$ with $1 \le i \le j \le n$. When we mention letters and factors of u we implicitly also mention their positions in u. We say that v is a *sub-factor* of v', denoted $v \le v'$, if $v = u[i, j]$ and $v' = u[i', j']$ with $[i, j] \subseteq [i', j']$. Similarly we say that $v = v'$ if $[i, j] = [i', j']$. If $i \le i' \le j \le j'$ we say that the *overlap* of v and v' is $u[i', j]$. If v is a sub-factor of v' then the overlap of v and v' is v. Given two multisets of factors S and S', we say that $S \le S'$ if there is an injection $f : S \mapsto S'$ such that for each factor $v \in S$, $v \le f(v)$.

2.1 Weighted Words and Sampling

A *weight function* on a word u with n letters is a function $\lambda : [n] \to \mathbb{N}^*$ on the letters of u, whose value $\lambda(i)$ is called the *weight of* $u(i)$. A *weighted word* over Σ is a pair (u, λ) where $u \in \Sigma^*$ and λ is a weight function on u. We define $|u(i)| = \lambda(i)$ and $|u[i, j]| = \lambda(i) + \lambda(i+1) + \ldots + \lambda(j)$. The length of (u, λ) is the length of u. For simplicity, we will denote by u the weighted word (u, λ). Weighted letters will be used to substitute factors of same weights.

Our algorithms will be based on sampling of small factors according to their weights. We introduce a very specific notion adapted to our setting. For a weighted word u, we denote by *k-factor sampling on u* the sampling over factors $u[i, i + l]$ with probability $|u(i)|/|u|$, where $l \ge 0$ is the smallest integer such that $|u[i, i + l]| \ge k$ if it exists, otherwise l is such that $i + l$ is the last letter of u. More generally, we call *k-factor* such a factor. For the special case of $k = 1$, we call this sampling a *letter sampling on u*. In fact the general case $k > 1$ simply reduces to $k = 1$. Indeed, simply observe that k-factor sampling can be obtained from letter sampling by sampling on the first letters of the factors and online completing any sampled letter to produce its associated k-factor. Therefore, from now on, we only focus on how to perform letter samplings, that we implicitly extend to samplings on k-factors when

Algorithm 1 Reservoir Sampling

```
Input: Data stream $u$, Integer $t > 1$ standing for the number of samples
Data structure:
  $\sigma \leftarrow 0$ // Current weight of the processed stream
  $S \leftarrow$ empty multiset // Multiset of sampled letters
Code:
$a \leftarrow \mathrm{Next}(u)$, $\sigma \leftarrow |a|$
$S \leftarrow t$ copies of $a$
While $u$ not finished
    $a \leftarrow \mathrm{Next}(u)$, $\sigma \leftarrow \sigma + |a|$
    For each $b \in S$
        Replace $b$ by $a$ with probability $|a|/\sigma$
Output $S$
```

required. In particular, without further constraints, letter sampling can be implemented using a standard reservoir sampling (see Algorithm 1).

Even if our algorithm will require several samples from a k-factor sampling, we will often only be able to simulate this sampling by sampling either larger factors, more factors, or both. We introduce the notion of *over-sampling* to formalize this:

▶ **Definition 1.** Let $\mathcal{W}_1$ be a sampler producing a random multiset S_1 of factors of some given weighted word u. Then $\mathcal{W}_2$ *over-samples* $\mathcal{W}_1$ if it produces a random multiset S_2 of factors of u such that for each factor v of u, we have $\Pr(\exists v' \in S_2$ such that v is a factor of $v') \geq \Pr(\exists v' \in S_1$ such that v is a factor of $v')$.

2.2 Finite State Automata and Visibly Pushdown Automata

A *finite state automaton* is a tuple of the form $\mathcal{A} = (Q, \Sigma, Q_{in}, Q_f, \Delta)$ where Q is a finite set of control states, Σ is a finite input alphabet, $Q_{in} \subseteq Q$ is a subset of initial states, $Q_f \subseteq Q$ is a subset of final states and $\Delta \subseteq Q \times \Sigma \times Q$ is a transition relation. We write $p \xrightarrow{u} q$, to mean that there is a sequence of transitions in $\mathcal{A}$ from p to q while processing u, and we call (p, q) a u-*transition*. A word u is accepted if $q_{in} \xrightarrow{u} q_f$ for some $q_{in} \in Q_{in}$ and $q_f \in Q_f$. The language $L(\mathcal{A})$ of $\mathcal{A}$ is the set of words accepted by $\mathcal{A}$, and we refer to such a language as a *regular language*. For $\Sigma' \subseteq \Sigma$, the Σ'-*diameter* (or simply *diameter* when $\Sigma' = \Sigma$) of $\mathcal{A}$ is the maximum over all possible pairs $(p, q) \in Q^2$ of $\min\{|u| : p \xrightarrow{u} q$ and $u \in \Sigma'^*\}$, whenever this minimum is not over an empty set. We say that $\mathcal{A}$ is Σ'-*closed*, when $p \xrightarrow{u} q$ for some $u \in \Sigma^*$ if and only if $p \xrightarrow{u'} q$ for some $u' \in \Sigma'^*$.

A *pushdown alphabet* is a triple $\langle \Sigma_+, \Sigma_-, \Sigma_= \rangle$ that comprises three disjoint finite alphabets: Σ_+ is a finite set of *push symbols*, Σ_- is a finite set of *pop symbols*, and $\Sigma_=$ is a finite set of *neutral symbols*. For any such triple, let $\Sigma = \Sigma_+ \cup \Sigma_- \cup \Sigma_=$. Intuitively, a *visibly pushdown automaton* [26] over $\langle \Sigma_+, \Sigma_-, \Sigma_= \rangle$ is a pushdown automaton restricted so that it pushes onto the stack only on reading a push, it pops the stack only on reading a pop, and it does not modify the stack on reading a neutral symbol. Up to coding, this notion is similar to the one of input driven pushdown automata [21] and of nested word automata [6].

▶ **Definition 2.** A *visibly pushdown automaton* (VPA) over $\langle \Sigma_+, \Sigma_-, \Sigma_= \rangle$ is a tuple $\mathcal{A} = (Q, \Sigma, \Gamma, Q_{in}, Q_f, \Delta)$ where Q is a finite set of states, $Q_{in} \subseteq Q$ is a set of initial states, $Q_f \subseteq Q$ is a set of final states, Γ is a finite stack alphabet, and $\Delta \subseteq (Q \times \Sigma_+ \times Q \times \Gamma) \cup (Q \times \Sigma_- \times \Gamma \times Q) \cup (Q \times \Sigma_= \times Q)$ is the transition relation.

To represent stacks we use a special bottom-of-stack symbol $\perp$ that is not in Γ. A *configuration* of a VPA $\mathcal{A}$ is a pair (σ, q), where $q \in Q$ and $\sigma \in \perp \cdot \Gamma^*$. For $a \in \Sigma$, there is an *a-transition* from a configuration (σ, q) to (σ', q'), denoted $(\sigma, q)\xrightarrow{a}(\sigma', q')$, in the following cases:

- If a is a push symbol, then $\sigma' = \sigma\gamma$ for some $(q, a, q', \gamma) \in \Delta$, and we write $q\xrightarrow{a}(q', \mathsf{push}(\gamma))$.
- If a is a pop symbol, then $\sigma = \sigma'\gamma$ for some $(q, a, \gamma, q') \in \Delta$, and we write $(q, \mathsf{pop}(\gamma))\xrightarrow{a}q'$.
- If a is a neutral symbol, then $\sigma = \sigma'$ and $(q, a, q') \in \Delta$, and we write $q\xrightarrow{a}q'$.

For a finite word $u = a_1 \cdots a_n \in \Sigma^*$, if $(\sigma_{i-1}, q_{i-1})\xrightarrow{a_i}(\sigma_i, q_i)$ for every $1 \leq i \leq n$, we also write $(\sigma_0, q_0)\xrightarrow{u}(\sigma_n, q_n)$. The word u is *accepted* by a VPA if there is $(p, q) \in Q_{in} \times Q_f$ such that $(\perp, p)\xrightarrow{u}(\perp, q)$. The language $L(\mathcal{A})$ of $\mathcal{A}$ is the set of words accepted by $\mathcal{A}$, and we refer to such a language as a *visibly pushdown language* (VPL).

At each step, the height of the stack is pre-determined by the prefix of u read so far. The *height* $\text{height}(u)$ of $u \in \Sigma^*$ is the difference between the number of its push symbols and of its pop symbols. A word u is *balanced* if $\text{height}(u) = 0$ and $\text{height}(u[1, i]) \geq 0$ for all i. We also say that a push symbol $u(i)$ *matches* a pop symbol $u(j)$ if $\text{height}(u[i, j]) = 0$ and $\text{height}(u[i, k]) > 0$ for all $i < k < j$. By extension, the height of $u(i)$ is $\text{height}(u[1, i-1])$ when $u(i)$ is a push symbol, and $\text{height}(u[1, i])$ otherwise.

For all balanced words u, the property $(\sigma, p)\xrightarrow{u}(\sigma, q)$ does not depend on σ, therefore we simply write $p\xrightarrow{u}q$, and say that (p, q) is a *u-transition.* We also define similarly to the notions for finite automata above the Σ'*-diameter* of $\mathcal{A}$ (or simply diameter) and the notion of $\mathcal{A}$ being Σ'*-closed.* These definitions only consider balanced words.

Our model is inherently restricted to input words having no prefix of negative stack height, and we defined acceptance with an empty stack. This implies that only balanced words can be accepted. From now on, we assume that the input is balanced as verifying this in a streaming context is easy.

2.3 Streaming Property Testers

Assume we have, for any $\varepsilon > 0$, a criterion to declare that an input u is ε-far from a language L. An ε-tester for L accepts all inputs in L with probability 1 and rejects with high probability all inputs ε-far from L. Two-sided error testers have also been studied but in this paper we stay with the notion of one-sided testers, that we adapt in the context of streaming algorithm as in [14].

▶ **Definition 3.** Let $\varepsilon > 0$ and let L be a language. A *streaming ε-tester* for L with one-sided error η and memory $s(n)$ is a randomized algorithm A such that, for any input u of length n given as a data stream:

- If $u \in L$, then A accepts with probability 1;
- If u is ε-far from L, then A rejects with probability at least $1 - \eta$;
- A processes u within a single sequential pass while maintaining a memory

Even if we only focus on the space complexity of streaming testers, all our streaming testers have polylogarithmic (in n/ε) time per processing letter.

For a distance d between words, we say that a word u is ε-far from a language L if $d(u, v) > \varepsilon|u|$ for every $v \in L$, *i.e.* the ε-neighborhood of u does not intersect L. Hence, any distance on words leads to a notion of streaming property tester. Remark that any ε-tester for some distance d_1 turns out to be also a $(c\varepsilon)$-tester for any other distance d_2 such that $d_2 \leq cd_1$, where $c > 0$ is some constant.

2.4 Balanced/Standard Edit Distance

The usual distance between words in property testing is the Hamming distance. In this work, we consider an easier distance to manipulate in property testing but still relevant for most applications, which is the edit distance, that we adapt to weighted words.

Given a word u, we define two possible *edit operations*: the *deletion* of a letter in position i with corresponding cost $|u(i)|$, and its converse operation, the *insertion* where we also select a weight for the new $u(i)$. Note that, for simplicity, we drop the usual substitution operation, leading to a possible multiplicative factor of 2 in the resulting distance. This is not an issue when designing streaming property testers as observed above. The *(standard) edit distance* $\mathsf{dist}(u,v)$ between two weighted words u and v is defined as the minimum total cost of a sequence of edit operations changing u to v. All letters that have not been inserted nor deleted must keep the same weight. For a restricted set of letters Σ', define $\mathsf{dist}_{\Sigma'}(u,v)$ when insertions (but not deletions) are restricted to letters in Σ' (this makes $\mathsf{dist}_{\Sigma'}$ not symmetric).

We will also consider a restricted version of this distance for balanced words, motivated by our study of VPL. Similarly, *balanced-edit operations* can be deletions or insertions of letters, but each deletion of a push symbol (resp. pop symbol) requires the deletion of the matching pop symbol (resp. push symbol). Similarly for insertions: if a push (resp. pop) symbol is inserted, then a matching pop (resp. push) symbol must also be inserted simultaneously. The cost of these operations is the weight of the affected letters, as with the edit operations. We define the *balanced-edit distance* $\mathsf{bdist}(u,v)$ between two balanced words as the total cost of a sequence of balanced-edit operations changing u to v. Similarly to $\mathsf{dist}_{\Sigma'}(u,v)$ we define $\mathsf{bdist}_{\Sigma'}(u,v)$. We omit Σ' when $\Sigma' = \Sigma$.

When dealing with a visibly pushdown language, we will always use the balanced-edit distance, whereas we will use the standard-edit distance for regular languages. Note that since balanced-edit distance is larger than the standard edit distance, our testers will also be valid for that distance.

3 Exact Algorithm

Fix a VPA $\mathcal{A}$ recognizing some VPL L on $\Sigma = \Sigma_+ \cup \Sigma_- \cup \Sigma_=$. In this section, we design an exact streaming algorithm that decides whether an input belongs to L. Algorithm 2 maintains a stack of small height but whose items can be of linear size. In Section 5, we replace stack items by appropriate small sketches.

3.1 Notations and Algorithm Description

Call a *peak* a sequence of push symbols followed by an equal number of pop symbols, with possibly intermediate neutral symbols, *i.e.* an element of the language $\Lambda = \bigcup_{j\geq 0}((\Sigma_=)^* \cdot \Sigma_+)^j \cdot (\Sigma_=)^* \cdot (\Sigma_- \cdot (\Sigma_=)^*)^j$. One can compress any peak $v \in \Lambda$ by the set $R_v = \{(p,q) : p \xrightarrow{v} q\}$ of the v-transitions, and consider R_v as a new neutral symbol with weight $|v|$. In fact, for the purpose of the analysis of our algorithm, we augment neutral symbols by many more relations for which $\mathcal{A}$ remains Σ-closed. Indeed, we allow any relation R of any weight such that, when $(p,q) \in R$, there is a $v \in \Lambda$ such that $p \xrightarrow{v} q$, but that v could be different for every $(p,q) \in R$. For the rest of the paper, they will be the only symbols with weight potentially larger than 1.

▶ **Definition 4.** Let Σ_Q be $\Sigma_=$ augmented by all letters 'R' encoding a relation $R \subseteq Q \times Q$ such that for every $(p,q) \in R$ there is a balanced word $u \in \Sigma^*$ with $p \xrightarrow{u} q$. In addition we allow any weight $|R| \geq 1$ for those letters. Let Λ_Q be Λ where $\Sigma_=$ is replaced by Σ_Q.

Algorithm 2 Exact Tester for a VPL

```
Input: Balanced data stream $u$
Data structure:
$Stack \leftarrow$ empty stack // Stack of items $v$ with $v \in \mathrm{Prefix}(\Lambda_Q)$
$u_0 \leftarrow \emptyset$ // $u_0 \in \mathrm{Prefix}(\Lambda_Q)$ is a suffix of the processed part $u[1,i]$ of $u$
     // with possibly some factors $v \in \Lambda_Q$ replaced by $R_v$
$R_{\mathtt{temp}} \leftarrow \{(p,p)\}_{p\in Q}$ //Set of transitions for the max. prefix of $u[1,i]$ in $\Lambda_Q$
Code:
While $u$ not finished
  $a \leftarrow \mathrm{Next}(u)$ //Read and process a new symbol $a$
  If $a \in \Sigma_+$ and $u_0$ has a letter in $\Sigma_-$ // $u_0 \cdot a \notin \mathrm{Prefix}(\Lambda_Q)$
    Push $u_0$ on $Stack$, $u_0 \leftarrow a$
  Else $u_0 \leftarrow u_0 \cdot a$
  If $u_0$ is balanced // $u_0 \in \Lambda_Q$: compression
    Compute $R_{u_0}$ the set of $u_0$-transitions
    If $Stack = \emptyset$, then $R_{\mathtt{temp}} \leftarrow R_{\mathtt{temp}} \circ R_{u_0}$, $u_0 \leftarrow \emptyset$
      // where $\circ$ denotes the composition of relations
    Else Pop $v$ from $Stack$, $u_0 \leftarrow v \cdot R_{u_0}$
  Let $(v_1 \cdot v_2) \leftarrow \mathrm{top}(Stack)$ s.t. $v_2$ is maximal and balanced // $v_2 \in \Lambda_Q$
  If $|u_0| \geq |v_2|/2$  // $u_0$ is big enough and $v_2$ can be replaced by $R_{v_2}$
    Compute $R_{v_2}$ the $v_2$-transitions, Pop $v$ from Stack, $u_0 \leftarrow (v_1 \cdot R_{v_2}) \cdot u_0$
If $(Q_{in} \times Q_f) \cap R_{\mathtt{temp}} \neq \emptyset$, Accept; Else Reject // $R_{\mathtt{temp}} = R_u$
```

We then write $p\xrightarrow{R}q$ whenever $(p,q) \in R$, and extend $\mathcal{A}$ and L accordingly. Of course, our notion of distance will be solely based on the initial alphabet Σ. If $R_1, R_2 \subseteq Q \times Q$ are two relations on Q we define their composition $R_1 \circ R_2$ to be $\{(x,z) \mid \exists y \text{ s.t. } (x,y) \in R_1 \text{ and } (y,z) \in R_2\}$.

A general balanced input instance u will consist of many nested peaks. However, we will recursively replace each factor $v \in \Lambda_Q$ by R_v with weight $|v|$.

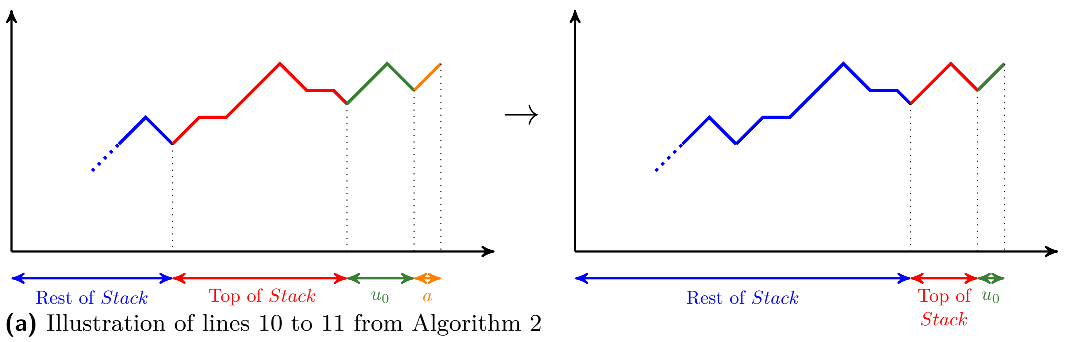
Denote by $\mathrm{Prefix}(\Lambda_Q)$ the language of prefixes of words in Λ_Q. While processing the prefix $u[1,i]$ of the data stream u, Algorithm 2 maintains a suffix $u_0 \in \mathrm{Prefix}(\Lambda_Q)$ of $u[1,i]$, that is an unfinished peak, with some simplifications of factors v in Λ_Q by their corresponding relation R_v. Therefore u_0 consists of a sequence of push symbols and neutral symbols possibly followed by a sequence of pop symbols and neutral symbols. The algorithm also maintains a subset $R_{\mathtt{temp}} \subseteq Q \times Q$ that is the set of transitions for the maximal prefix of $u[1,i]$ in Λ_Q. When the stream is over, the set $R_{\mathtt{temp}}$ is used to decide whether $u \in L$ or not.

When a push symbol a comes after a pop sequence, $u_0 \cdot a$ is no longer in $\mathrm{Prefix}(\Lambda_Q)$ hence, Algorithm 2 puts u_0 on the stack of unfinished peaks (see lines 10 to 11 and Figure 1a) and u_0 is reset to a. In other situations, it adds a to u_0. In case u_0 becomes a word in Λ_Q (see lines 13 to 17 and Figure 1b), Algorithm 2 computes the set of u_0-transitions $R_{u_0} \in \Sigma_Q$, and adds R_{u_0} to the previous unfinished peak that is retrieved on top of the stack and becomes the current unfinished peak; in the special case where the stack is empty it simply updates $R_{\mathtt{temp}}$ by taking its composition with R_{u_0}.

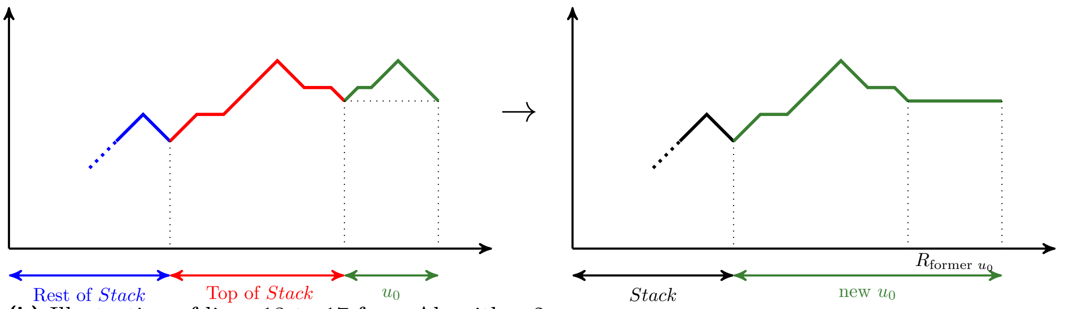
3.2 Algorithm Analysis

For each factor v constructed in Algorithm 2, we define $\mathrm{Depth}(v)$ as the number of processed nested peaks in v. This is formalized as follows.

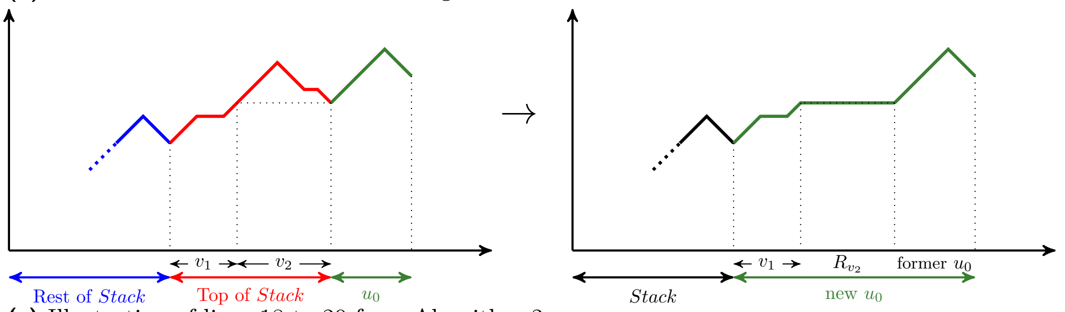
▶ **Definition 5.** For each factor constructed in Algorithm 2, Depth is defined dynamically by $\mathrm{Depth}(a) = 0$ when $a \in \Sigma$, $\mathrm{Depth}(v) = \max_i \mathrm{Depth}(v(i))$ and $\mathrm{Depth}(R_v) = \mathrm{Depth}(v) + 1$.



(a) Illustration of lines 10 to 11 from Algorithm 2



(b) Illustration of lines 13 to 17 from Algorithm 2



(c) Illustration of lines 18 to 20 from Algorithm 2

Figure 1 Illustration of Algorithm 2.

In order to bound the size of the stack, Algorithm 2 considers the maximal balanced suffix v_2 of the topmost element $v_1 \cdot v_2$ of the stack and, whenever $|u_0| \geq |v_2|/2$, it computes the relation R_{v_2} and continues with a bigger current peak starting with v_1 (see lines 18 to 20 and Figure 1c). A consequence of this compression is that the elements in the stack have geometrically decreasing weight and therefore the height of the stack used by Algorithm 2 is logarithmic in the length of the input stream. This can be proved by a direct inspection of Algorithm 2.

▶ **Proposition 6.** *Algorithm 2 accepts exactly when $u \in L$, while maintaining a stack of at most $\log |u|$ items.*

We state that Algorithm 2, when processing an input u of length n, considers at most $\mathrm{O}(\log n)$ nested peaks, that is $\mathrm{Depth}(v) = \mathrm{O}(\log n)$ for all factors constructed in Algorithm 2.

▶ **Lemma 7.** *Let v be the factor used to compute R_v at line either 14 or 20 of Algorithm 2. Then $|v(i)| \leq 2|v|/3$, for all i. Moreover, for any factor w constructed by Algorithm 2 it holds that $\mathrm{Depth}(w) = \mathrm{O}(\log |w|)$.*

4 The Special Case Of Peaks

We now consider restricted instances consisting of a *single peak*. For these instances, Algorithm 2 never uses its stack but u_0 can be of linear size. We show how to replace u_0 by a small random sketch in order to get a streaming property tester using polylogarithmic memory. In Section 5, this notion of sketch will be later extended to obtain our final streaming property tester for general instances.

4.1 Hard Peak Instances

Peaks are already hard for both streaming algorithms and property testers. Indeed, consider the language Disj $\subseteq \Lambda$ over alphabet $\Sigma = \{0,1,\overline{0},\overline{1},a\}$ and defined as the union of all languages $a^* \cdot x(1) \cdot a^* \cdot \ldots \cdot x(j) \cdot a^* \cdot \overline{y(j)} \cdot a^* \cdot \ldots \cdot \overline{y(1)} \cdot a^*$, where $j \geq 1$, $x, y \in \{0,1\}^j$, and $x(i)y(i) \neq 1$ for all i.

Then Disj can be recognized by a VPA with 3 states, $\Sigma_+ = \{0,1\}$, $\Sigma_- = \{\overline{0},\overline{1}\}$ and $\Sigma_= = \{a\}$. However, the following fact states its hardness for both models. The hardness for non-approximation streaming algorithms comes for a standard reduction to Set-Disjointness. The hardness for property testing algorithms is a corollary of a similar result due to [25] for parenthesis languages with two types of parentheses.

▶ **Fact 8.** *Any randomized p-pass streaming algorithm for* Disj *requires memory space $\Omega(n/p)$, where n is the input length. Moreover, any (non-streaming) (2^{-6})-tester for* Disj *requires to query $\Omega(n^{1/11}/\log n)$ letters of the input word.*

Surprisingly, for every $\varepsilon > 0$, we will show that languages of the form $L \cap \Lambda$, where L is a VPL, become easy to ε-test by streaming algorithms. This is mainly because, given their full access to the input, streaming algorithms can perform an input sampling which makes the property testing task easy, using only a single pass and little memory.

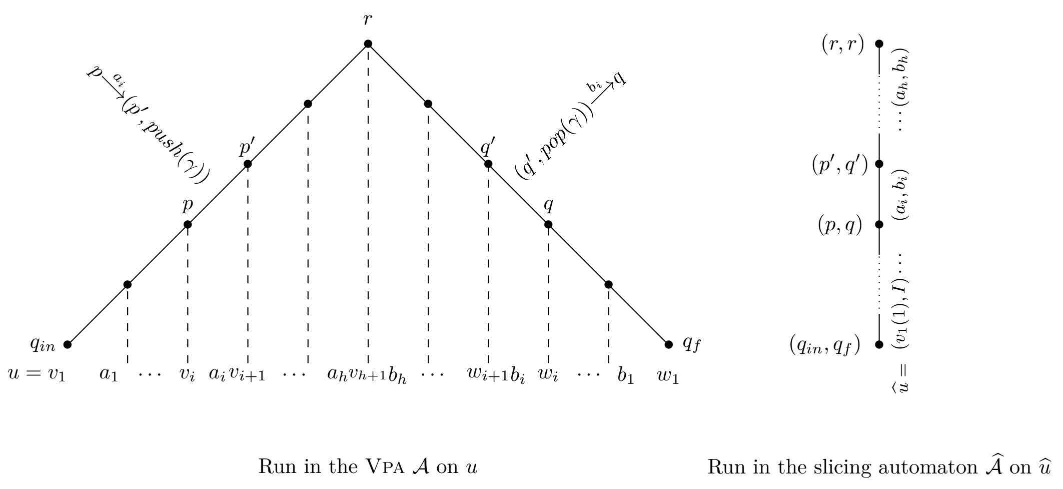
4.2 Slicing Automaton

Observe that Algorithm 2 will never use the stack in the case of a single peak. After Algorithm 2 has processed the i-th letter of the data stream, u_0 contains $u[1,i]$ where the eventual initial sequence of neutral symbols has been removed. We will show how to compute R_{u_0} at line 14 using a standard finite state automaton without any stack.

Indeed, for every VPL L, one can construct a regular language $\widehat{L}$ such that testing whether $u \in L \cap \Lambda$ is equivalent to test whether some other word $\widehat{u}$ belongs to $\widehat{L}$. For this, let I be a special symbol not in $\Sigma_=$ encoding the relation set $\{(p,p) : p \in Q\}$. For a word $v \in \Sigma_=^l$, write $[v,\mathrm{I}]$ for the word $(v(1),\mathrm{I}) \cdot (v(2),\mathrm{I}) \cdots (v(l),\mathrm{I})$, and similarly $[\mathrm{I},v]$. Consider a weighted word of the form $u = \left(\prod_{i=1}^{j} v_i \cdot a_i\right) \cdot v_{j+1} \cdot \left(\prod_{i=j}^{1} b_i \cdot w_i\right)$, where $a_i \in \Sigma_+$, $b_i \in \Sigma_-$, and $v_i, w_i \in \Sigma_=^*$. Then the *slicing* of u (see Figure 2) is the word $\widehat{u}$ over the alphabet $\widehat{\Sigma} = (\Sigma_+ \times \Sigma_-) \cup (\Sigma_= \times \{\mathrm{I}\}) \cup (\{\mathrm{I}\} \times \Sigma_=)$ defined by $\widehat{u} = \left(\prod_{i=1}^{j} [v_i,\mathrm{I}] \cdot [\mathrm{I},w_i] \cdot (a_i,b_i)\right) \cdot [v_{j+1},\mathrm{I}]$, and which has weight $\left(\sum_{i=1}^{j} \lambda(v_i) + \lambda(w_i) + 2\right) + \lambda(v_{j+1})$.

▶ **Definition 9.** Let $\mathcal{A} = (Q,\Sigma,\Gamma,Q_{in},Q_f,\Delta)$ be a VPA. Define $\widehat{Q} = Q \times Q$, $\widehat{Q_{in}} = Q_{in} \times Q_f$, $\widehat{Q_f} = \{(p,p) : p \in Q\}$. The *slicing* of $\mathcal{A}$ is the finite automaton $\widehat{\mathcal{A}} = (\widehat{Q},\widehat{\Sigma},\widehat{Q_{in}},\widehat{Q_f},\widehat{\Delta})$ where the transitions $\widehat{\Delta}$ are:

1. $(p,q)\xrightarrow{(a,b)}(p',q')$ when $p\xrightarrow{a}(p',\mathsf{push}(\gamma))$ and $(q',\mathsf{pop}(\gamma))\xrightarrow{b}q$ are both transitions of Δ.
2. $(p,q)\xrightarrow{(c,\mathrm{I})}(p',q)$, resp. $(p,q)\xrightarrow{(\mathrm{I},c)}(p,q')$, when $p\xrightarrow{c}p'$, resp. $q\xrightarrow{c}q'$, is a transition of Δ.



r
$p \xrightarrow{a_i} (p', push(\gamma))$
$(q', pop(\gamma)) \xrightarrow{b_i} q$
p'
q'
p
q
q_{in}
q_f
$u = v_1$ a_1 $\cdots$ v_i $a_i v_{i+1}$ $\cdots$ $a_h v_{h+1} b_h$ $\cdots$ $w_{i+1} b_i$ w_i $\cdots$ b_1 w_1
(r,r)
(p',q')
(p,q)
(q_{in},q_f)
$\widehat{u} = (v_1(1), I) \cdots (a_i, b_i) \cdots (a_h, b_h)$
Run in the VPA $\mathcal{A}$ on u
Run in the slicing automaton $\widehat{\mathcal{A}}$ on $\widehat{u}$

Figure 2 Slicing of a word $u \in \Lambda$.

This construction will be later used in Section 5 for weighted languages. In that case, we define the weight of a letter in $\widehat{u}$ by $|(a,b)| = |a| + |b|$, with the convention that $|I| = 0$. Moreover, we write $\widehat{\Sigma_Q}$ for the alphabet obtained similarly to $\widehat{\Sigma}$ using Σ_Q instead of $\Sigma_{=}$. Note that the slicing automaton $\widehat{\mathcal{A}}$ defined on $\widehat{\Sigma_Q}$ is $\widehat{\Sigma}$-closed and has $\widehat{\Sigma}$-diameter at most $2m^2$ where $m = |Q|$. Indeed, the slicing automaton has m^2 states and every letter in $\widehat{\Sigma}$ has weight at most 2, hence the shortest path from two states (when exists) has weight at most $2m^2$. In particular, it directly implies the following.

▶ **Proposition 10.** *Let $v \in \Lambda$ be such that $(p,q) \xrightarrow{\widehat{v}} (p',q')$. There is $w \in \Lambda$ such that $|w| \leq 2m^2$ and $(p,q) \xrightarrow{\widehat{w}} (p',q')$.*

▶ **Lemma 11.** *If $\mathcal{A}$ is a* VPA *accepting L, then $\widehat{\mathcal{A}}$ is accepts $\widehat{L} = \{\widehat{u} : u \in L \cap \Lambda\}$.*

4.3 Random Sketches

We are now ready to build a tester for $L \cap \Lambda$. To test a word u we use a property tester for the regular language $\widehat{L}$. Regular languages are known to be ε-testable for the Hamming distance with $\mathrm{O}((\log 1/\varepsilon)/\varepsilon)$ non-adaptive queries on the input word [1], that is queries that can all be made simultaneously. Those queries define a small random sketch of u that can be sent to the tester for approximating R_u. Since the Hamming distance is larger than the edit distance, those testers are also valid for the latter distance. Observe also that, for $v_1, v_2 \in \Lambda_Q$, we have $\mathsf{bdist}(v_1, v_2) \leq 2\mathsf{dist}(\widehat{v_1}, \widehat{v_2})$. The only remaining difficulty is to provide to the tester an appropriate sampling on $\widehat{u}$ while processing u.

We will proceed similarly for the general case in Section 5, but then we will have to consider weighted words. Therefore we show how to sketch u in that general case already. Indeed, the tester of [1] was simplified for the edit distance in [23], and later on adapted for weighted words in [24]. We consider here an alternative approach that we believe to be simpler, but slightly less efficient than the tester of [24].

Our tester for weighted regular languages is based on k-factor sampling on $\widehat{u}$ that we will simulate by an over-sampling built from a letter sampling on u, that is according to the weights of the letters of u only. This new sampling can be easily performed given a stream of u using a standard reservoir sampling.

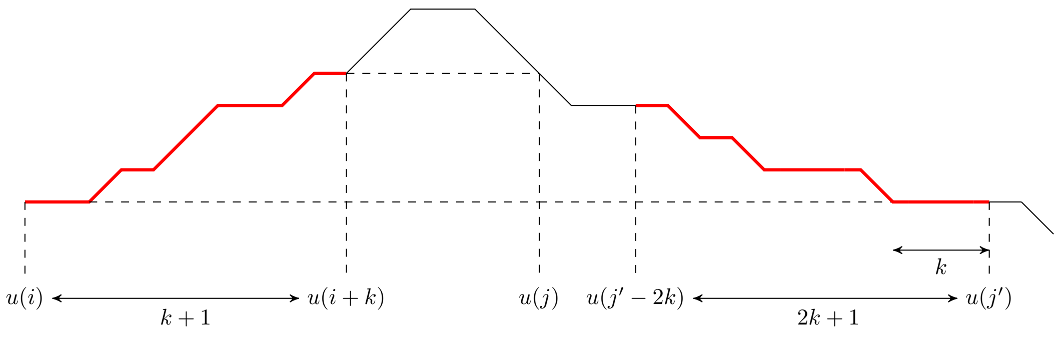


Figure 3 The sampling $\mathcal{W}_k(u)$ from Definition 12: sample is in red.

Let $u \in \Lambda$ and let $u[i, i+k]$ be a factor that contains at least one push symbol. Call i_1 (resp. i_2) the smallest (resp. largest) integer such that $i_1 \geq i$ (resp. $i_2 \leq i+k$) and $u(i_1)$ (resp. $u(i_2)$) is a push symbol. Then the *matching pop sequence* of $u[i, i+k]$ is defined as $u[j_1, j_2]$ where $u(j_1)$ (resp. $u(j_2)$) is the matching pop symbol of $u(i_1)$ (resp. $u(i_2)$).

▶ **Definition 12.** For a weighted word $u \in \Lambda_Q$, denote by $\mathcal{W}_k(u)$ the sampling over subwords of u constructed as follows (see Figure 3):
(1) Sample a factor $u[i, i+k]$ of u with probability $|u(i)|/|u|$.
(2) If $u[i, i+k]$ contains at least one push symbol, let $u[j, j']$ be the matching pop sequence of $u[i, i+k]$, extended by the first k neutral symbols after the last pop symbol, if any. Add $u[\max(j, j'-2k), j']$ to the sample (hence, some matching pops of $u[i, i+k]$ may not belong to $u[\max(j, j'-2k), j']$).

Let us stress that in the above definition the weight of letters only matter in (1), and not in (2) which cares about matching push and pop symbols, which are of weight 1. One consequence is that one can design a randomized streaming algorithm performing this sampling.

▶ **Fact 13.** *There is a randomized streaming algorithm with memory* $\mathrm{O}(k + \log n)$ *which, given* k *and* u *as input, samples* $\mathcal{W}_k(u)$.

▶ **Lemma 14.** *Let* u *be a weighted word, and let* k *be such that* $4k \leq |u|$. *Then* $4k$ *independent copies of* $\mathcal{W}_k(u)$ *over-sample the* k*-factor sampling on* $\widehat{u}$.

We can now give an analogue of the property tester for weighted regular languages in $L \cap \Lambda_Q$. For that, we use the following notion of approximation.

▶ **Definition 15.** Let $R \subseteq Q^2$ and $\varepsilon \geq 0$. Then R (ε, Σ)-*approximates* a balanced word $u \in (\Sigma_+ \cup \Sigma_- \cup \Sigma_Q)^*$ on $\mathcal{A}$, if for all $p, q \in Q$:
(1) If $p \xrightarrow{u} q$, then $(p, q) \in R$;
(2) If $(p, q) \in R$, there is a word v such that $\mathsf{dist}_\Sigma(u, v) \leq \varepsilon|u|$ and $p \xrightarrow{v} q$.

Our tester is going to be robust enough in order to consider samples that do not exactly match the peaks we want to compress.

▶ **Theorem 16.** *Let* $\mathcal{A}$ *be a* VPA *with* $m \geq 2$ *states and* Σ*-diameter* $d \geq 2$. *Let* $\varepsilon > 0$, $\eta > 0$, $t = 2\lceil 4dm^3(\log 1/\eta)/\varepsilon\rceil$, $k = \lceil 4dm/\varepsilon\rceil$ *and* $T = 4kt$. *There is an algorithm that, given* T *random subwords* $z_1, \ldots, z_T$ *of some weighted word* $v \in \Lambda_Q$, *such that each* z_i *comes from an independent sampling* $\mathcal{W}_k(v)$, *outputs a set* $R \subseteq Q \times Q$ *that* (ε, Σ)*-approximates* v *on* $\mathcal{A}$ *with bounded error* η.

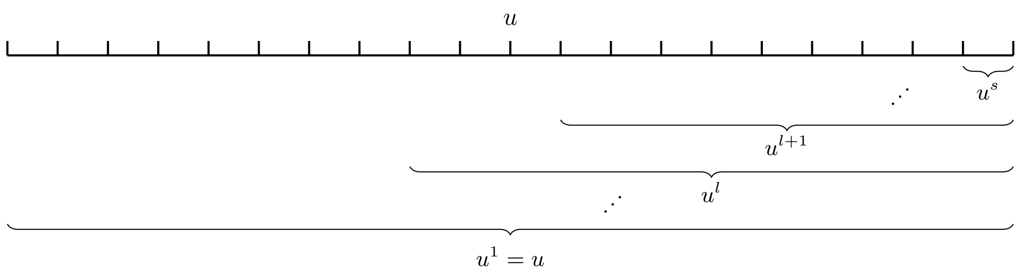


Figure 4 An α suffix decomposition of u of size s. For every l, either $|u^l| \leq \alpha|u^{l+1}|$, or $u^l = a \cdot u^{l+1}$ where a is a letter.

Let v' be obtained from v by at most $\varepsilon|v|$ balanced deletions. Then, the conclusion is still true if the algorithm is given an independent $\mathcal{W}_k(v')$ for each z_i instead, except that R now provides a $(3\varepsilon, \Sigma)$-approximation. Last, each sampling can be replaced by an over-sampling.

As a consequence we get our first streaming tester for $L \cap \Lambda$.

▶ **Theorem 17.** *Let $\mathcal{A}$ be a* VPA *for L with $m \geq 2$ states, and let $\varepsilon, \eta > 0$. Then there is a streaming ε-tester for $L \cap \Lambda$ with one-sided error η and memory space $O((m^8 \log(1/\eta)/\varepsilon^2)(m^3/\varepsilon + \log n))$, where n is the input length.*

Proof. We use Algorithm 2 where we replace the current factor u_0 by $T = 4kt$ independent samplings $\mathcal{W}_k(u_0)$. We know that such samplings can be computed using memory space $O(k + \log n)$ by Fact 13. By Proposition 10, the slicing automaton has $\widehat{\Sigma}$-diameter d at most $2m^2$. Therefore, from Theorem 16, taking $t = 4\lceil 4dm^3(\log 1/\eta)/\varepsilon\rceil$ and $k = \lceil 4dm/\varepsilon\rceil$ leads to the desired conclusion. ◀

5 Algorithm With Sketching

5.1 Sketching Using Suffix Samplings

We now describe the sketches used by our main algorithm. They are based on the generalization of the random sketches described in Section 4.3. Moreover, they rely on a notion of suffix sampling, that ensures a good letter sampling on each suffix of a data stream. Recall (see Section 2.1) that a letter sampling on a weighted word u samples a random letter $u(i)$ (with its position) with probability $|u(i)|/|u|$, and that a sampling on k-factors can be derived from a letter sampling. Therefore we will sample k-factors using an (α, t)-suffix sampling.

▶ **Definition 18.** Let u be a weighted word and let $\alpha > 1$. An *α-suffix decomposition of u of size s* (see Figure 4) is a sequence of suffixes $(u^l)_{1 \leq l \leq s}$ of u such that: $u^1 = u$, u^s is the last letter of u, and for all l, u^{l+1} is a strict suffix of u^l and if $|u^l| > \alpha|u^{l+1}|$ then $u^l = a \cdot u^{l+1}$ where a is a single letter.
An *(α, t)-suffix sampling on u of size s* is an α-suffix decomposition of u of size s with t letter samplings on each suffix of the decomposition.

We observe that (α, t)-suffix samplings can be either concatenated or compressed as stated below.

▶ **Proposition 19.** *Given an (α, t)-suffix sampling D_u on u of size s_u and another one D_v on v of size s_v, there is an algorithm **Concatenate**(D_u, D_v) computing an (α, t)-suffix sampling on the concatenated word $u \cdot v$ of size at most $s_u + s_v$ in time $O(s_u)$.*

Data Structure 3 Sketch for an unfinished peak

```
Parameters: real $\varepsilon' > 0$, integers $T \geq 1$ and $k \geq 1$.
Data structure for a weighted word $v \in \mathrm{Prefix}(\Lambda_Q)$
  Weights of $v$ and of its first letter $v(1)$
  Height of $v(1)$
  Boolean indicating whether $v$ contains a pop symbol
  $(1+\varepsilon')$-suffix decomposition $v^1, \ldots, v^s$ of $v$ encoded for $l = 1, \ldots, s$ by
     Estimates $|v^l|_{\mathtt{low}}$ and $|v^l|_{\mathtt{high}}$ of $|v^l|$
     $T$ independent samplings $S_{v^l}$ on $k$-factors of $v^l$ //See details
        below with corresponding weights and heights
```

Moreover, given an (α, t)-suffix sampling D_u on u of size s_u, there is an algorithm ***Simplify***(D_u) *computing an (α, t)-suffix sampling on u of size at most $2\lceil \log |u| / \log \alpha \rceil$ in time $\mathrm{O}(s_u)$.*

Proof. For **Concatenate**, it suffices to do the following. For each suffix u^l of D_u: (1) replace u^l by $u^l \cdot v$; and (2) replace the i-th sampling of u^l by the i-th sampling of v with probability $|v|/(|u| + |v|)$, for $i = 1, \ldots, t$.

For **Simplify**, do the following. For each suffix u^l of D_u, from $l = s_u$ (the smallest one) to $l = 1$ (the largest one): (1) replace all suffixes $u^{l-1}, u^{l-2}, \ldots, u^m$ by the largest suffix u^m such that $|u^m| \leq \alpha |u^l|$; and (2) suppress all samples from deleted suffixes. ◀

Using this proposition, one can easily design a streaming algorithm constructing online a suffix decomposition of polylogarithmic size. Starting with an empty suffix-sampling S, simply concatenate S with the next processed letter a of the stream, and then simplify it.

5.2 Final Algorithm

Our final algorithm is a modification of Algorithm 2: in particular it approximates relations R_v (in the spirit of Definition 15) by elements in Σ_Q, instead of exactly computing them. Let us stress that even if some R_v is approximated by an R that does not correspond to any R_u, one has $R \in \Sigma_Q$, which means that for any $(p, q) \in R$, there is a balanced word $u \in \Sigma^*$ depending on (p, q) with $p \xrightarrow{u} q$.

To mimic Algorithm 2 we need to encode (compactly) each unfinished peak v of the stack and u_0: for that we use the data structure described in Data Structure 3. Our final algorithm, Algorithm 4, is simply Algorithm 2 with this new data structure and corresponding adapted operations, where $\varepsilon' = \varepsilon/(6 \log n)$, $T = 4608 m^4 2^{2m^2} (\log^2 n)(\log 1/\eta)/\varepsilon^2$ and $k = 24 m 2^{m^2} (\log n)/\varepsilon$.

The methods are described in Algorithm 4, where we implicitly assume that each letter processed by the algorithm comes with its respective height and (exact or approximate) weight. They use functions **Concatenate** and **Simplify** described in Proposition 19, while adapting them.

In the next section, we show that the samplings S_{v^l} are close enough to an $(1+\varepsilon')$-suffix sampling on v^l. This lets us build an over-sampling of an $(1+\varepsilon')$-suffix sampling. We also show that it only requires a polylogarithmic number of samples. Then, we explain how to recursively apply the tester from Theorem 16 (with ε') in order to obtain the compressions at line 14 and 20 while keeping a cumulative error below ε. We now state our main result whose proof relies on Lemmas 22 and 23.

Algorithm 4 Adaptation of Algorithm 2 using sketches

```
Run Algorithm 2 using Data structure 3 with the following adaptations:
Adaption of functions from Proposition 19
 Concatenate($D_u, D_v$) with an exact estimate of $|v|$ is modified s.t.
   the replacement probability is now $|v|/(|u|_{\texttt{high}} + |v|)$
   and $|u^l \cdot v|_z \leftarrow |u^l|_z + |v|$, for $z = \texttt{low}, \texttt{high}$
 Simplify($D_u$) with $\alpha = 1 + \varepsilon'$ has now relaxed condition $|u^m|_{\texttt{high}} \leq (1+\varepsilon')|u^l|_{\texttt{low}}$
 Online-Suffix-Sampling is unchanged except for doing $k$-factor sampling.
Adaption of operations on factors used in Algorithm 2
 Compute relation: $R_v$
   Run the algorithm of Theorem 16 using samples in $D_v$
 Decomposition: $v_1 \cdot v_2 \leftarrow v$
  Find largest suffix $v^i$ in $D_v$ s.t. $v^i \in \mathrm{Prefix}(\Lambda_Q)$ //i.e. $v^i$ is in $v_2$
  $D_{v|v_1} \leftarrow$suffixes $(v^l)_{l<i}$ with their samples
  $D_{v_2} \leftarrow$suffix $v^i$ with its samples & weight estimates //to compute $R_{v_2}$
   -($|v^i|_{\texttt{high}}, |v^i|_{\texttt{low}}$) when $v^{i-1}$ and $v^i$ differ by only one letter (then $v^i = v_2$)
   -($|v^{i-1}|_{\texttt{high}}, |v^i|_{\texttt{low}}$) otherwise
 Test: $|u_0| \geq |v_2|/2$ using $|v_2|_{\texttt{low}}$ instead of $|v_2|$
 Concatenation: $u_0 \leftarrow (v_1 \cdot R_{v_2}) \cdot u_0$
   $D_{v'} \leftarrow (D_{v|v_1}, R_{v_2})$ replacing each sample of $D_{v|v_1}$ in $v_2$ by $R_{v_2}$
   // The height of a sample determines whether it is in $v_2$
   $D_{u_0} \leftarrow$ Simplify(Concatenate($D_{v'}, D_{u_0}$))
```

▶ **Theorem 20.** *Let $\mathcal{A}$ be a* VPA *for L with $m \geq 2$ states, and let $\varepsilon, \eta > 0$. Then there is an ε-streaming algorithm for L with one-sided error η and memory space* $\mathrm{O}(m^5 2^{3m^2} (\log^6 n)(\log 1/\eta)/\varepsilon^4)$, *where n is the input length.*

5.3 Final Analysis

As Algorithm 4 may fail at various steps, the relations it considers may not correspond to any word. However, each relation R that it produces is still in Σ_Q. Furthermore, the slicing automaton $\widehat{\mathcal{A}}$ over $\widehat{\Sigma_Q}$ is $\widehat{\Sigma}$-closed. Fact 21 below bounds the $\widehat{\Sigma}$-diameter of $\widehat{\mathcal{A}}$ (which is equal to the Σ-diameter of $\mathcal{A}$) by 2^{m^2}. For simpler languages, as those coming from a DTD, this bound can be lowered to m.

▶ **Fact 21.** *Let $\mathcal{A}$ be a* VPA *with m states. Then the Σ-diameter of $\mathcal{A}$ is at most 2^{m^2}.*

We first state that the decomposition, weights and sampling we maintain are close enough to an $(1+\varepsilon')$-suffix sampling with the correct weights. Recall that $\varepsilon' = \varepsilon/(6 \log n)$.

▶ **Lemma 22** (Stability lemma). *Let v be an unfinished peak with $\mathcal{W}_1, \mathcal{W}_2$ two of the T samplers maintained by Algorithm 4. Then the joint process $(\mathcal{W}_1, \mathcal{W}_2)$ over-samples an $(1+\varepsilon')$-suffix sampling on v, and the decomposition has size at most $144(\log|v|)(\log n)/\varepsilon + \mathrm{O}(\log n)$.*

Using the tester from Theorem 16 for computing each R, we get our robustness lemma.

▶ **Lemma 23** (Robustness lemma). *Let $\mathcal{A}$ be a* VPA *recognizing L and let $u \in \Sigma^n$. Let $R_{\texttt{final}}$ be the final value of $R_{\texttt{temp}}$ in Algorithm 4.*

If $u \in L$, then $R_{\texttt{final}} \in L$; and if $R_{\texttt{final}} \in L$, then $\mathsf{bdist}_\Sigma(u, L) \leq \varepsilon n$ *with probability at least $1 - \eta$.*

References

1 N. Alon, M. Krivelevich, I. Newman, and M. Szegedy. Regular languages are testable with a constant number of queries. *SIAM Journal on Computing*, 30(6), 2000.

2 N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999.

3 R. Alur. Marrying words and trees. In *Proc. of 26th ACM Symposium on Principles of Database Systems*, pages 233–242, 2007.

4 R. Alur, M. Arenas, P. Barceló, K. Etessami, N. Immerman, and L. Libkin. First-order and temporal logics for nested words. In *Proc. of 22nd IEEE Symposium on Logic in Computer Science*, pages 151–160, 2007.

5 R. Alur, K. Etessami, and P. Madhusudan. A temporal logic of nested calls and returns. In *Proc. of 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 467–481, 2004.

6 R. Alur and P. Madhusudan. Adding nesting structure to words. *Journal of the ACM*, 56(3), 2009.

7 A. Babu, N. Limaye, and G. Varma. Streaming algorithms for some problems in log-space. In *Proc. of 7th Conference on Theory and Applications of Models of Computation*, pages 94–104, 2010.

8 M. Blum, W. Evans, P. Gemmell, S. Kannan, and M. Naor. Checking the correctness of memories. *Algorithmica*, pages 90–99, 1995.

9 M. Blum and S. Kannan. Designing programs that check their work. *Journal of the ACM*, 42(1):269–291, 1995. doi:10.1145/200836.200880.

10 M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, 47(3):549–595, 1993. doi:10.1016/0022-0000(93)90044-W.

11 B. von Braunmühl and R. Verbeek. Input-driven languages are recognized in log n space. In *Proc. of 4th Conference on Fundamentals of Computation Theory*, volume 158, pages 40–51, 1983.

12 M. Chu, S. Kannan, and A. McGregor. Checking and spot-checking the correctness of priority queues. In *Proc. of 34th International Colloquium on Automata, Languages and Programming*, pages 728–739, 2007.

13 P. Dymond. Input-driven languages are in $\log n$ depth. *Information Processing Letters*, 26(5):247–250, 1988.

14 J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. Testing and spot-checking of data streams. *Algorithmica*, 34(1):67–80, 2002. doi:10.1007/s00453-002-0959-4.

15 E. Fischer, F. Magniez, and M. de Rougemont. Approximate satisfiability and equivalence. *SIAM Journal on Computing*, 39(6):2251–2281, 2010.

16 O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. In *Proc. of 37th IEEE Symposium on Foundations of Computer Science*, pages 339–348, 1996.

17 C. Konrad and F. Magniez. Validating XML documents in the streaming model with external memory. *ACM Transactions on Database Systems*, 38(4):27, 2013. Special issue of ICDT'12.

18 L. Libkin. Logics for unranked trees: An overview. *Logical Methods in Computer Science*, 2(3), 2006.

19 F. Magniez and M. de Rougemont. Property testing of regular tree languages. *Algorithmica*, 49(2):127–146, 2007.

20 F. Magniez, C. Mathieu, and A. Nayak. Recognizing well-parenthesized expressions in the streaming model. *SIAM Journal on Computing*, 43(6):1880–1905, 2014.

21 K. Mehlorn. Pebbling mountain ranges and its application to dcfl-recognition. In *Proc. of 7th International Colloquium on Automata, Languages, and Programming*, pages 422–435, 1980.

22 S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2):117–236, 2005. doi:10.1561/0400000002.

23 A. Ndione, A. Lemay, and J. Niehren. Approximate membership for regular languages modulo the edit distance. *Theoretical Computer Science*, 487:37–49, 2013.

24 A. Ndione, A. Lemay, and J. Niehren. Sublinear DTD validity. In *Proc. of 19th International Conference on Language and Automata Theory and Applications*, pages 739–751, 2015.

25 M. Parnas, D. Ron, and R. Rubinfeld. Testing membership in parenthesis languages. *Random Structures & Algorithms*, 22(1):98–138, 2003.

26 A. Rajeev and P. Madhusudan. Visibly pushdown languages. In *Proc. of 36th ACM Symposium on Theory of Computing*, pages 202–211, 2004.

27 L. Segoufin and C. Sirangelo. Constant-memory validation of streaming XML documents against DTDs. In *Proc. of 11th International Conference on Database Theory*, pages 299–313, 2007.

28 L. Segoufin and V. Vianu. Validating streaming XML documents. In *Proc. of 11th ACM Symposium on Principles of Database Systems,*, pages 53–64, 2002.

Streaming Pattern Matching with d Wildcards*

Shay Golan[1], Tsvi Kopelowitz[2] and Ely Porat[3]

1 **Bar Ilan University, Ramat Gan, Israel**
golansh1@cs.biu.ac.il
2 **University of Michigan, Ann Arbor, Michigan, USA**
kopelot@gmail.com
3 **Bar Ilan University, Ramat Gan, Israel**
porately@cs.biu.ac.il

Abstract

In the pattern matching with d wildcards problem we are given a text T of length n and a pattern P of length m that contains d wildcard characters, each denoted by a special symbol $'?'$. A wildcard character matches any other character. The goal is to establish for each m-length substring of T whether it matches P. In the streaming model variant of the pattern matching with d wildcards problem the text T arrives one character at a time and the goal is to report, before the next character arrives, if the last m characters match P while using only $o(m)$ words of space.

In this paper we introduce two new algorithms for the d wildcard pattern matching problem in the streaming model. The first is a randomized Monte Carlo algorithm that is parameterized by a constant $0 \le \delta \le 1$. This algorithm uses $\tilde{O}(d^{1-\delta})$ amortized time per character and $\tilde{O}(d^{1+\delta})$ words of space. The second algorithm, which is used as a black box in the first algorithm, is a randomized Monte Carlo algorithm which uses $O(d + \log m)$ worst-case time per character and $O(d \log m)$ words of space.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases wildcards,don't-cares,streaming pattern matching,fingerprints

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.44

1 Introduction

We investigate one of the basic problems in pattern matching, the *pattern matching with d wildcards problem* (PMDW), in the *streaming model*. Let Σ be an alphabet and let $'?' \notin \Sigma$ be a special character called the *wildcard* which matches any character in Σ. The PMDW problem is defined as follows. Given a *text* string $T = t_0t_1 \dots t_{n-1}$ over Σ and a *pattern* string $P = p_0p_1 \dots p_{m-1}$ over alphabet $\Sigma \cup \{?\}$ such that P contains exactly d wildcard characters, report all of the occurrences of P in T. This definition of a match is one of the most well studied problems in pattern matching [21, 35, 26, 28, 18, 9].

The streaming model. The advances in technology over the last decade and the massive amount of data passing through the internet has intrigued and challenged computer scientists, as the old models of computation used before this era are now less relevant or too slow. To this end, new computational models have been suggested to allow computer scientists to tackle these technological advances. One prime example of such a model is the *streaming*

* Supported in part by NSF grants CCF-1217338, CNS-1318294, and CCF-1514383.

model [1, 25, 34, 29]. Pattern matching problems in the streaming model are allowed to preprocess P into a data structure that uses space that is sublinear in m (notice that space usage during the preprocessing phase itself is not restricted). Then, the text T is given online, one character at a time, and the goal is to report any matching substrings right after the last relevant text character has arrived, but before the next text character arrives. Another closely related model is the *online* model, which is the same as the streaming model without the constraint of using sublinear space.

Following the breakthrough result of Porat and Porat [36], there has recently been a rising interest in solving pattern matching problems in the streaming model [6, 19, 33, 7, 27, 13, 14]. However, this is the first paper to directly consider the important wildcard variant. Notice that one way for solving PMDW (not necessarily in the streaming model), is to treat $'?'$ as a regular character, and then run an algorithm that finds all occurrences of P (that does not contain any wildcards) in T with up to $k = d$ mismatches. This is known as the k-mismatch problem [32, 37, 2, 12, 11, 16, 14]. The most recent result by Clifford et al. [14] for the k-mismatch problem in the streaming model implies a solution for PMDW in the streaming model that uses $O(d^2 \operatorname{polylog} m)$ words[1] of space and $O(\sqrt{d}\log d + \operatorname{polylog} m)$ time per character. Notice that Clifford et al. [14] focused on solving a more general problem.

1.1 New results and Related Work

We improve upon the work of Clifford et al. [14], for the special case that applies to PMDW, by presenting the following algorithms (the $\tilde{O}$ notation hides logarithmic factors). Notice that Theorem 1 improves upon the results of Clifford et al. [14] whenever $\delta > 1/2$.

▶ **Theorem 1.** *For any constant $0 \le \delta \le 1$ there exists a a randomized Monte Carlo algorithm for the PMDW problem in the streaming model that succeeds with probability $1 - 1/poly(n)$, uses $\tilde{O}(d^{1+\delta})$ words of space and spends $\tilde{O}(d^{1-\delta})$ amortized time per arriving text character.*

▶ **Theorem 2.** *There exists a a randomized Monte Carlo algorithm for the PMDW problem in the streaming model that succeeds with probability $1 - 1/poly(n)$, uses $O(d \log m)$ words of space and spends $O(d + \log m)$ time per arriving text character.*

1.2 Algorithmic Overview and Related Work

Our algorithms make use of the notion of a *candidate*, which is a location in the last m indices of the current text that is currently considered as a possible occurrence of P. As more characters arrive, it becomes clear if this candidate is an actual occurrence or not. In general, an index continues to be a candidate until the algorithm encounters proof that the candidate is not a valid occurrence (or until it is reported as a match). The algorithm of Theorem 2 works by obtaining such proofs efficiently. We discuss some of the ideas used in this algorithm after discussing the overview of Theorem 1.

Overview of algorithm for Theorem 1. The algorithm of Theorem 1 uses the algorithm of Theorem 2 (with a minor adaptation) combined with a new combinatorial perspective of *periodicity* that applies to strings with wildcards. The notion of periodicity in strings (without wildcards) and its usefulness are well studied [20, 31, 36, 6, 23, 22]. However, extending the usefulness of periodicity to strings with wildcards runs into difficulties, since the notions

[1] We assume the RAM model where each word has size of $O(\log n)$ bits.

are either too inclusive or too exclusive (see [4, 3, 5, 8, 38]). Thus, we introduce a new definition of periodicity, called the *wildcard-period length* that captures, for a given pattern with wildcards, the smallest possible average distance between occurrences of the pattern in any text. See Definition 5. For a string with wildcards S, we denote the wildcard-period length of S by π_S.

Let P^* be the longest prefix of P such that $\pi_{P^*} \leq d^\delta$. The algorithm of Theorem 1 has two main components, depending on whether $P^* = P$ or not. In the case where $P^* = P$, the algorithm takes advantage of the wildcard-period length of P being small, which together with techniques from number theory and new combinatorial properties of strings with wildcards allows to spend only $\tilde{O}(1)$ time per character and uses $\tilde{O}(d^{1+\delta})$ words of space. This is summarized in Theorem 18. Of particular interest is Lemma 17 which combines number theory with combinatorial string properties in a new way. We expect these ideas to be useful in other applications.

If $P^* \neq P$, then we use the algorithm of Theorem 18 to locate occurrences of P^*, and by maximality of P^*, occurrences of any prefix of P that is longer than P^* must appear far apart (on average). These occurrences are given as input to a minor adaptation of the algorithm of Theorem 2 in the form of candidates. Utilizing the large average distance between candidates and combining with a lazy approach, we obtain an $\tilde{O}(d^{1-\delta})$ amortized time cost per character.

Overview of algorithm for Theorem 2. For the streaming pattern matching problem without wildcards, the algorithms of Porat and Porat [36] and Breslauer and Galil [6] have three major components[2]. The first component is a partitioning of the pattern into *pattern intervals* of exponentially increasing lengths. The algorithm uses *text intervals* corresponding to the pattern intervals, which are the reversed pattern intervals and appear at the end of the text. When a new text character arrives, the text intervals are shifted by one location. The second component maintains all of the candidates in a given text interval. This implementation leverages periodicity properties of strings in order to guarantee that the candidates in a given text interval form an arithmetic progression, and thus can be maintained with constant space. The third component is a fingerprint mechanism for testing if a candidate is still valid. A candidate is tested each time it leaves a text interval.

The main challenge in applying the above framework for patterns with wildcards comes from the lack of a good notion of periodicity which can guarantee that the candidates in a text interval form an arithmetic progression. Notice that the notion of wildcard-period length, for example, has to do with the average of distances between occurrences, and so it does not apply to arithmetic progressions. To tackle this challenge, we design a new method for partitioning the pattern into intervals, which combined with new fundamental combinatorial properties leads to an efficient way for maintaining the candidates in small space. In particular, we prove that with our new partitioning there are at most $O(d \log m)$ candidates that are not part of the arithmetic progression of some text interval. Remarkably, the proof bounding the number of such candidates uses a more global perspective of the pattern, as opposed to the techniques used in non-wildcard results.

More related work. We mention that while our work is in the streaming model, in the closely related online model (see [17, 15]), Clifford et al. [10] presented an algorithm, known

[2] The algorithms of Porat and Porat [36] and Breslauer and Galil [6] are not presented in this way. However, we find that this new way of presenting our algorithm (and theirs) does a better job of explaining what is going on.

as the black box algorithm, that when applied to PMDW uses $O(m)$ words of space and $O(\log^2 m)$ time per arriving text character.

Full version. Due to space consideration some of the proofs and details have been omitted, for a full version of this paper see [24].

2 Preliminaries

2.1 Periods

We assume without loss of generality that the alphabet is $\Sigma = \{1, 2, \dots, n\}$. For a string $S = s_0 s_1 \dots s_{\ell-1}$ over Σ and integer $0 \le k \le \ell$, the substring $s_0 s_1 \dots s_{k-1}$ is called a *prefix* of S and $s_{\ell-k} \dots s_{\ell-1}$ is called a *suffix* of S.

A prefix of S of length $i \ge 1$ is a *period* of S if and only if $s_j = s_{j+i}$ for every $0 \le j \le \ell - i - 1$. The shortest period of S is called *the principal period* of S, and its length is denoted by ρ_S. If $\rho_S \le \frac{|S|}{2}$ we say that S is *periodic.*

Due to space reasons, we omit proofs here, proofs appear in the full version of the paper.

▶ **Lemma 3.** *Let v be a string of length ℓ and let u be a string of length at most 2ℓ. If u contains at least three occurrences of v then:*
1. *v is a periodic string.*
2. *the distance between any two occurrences of v in u is a multiple of ρ_v.*

▶ **Lemma 4.** *Let u be a periodic string over Σ with principal period length ρ_u. If v is a substring of u of length $> 2\rho_u$ then $\rho_u = \rho_v$.*

Periods and wildcards. For strings with no wildcards there is an inverse relation between the maximum number of occurrences of u in a text of a given length and ρ_u. Here we define the *wildcard-period length* of a string over $\Sigma \cup \{?\}$ which captures a similar type of relationship for strings with wildcards. The usefulness of this definition for our needs is discussed in more detail in Section 6. Let $occ(S', S)$ be the number of occurrences of a string S in a string S'.

▶ **Definition 5.** For a string S over $\Sigma \cup \{?\}$, its wildcard-period length is

$$\pi_S = \left\lceil \frac{|S|}{\max_{S' \in \Sigma^{2|S|-1}} occ(S', S)} \right\rceil .$$

Notice that for periodic string S without wildcards $\pi_S = \rho_S$.

2.2 Fingerprints

For the following let $u, v \in \bigcup_{i=0}^{n} \Sigma^i$ be two strings of size at most n. Porat and Porat [36] and Breslauer and Galil [6] proved the existence of a *sliding fingerprint function* $\phi : \bigcup_{i=0}^{n} \Sigma^i \to [n^c]$, for some constant $c > 0$, which is a function where:
1. If $|u| = |v|$ and $u \ne v$ then $\phi(u) \ne \phi(v)$ with high probability (at least $1 - \frac{1}{n^{c-1}}$).
2. *The sliding property:* Let w=uv be the concatenation of u and v. If $|w| \le n$ then given the length and the fingerprints of any two strings from u,v and w, one can compute the fingerprint of the third string in constant time.

3 A Generic Algorithm

We introduce a generic algorithm (pseudo-code is given in Figure 1) for solving online pattern matching problems. With proper implementations of its components, this generic algorithm solves the PMDW problem. The generic algorithm makes use of the notion of a *candidate*. Initially every text index c is considered as a candidate for a pattern occurrence from the moment t_c arrives. An index continues to be a candidate until the algorithm encounters proof that the candidate is not a valid occurrence (or until it is reported as a match). A candidate is *alive* until such proof is given.

The generic algorithm is composed of three conceptual parts that affect the complexities of the algorithm; a running example of the execution of the generic algorithm appears in Figures 2 and 3:

- **Pattern and text intervals.** The first part is an ordered partitioning $\mathcal{I} = (I_0, \dots, I_k)$ of the interval $[0, m-1]$. Each interval $I \in \mathcal{I}$ is called a **pattern interval**. When a character t_α arrives then a candidate c is alive if and only if there is a pattern interval $I = [i, j] \in \mathcal{I}$ such that $t_c \cdots t_{c+i-1}$ matches $p_0 \cdots p_{i-1}$ and $\alpha - c + 1 \in [i, j]$. Notice that for any pattern interval $I = [i, j]$, any candidate c that is alive in I must be in exactly one **text interval** $c \in [\alpha - j + 1, \alpha - i + 1]$. When a new text character arrives, all the text intervals move one position ahead, and some candidates move between intervals.
- **Candidate queues.** The second conceptual part of the generic algorithm is an implementation of a **candidate-queue** data structure. This data structure supports the following operations on candidates that are in the same text interval $[\alpha - j + 1, \alpha - i + 1]$, where α is the index of the last character to arrive from T.

▶ **Definition 6.** Let α be the index of the last text character that has arrived. Then a candidate-queue on an interval $I = [i, j]$ supports the following operations.

1. *Enqueue*(c): Given candidate $c = \alpha - i + 1$ add c to the candidate-queue.
2. *Dequeue*$()$: Remove and return a candidate $c = \alpha - j$, if it exists.

Since there is a bijection between pattern intervals and text intervals we say that a candidate-queue that is associated with a given text interval is also associated with the corresponding pattern interval.

- **Assassinating candidates.** The third conceptual part is a mechanism for testing if a candidate is alive after it leaves one text interval, in order to determine if the candidate should enter the candidate queue of the next text interval, or be reported as a match if there is no more text intervals.

The implementation of each of these components controls the complexities of the algorithm. Minimizing the number of intervals reduces the number of candidates leaving an interval at any given time. Efficient implementations of the queue operations and testing if a candidate is alive control both the space usage and the amount of time spent on each candidate that leaves an interval. Notice that the implementations of these components may depend on each other, which is also the case in our solution.

If there are no wildcards in P, one can use a sliding fingerprint based approach (related to the Karp and Rabin [30] algorithm) for testing if a candidate is alive. In order to use these fingerprints, we maintain the *text fingerprint* which is the fingerprint of the text from its beginning up to the last arriving character. This maintenance uses only constant time per character and constant space.

```
PROCESS-CHARACTER($t_\alpha$)
1  $Q_0.Enqueue(\alpha)$
2  for $h = 0$ to $k$
3      $c = Q_h.Dequeue()$
4      if $c$ exists and $c$ is still alive
5          if $h = k$
6              report $c$ as a match
7          else $Q_{h+1}.Enqueue(c)$
```

Figure 1 Generic Algorithm.

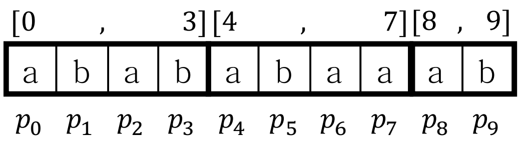


Figure 2 Example of a pattern and its arbitrary chosen pattern intervals. The pattern length is 10 and the pattern intervals are $[0, 3]$, $[4, 7]$ and $[8, 9]$.

3.1 Fingerprints with Wildcards

Using fingerprints together with wildcards seems to be a difficult task, since for any string with x wildcards there are $|\Sigma|^x$ different strings over Σ that match the string. Each one of these different strings may have a different fingerprint and therefore there are $O(|\Sigma|^x)$ fingerprints to store, which is not feasible. In order to still use fingerprints for solving PMDW we use a special partitioning of $[0, m-1]$.

Partitioning algorithm. We use a representation of P as $P = P_0?P_1?\ldots?P_d$ where each subpattern P_i contains only characters from Σ (and may also be the empty string). Let $W = (w_1, w_2, \ldots, w_d)$ be the indices such that $p_{w_i} = '?'$ and for all $1 \le i < d$ we have $w_i < w_{i+1}$. The interval $[0, m-1]$ is partitioned into pattern intervals as follows:

$$\mathcal{J} = ([0, w_1 - 1], [w_1, w_1], [w_1 + 1, w_2 - 1], \ldots, [w_d, w_d], [w_d + 1, m - 1]).$$

Since some of the pattern intervals in this partitioning could be empty, we discard such intervals. The pattern intervals of the form $[w_i, w_i]$ are called *wildcard intervals* and the other pattern intervals are called *regular intervals*. Notice that for a text index c, the substring $t_c \ldots t_{c+m-1}$ matches P if and only if for each regular interval $[i, j]$, $t_{c+i} \ldots t_{c+j} = p_i \ldots p_j$. During the initialization of the algorithm we precompute and store the fingerprints for all of the subpatterns corresponding to regular intervals.

Testing liveness. Given the partition $\mathcal{J}$, the algorithm for testing if a candidate c is alive is as follows. Each time a candidate c is added to a candidate-queue for interval $[i, j] \in \mathcal{J}$ via the $Enqueue(c)$ operation, the algorithm stores the current text fingerprint $\phi(t_0 \ldots t_{c+i-1})$ with the candidate c. When the character t_{c+j} arrives the text fingerprint is $\phi(t_0 \ldots t_{c+j})$. At this time, the algorithm uses the $Dequeue()$ operation to extract c together with $\phi(t_0 t_1 \ldots t_{c+i-1})$

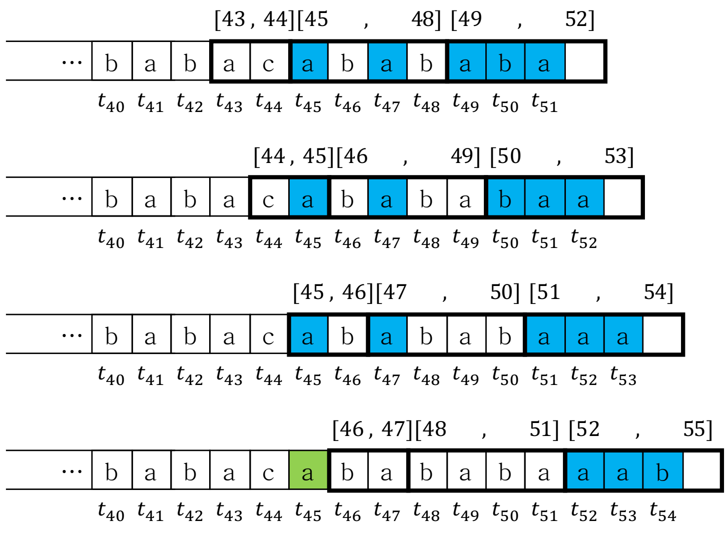

[43 , 44][45 , 48] [49 , 52]
... b a b a c a b a b a b a
t_{40} t_{41} t_{42} t_{43} t_{44} t_{45} t_{46} t_{47} t_{48} t_{49} t_{50} t_{51}
[44 , 45][46 , 49] [50 , 53]
... b a b a c a b a b a b a a
t_{40} t_{41} t_{42} t_{43} t_{44} t_{45} t_{46} t_{47} t_{48} t_{49} t_{50} t_{51} t_{52}
[45 , 46][47 , 50] [51 , 54]
... b a b a c a b a b a b a a a
t_{40} t_{41} t_{42} t_{43} t_{44} t_{45} t_{46} t_{47} t_{48} t_{49} t_{50} t_{51} t_{52} t_{53}
[46 , 47][48 , 51] [52 , 55]
... b a b a c a b a b a b a a a b
t_{40} t_{41} t_{42} t_{43} t_{44} t_{45} t_{46} t_{47} t_{48} t_{49} t_{50} t_{51} t_{52} t_{53} t_{54}

Figure 3 Example of the generic algorithm execution with the pattern of Figure 2. In each row a new text character arrives. The bold borders are on the text intervals, each blue cell is a position of a candidate and the green cell corresponds a match.
When t_{52} arrives the candidate $c_1 = 45$ is tested, since it exists a text interval and found to be alive because *abababaa* is a prefix of the pattern. At this time we can see that the candidate $c_2 = 47$ cannot be a valid occurrence of the pattern, however the algorithm will not remove it until it reaches the end of the text interval.
When t_{54} arrives, the candidates $c_1 = 45$ and $c_2 = 47$ are tested since they reach the end of their text intervals. c_2 is removed because the text *ababaaab* is not a prefix of the pattern. The candidate c_1 is still alive, and since it reaches the end of the last text interval, it reported as a match.

from the candidate-queue of interval $[i, j]$. If $[i, j]$ is a regular interval, then the algorithm tests if c is alive. This is done by applying the sliding property of the fingerprint function to compute $\phi(t_{c+i} \dots t_{c+j})$ from the current text fingerprint $\phi(t_0 t_1 \dots t_{c+j})$ and the fingerprint $\phi(t_0 t_1 \dots t_{c+i-1})$, and then testing if $\phi(t_{c+i} \dots t_{c+j})$ is the same as $\phi(p_i \dots p_j)$. If $[i, j]$ is a wildcard interval then c stays alive without any test.

A naïve implementation of the candidate queues provides an algorithm that costs $O(d)$ time per character, but uses $\Theta(m)$ words of space. To overcome this space usage we employ a more complicated partitioning, which together with a modification of the requirements from the candidate-queues allows us to design a data structure that uses much less space. However, this space efficiency comes at the expense of a slight increase in time per character.

4 The Partitioning

The key idea of the new partitioning is to use the partitioning of Section 3.1 as a preliminary partitioning, and then perform a secondary partitioning of the regular pattern intervals, thereby creating even more regular intervals. As mentioned, the intervals are partitioned in a special way which allows us to implement candidate-queues in a compact manner (see Section 5).

The following definition is useful in the next lemma.

▶ **Definition 7.** For an ordered set of intervals $\mathcal{I} = (I_0, I_1, \dots I_k)$ and for any integer $0 \le i \le k$, let $\mu_{\mathcal{I}}(i) = \max_{j=0}^{i} |I_j|$ be the length of the longest interval in the sequence $I_0, \dots I_i$. When $\mathcal{I}$ is clear from context we simply write $\mu(i) = \mu_{\mathcal{I}}(i)$

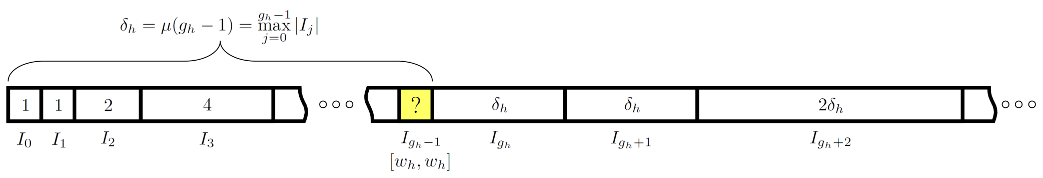


Figure 4 The general case: on each $J_h \in \mathcal{J}$ we first create two intervals of length δ_h and then we iteratively create pattern intervals where the length of each pattern interval is double the length of the previous pattern interval.

The following lemma shows a partitioning which is used to improve the preliminary partitioning algorithm. The properties of the partitioning that are described in the statement of the lemma are essential for our new algorithm. In the proof we introduce a specific partitioning which has all of these properties.

▶ **Lemma 8.** *Given a pattern P of length m with d wildcards There exists a partitioning of the interval $[0, m-1]$ into subintervals $\mathcal{I} = (I_0, I_1 \dots, I_k)$ which has the following properties:*

1. *If $I = [i, j]$ is a pattern interval then $p_i \dots p_j$ either corresponds to exactly one wildcard from P (and so $j = i$) or it is a substring that does not contain any wildcards.*
2. *$k = O(d + \log m)$.*
3. *For each regular pattern interval $I = [i, j]$ with $|I| > 1$, the length i prefix of P contains a consecutive sequence of $|I|$ non-wildcard characters.*
4. *$|\{\mu_{\mathcal{I}}(0), \mu_{\mathcal{I}}(1) \dots \mu_{\mathcal{I}}(k)\}| = O(\log m)$.*

Proof. We introduce a secondary partitioning of the preliminary partitioning described in Section 3.1, and prove that it has all the required properties; see Figures 4, 5 and 6. Let J_h be the preliminary pattern interval corresponding to P_h. The secondary partitioning is executed on the pattern intervals $\mathcal{J} = (J_0, J_1, \dots, J_d)$, where the partitioning of J_h is dependent on the partitioning of $J_0, \dots, J_{h-1}$. Thus, the secondary partitioning of J_h takes place only after the second partitioning of J_{h-1}.

When partitioning pattern interval $J_h = [i, j]$, let g_h be the number of pattern intervals in the secondary partitioning in $[0, i-1]$, and let $\delta_h = \mu_{\mathcal{I}}(g_h - 1)$ be the length of the longest pattern interval in the secondary partitioning of $[0, i-1]$. For the first pattern interval let $\delta_0 = 1$. If $j \le i + \delta_h - 1$ then the only pattern interval is all of J_h. If $j \le i + 2\delta_h - 1$ then we create the pattern intervals $[i, i + \delta_h - 1]$ and $[i + \delta_h, j]$. Otherwise, we first create the pattern intervals $[i, i + \delta_h - 1]$ and $[i + \delta_h, i + 2\delta_h - 1]$, and for as long as there is enough room in the remaining preliminary pattern interval J_h (between the position right after the end of the last secondary pattern interval that was just created and j) we iteratively create pattern intervals where the length of each pattern interval is double the length of the previous pattern interval. Once there is no more room left in J_h , let ℓ be the length of the last pattern interval we created. If the remaining part of the preliminary pattern interval is of length at most ℓ, then we create one pattern interval for all the remaining preliminary pattern interval. Otherwise we create two pattern intervals, the first pattern interval of length ℓ and the second pattern interval using the remaining part of J_h.

The secondary partitioning implies all of the desired properties, the proof appears in the full version of this paper. ◀

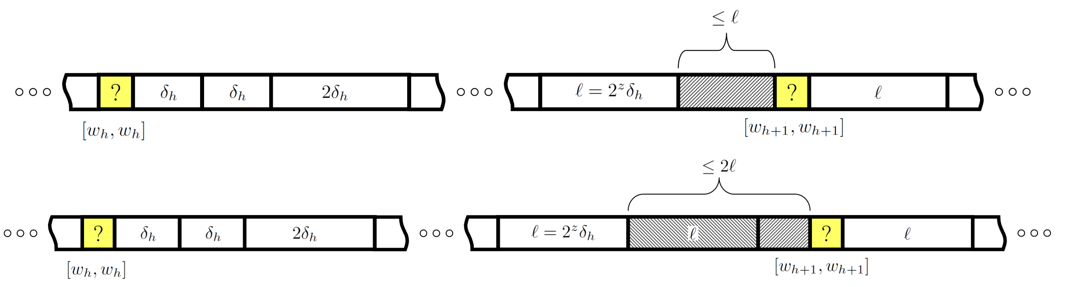


Figure 5 Once there is no more room left in J_h, if the remaining interval is of length at most ℓ (the top case), then we create one pattern interval for all the remaining interval. Otherwise (the bottom case) we create two pattern intervals, the first pattern interval of length ℓ and the second pattern interval using the remaining part of J_h.

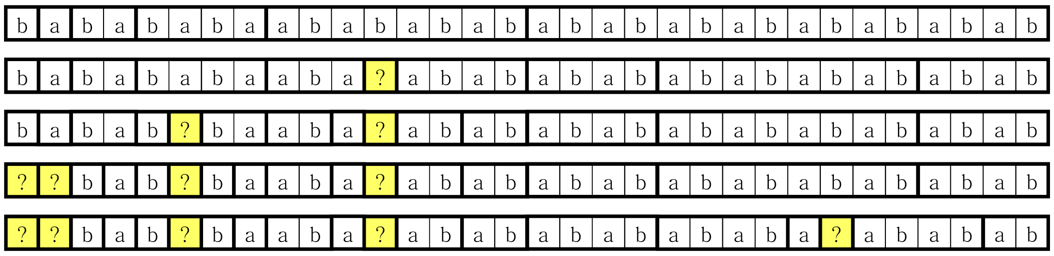

b a b a b a b a a b a b a b a b a b a b a b a b a b a b a b a b
b a b a b a b a a b a ? a b a b a b a b a b a b a b a b a b a b
b a b a b ? b a a b a ? a b a b a b a b a b a b a b a b a b a b
? ? b a b ? b a a b a ? a b a b a b a b a b a b a b a b a b a b
? ? b a b ? b a a b a ? a b a b a b a b a b a b a ? a b a b a b

Figure 6 Example of patterns and their intervals in the secondary partitioning. Each bold rectangle corresponds to an interval in the partition.

5 The Candidate-fingerprint-queue

The algorithm of Theorem 2 is obtained via an implementation of the candidate-queues that uses $O(d \log m)$ space, at the expense of having $O(d + \log m)$ intervals in the partitioning. Such space usage implies that we do not store all candidates explicitly. This is obtained by utilizing properties of periodicity in strings. Since candidates are not stored explicitly, we cannot store any explicit information per candidate, and in particular we cannot explicitly store fingerprints to quickly test if a candidate is still alive. On the other hand, we are still interested in using fingerprints in order to perform these tests.

To tackle this, we strengthen our requirements from the candidate-queue data structure to return not just the candidate but also the fingerprint information that is needed to perform the test of whether the candidate is still alive. Thus, we extend the definition of a candidate-queue to a *candidate-fingerprint-queue* as follows.

► **Definition 9.** Let α be the index of the last text character that has arrived. Then a candidate-fingerprint-queue on an interval $I = [i, j]$ supports the following operations.

1. *Enqueue*$(c, \phi(t_0 \dots t_{c-1}), \phi(t_0 \dots t_\alpha))$: given $c = \alpha - i + 1$ add c to the candidate-queue, together with $\phi(t_0 \dots t_{c-1})$ and $\phi(t_0 \dots t_\alpha)$.
2. *Dequeue*$()$: Remove and return a candidate $c = \alpha - j$, if it exists, together with $\phi(t_0 \dots t_{c-1})$ and $\phi(t_0 \dots t_{c+i-1})$.

In order to reduce clutter of presentation, in the rest of this section we refer to the candidate-fingerprint-queue simply as the *queue*.

5.1 Implementation

The implementation of the queue assumes that we use a partitioning that has the properties stated in Lemma 8. Let $I = [i, j]$ be a pattern interval in the partitioning and let c be a candidate which is maintained in the queue Q_I associated with I. For candidate c, the *entrance prefix* is the substring $t_c \dots t_{c+i-1}$, the *entrance interval* is $[c, c+i-1]$, and the *entrance fingerprint* is $\phi(t_c \dots t_{c+i-1})$. Since c was alive at the time it was inserted into Q_I, the entrance prefix of c matches $p_0 \dots p_{i-1}$ (which may contain wildcards). Recall that a candidate c is inserted into Q_I together with $\phi(t_0 \dots t_{c-1})$, which we call the *candidate fingerprint* of c.

Satellite information. The implementation associates each candidate c with *satellite information* (SI), which includes the candidate fingerprint and the entrance fingerprint of the candidate. The SI of a candidate combined with the the sliding property of fingerprints are crucial for the implementation of the queue. When c is added to Q_I, for some $I = [i, j]$, we compute the entrance fingerprint of c from the candidate fingerprint and from $\phi(t_0 \dots t_{c+i-1})$ which is the text fingerprint at this time. When c is removed from Q_I, we compute $\phi(t_0 \dots t_{c+i-1})$ in constant time from the SI of c.

Entrance prefixes and arithmetic progressions. A key component of the queue data structure is Lemma 10. This lemma defines for each interval $I = [i, j] \in \mathcal{I}$ at most one unique entrance prefix u_I that is the only string that can be the entrance prefix of more than two candidates in Q_I at the same time. That is, the existence of an entrance prefix u_I and determination u_I if it exists **depends only** on the prefix pattern $p_0 \dots p_{i-1}$, *regardless* of the characters in the text. If I has such a string u_I we say that I is an *arithmetic interval*, since, as we prove in Lemma 11 the candidates in Q_I that have entrance prefix u_I form an arithmetic progression.

▶ **Lemma 10.** *For a pattern interval $I = [i, j]$ with queue Q_I, there exists at most one string u_I such that if there are more than two candidates in Q_I with the same entrance prefix then this entrance prefix must be u_I.*

Proof Sketch. By Lemma 8 there is a string of length $|I|$ containing only non-wildcard characters that is a substring of $p_0 \dots p_{i-1}$. Let v be this string. Notice that v must appear in the entrance prefix of every candidate in Q_I. If there are three candidates in Q_I then they must all appear in the text within a range of size $j - i + 1$, which is close enough to guarantee that v must be periodic.

Now, consider three candidates $c_4 < c_5 < c_6$ in Q_I that have the same entrance prefix u. Since c_4, c_5, and c_6 are all occurrences of u then $c_6 - c_5$ and $c_5 - c_4$ are period lengths of u. Thus, $\rho_u \le \min\{c_5 - c_4, c_6 - c_5\} \le \frac{c_6 - c_4}{2} \le \frac{j-i}{2} < \frac{|I|}{2} = \frac{|v|}{2}$. Therefore, by Lemma 4 $\rho_u = \rho_v$. Combined with the fact that u contains v at a particular location (since u is an entrance prefix), and v is longer than ρ_u (since $\rho_u = \rho_v$), the string u must be uniquely defined. ◀

▶ **Lemma 11.** *Let $I = [i, j]$ be an arithmetic interval. If there are $h \ge 3$ candidates $c_1 < c_2 < \dots < c_h$ in Q_I that have u_I as their entrance prefix, then the sequence $c_1, c_2, \dots, c_h$ forms an arithmetic progression whose difference is ρ_{u_I}.*

Implementation details. For an interval I, we use a linked list $\mathcal{L}_{Q_I}$ to store all of the candidates in Q_I together with their SI, except for when I is an arithmetic interval in which case candidates whose entrance fingerprint is $\phi(u_I)$ are not stored in $\mathcal{L}_{Q_I}$. Adding and removing a candidate that belongs in $\mathcal{L}_{Q_I}$ together with its SI is straightforward. If I is an arithmetic interval, the candidates in Q_I whose entrance fingerprint is $\phi(u_I)$ are stored using a separate data structure that leverages Lemma 11. Thus, during a *Dequeue*() operation, the queue verifies if the candidate to be returned is in $\mathcal{L}_{Q_I}$ or in the separate data structure for the candidates with entrance fingerprint $\phi(u_I)$.

▶ **Lemma 12.** *There exists an implementation of candidate-fingerprint-queues so that for any arithmetic interval I, the queue Q_I maintains all the candidates with entrance fingerprint $\phi(u_I)$ and their SI using $O(1)$ words of space.*

Space usage. The space usage of all of the queues has two components. The first component is the lists $\mathcal{L}_{Q_I}$ for all the intervals I. The second component is the data structure for storing the candidates that create the arithmetic progression of an arithmetic interval, for each such arithmetic interval. By Lemma 10 there is at most one such progression per arithmetic interval, and so all of these arithmetic progressions use $O(d + \log m)$ space. In the following lemma we prove that the total space usage of all of the lists is $O(d \log m)$.

▶ **Lemma 13.** $\sum_{I\in\mathcal{I}} |\mathcal{L}_{Q_I}| = O(d\log m)$.

Proof. By Lemma 8, we know that $|\{\mu(0),\dots,\mu(k)\}| = O(\log m)$. For each $\ell \in \{\mu(0),\dots,\mu(k)\}$ let $\mathcal{I}_\ell$ be the sequence of all pattern intervals in $\mathcal{I}$ that are between the leftmost interval of length ℓ, inclusive, and the first of either the leftmost interval of length larger than ℓ, exclusive, or the last interval in $\mathcal{I}$, inclusive. Notice that $|I| \le \ell$ for each $I \in \mathcal{I}_\ell$. Let $\mathcal{D}_\ell$ be the set of queues for pattern intervals in $\mathcal{I}_\ell$. We show that $\sum_{I\in\mathcal{I}_\ell} |\mathcal{L}_{Q_I}| = O(|\mathcal{I}_\ell| + d)$. This implies that:

$$\sum_{I\in\mathcal{I}} |\mathcal{L}_{Q_I}| = \sum_{\ell\in\{\mu(0),\dots,\mu(k)\}} \sum_{I\in\mathcal{I}_\ell} |\mathcal{L}_{Q_I}| = \sum_{\ell\in\{\mu(0),\dots,\mu(k)\}} O(|\mathcal{I}_\ell| + d) = O(d\log m).$$

We focus on queues for which $|\mathcal{L}_{Q_I}| \ge 3$, since otherwise the bound is immediate. Notice that this includes all queues for wildcard intervals. Set $\ell \in \{\mu(0),\dots,\mu(k)\}$ and let α be the index of the last text character that has arrived.

We now establish that there exists a periodic string v of length ℓ that contains no wildcards, such that for any candidate c in any of the queues of $\mathcal{D}_\ell$ the entrance prefix of c contains v. Let $[i,j]$ be the leftmost interval in $\mathcal{I}_\ell$. Notice that $j = i + \ell - 1$ by the definition of $\mathcal{I}_\ell$. By Lemma 8, there is a string of length ℓ containing only non-wildcard characters that is a substring $p_0 \dots p_{i-1}$. Let r be the starting location of this string, and let $v = p_r \dots p_{r+\ell-1}$ be this string.

For each queue $Q_{I'} \in \mathcal{D}_\ell$ where $I' = [i',j']$ and for each candidate $c' \in Q_{I'}$, the entrance prefix of c' matches the prefix of P of length i'. Since $i' \ge i$ this means that $t_{c'+r} \dots t_{c'+r+\ell-1} = v$. The text substring $t_{\alpha-j'+r+1} \dots t_{\alpha-i'+r+\ell}$ contains at least three occurrences of v and its length is $|I'| + \ell \le 2\ell$. Therefore, by Lemma 3, v is a periodic string and the distance between any two candidates in the same queue is a multiple of ρ_v. Notice that for any $Q_{I'} \in \mathcal{D}_\ell$ that contains at least three candidates $c_1 < c_2 < c_3$, we bound $\rho_v \le \min\{c_3 - c_2, c_2 - c_1\} \le \frac{c_3-c_1}{2} \le \frac{j'-i'}{2}$.

Let $\hat{c}$ be the rightmost (largest index) candidate maintained in the queues of $\mathcal{D}_\ell$. In particular $t_{\hat{c}+r} \dots t_{\hat{c}+r+\ell-1} = v$. We extend this occurrence of v to the left and to the right in T for as long as the length of the period does not increase. Let the resulting substring be $t_{x+1} \dots t_{y-1}$. Unless, $x = -1$, the index x is called the *left violation* of v. Similarly, unless $y = \alpha+1$, the index y is called the *right violation* of v. Notice that $x < \hat{c}+r \le \hat{c}+r+\ell-1 < y$.

For the following, let the entrance interval of a candidate c be $[c, e_c]$.

▶ **Claim 14.** *If $Q_{[i',j']} \in \mathcal{D}_\ell$ contains at least three candidates, then for each candidate c in $\mathcal{L}_{Q_{[i',j']}}$ either $x \in [c, e_c]$ or $y \in [c, e_c]$.*

Proof Sketch. There exists a text index $\beta = \hat{c} + r$, such that β appears in the entrance interval of any candidate in the queues of $\mathcal{D}_\ell$, and $x < \beta < y$. Therefore, for each candidate $c \in \mathcal{L}_{Q_{[i',j']}}$, if $x \notin [c, e_c]$ and $y \notin [c, e_c]$ it must be the case that $[c, e_c] \subseteq [x+1, y-1]$. Recall that the principal period length of $t_{x+1} \dots t_{y-1}$ is ρ_v. Since $u = t_c \dots t_{e_c}$ is a substring of $t_{x+1} \dots t_{y-1}$, it must be that $\rho_u \le \rho_v$. Hence, one can prove that $[i', j']$ is an arithmetic interval with $u_{[i',j']} = u$, contradicting the fact that c is stored in $\mathcal{L}_{Q_{[i',j']}}$. ◀

Let $\mathcal{L}^x_{Q_I}$ ($\mathcal{L}^y_{Q_I}$) be the sets of candidates that are stored in $\mathcal{L}_{Q_I}$ such that their entrance interval contains x (y). $\mathcal{L}^x_{Q_I}$ and $\mathcal{L}^y_{Q_I}$ are not necessarily disjoint. Notice that by Claim 14, $\mathcal{L}^x_{Q_I} \cup \mathcal{L}^y_{Q_I}$ contains all the candidates stored in $\mathcal{L}_{Q_I}$.

▶ **Claim 15.** *$\sum_{I \in \mathcal{I}_\ell} |\mathcal{L}^x_{Q_I}| = O(|\mathcal{I}_\ell| + d)$ and $\sum_{I \in \mathcal{I}_\ell} |\mathcal{L}^y_{Q_I}| = O(|\mathcal{I}_\ell| + d)$.*

Proof. Let $I \in \mathcal{I}_\ell$ and let $\approx$ denote the match relation between symbols in $\Sigma \cup \{?\}$.

Notice that the contribution to $\sum_{I \in \mathcal{I}_\ell} |\mathcal{L}^x_{Q_I}|$ from all sets $\mathcal{L}^x_{Q_I}$ that have less than two candidates is at most $O(|\mathcal{I}_\ell|)$. Thus, for the following we assume that $\mathcal{L}^x_{Q_I}$ contains at least two candidates. Let $c_{I,x} = \max \mathcal{L}^x_{Q_I}$ be the most recent candidate in $\mathcal{L}^x_{Q_I}$. Let $c < c_{I,x}$ be a candidate in $\mathcal{L}^x_{Q_I}$. Since $c \in \mathcal{L}^x_{Q_I}$ we have that $p_{x-c} \approx t_{c+x-c} = t_x$ (recall that both x and c are indices in the text). Similarly, since $c_{I,x} \in \mathcal{L}^x_{Q_I}$ we have that $p_{x-c} \approx t_{c_{I,x}+x-c} = t_{x+(c_{I,x}-c)}$. Recall that the distance between any two candidates in Q_I is a multiple of ρ_v. In particular the distance $(c_{I,x} - c)$ is a multiple of ρ_v and $(c_{I,x} - c) \le |I| \le |v|$. Thus, $t_x \ne t_{x+(c_{I,x}-c)}$ since x violates the period of length ρ_v. Recall that $t_x \approx p_{x-c} \approx t_{x+(c_{I,x}-c)}$, and so p_{x-c} must be a wildcard. Therefore, each $c \in \mathcal{L}^x_{Q_I}$, except for possibly $c_{I,x}$, is in a position c such that p_{x-c} is a wildcard. Since x is the same for all of the candidates in all of the $\mathcal{L}^x_{Q_{I'}}$ for all $I' \in \mathcal{I}_\ell$, then the contribution to $\sum_{I \in \mathcal{I}_\ell} |\mathcal{L}^x_{Q_I}|$ of the candidates that are not the most recent in their set $\mathcal{L}^x_{Q_I}$ is at most $O(d)$. The contribution of the most recent candidates is at most $O(|\mathcal{I}_\ell|)$. Thus, $\sum_{I' \in \mathcal{I}_\ell} |\mathcal{L}^x_{Q_{I'}}| = O(|\mathcal{I}_\ell| + d)$. The proof that $\sum_{I' \in \mathcal{I}_\ell} |\mathcal{L}^y_{Q_{I'}}| = O(|\mathcal{I}_\ell| + d)$ is symmetric. ◀

Finally, $\sum_{I \in \mathcal{I}_\ell} |\mathcal{L}_{Q_I}| \le \sum_{I \in \mathcal{I}_\ell} |\mathcal{L}^x_{Q_I}| + \sum_{I \in \mathcal{I}_\ell} |\mathcal{L}^y_{Q_I}| = O(|\mathcal{I}_\ell| + d)$. Thus, we have completed the proof of Lemma 13. ◀

6 The Tradeoff Algorithm

The algorithm of Theorem 2 for PMDW uses $\tilde{O}(d)$ time per character and $\tilde{O}(d)$ words of space. In this section we introduce a randomized algorithm which expands this result for a parameter $0 \le \delta \le 1$ to an algorithm that uses $\tilde{O}(d^{1-\delta})$ time per character and $\tilde{O}(d^{1+\delta})$ words of space.

An overview of a slightly modified version (for the sake of intuition) of the tradeoff algorithm is described as follows. Let P^* be the longest prefix of P such that $\pi_{P^*} \le d^\delta$. The tradeoff algorithm first finds all the occurrences of P^* using a specialized algorithm for patterns with wildcard-period length at most d^δ. If $P^* = P$ then this completes the tradeoff algorithm. Otherwise, let $I^* = [i^*, j^*]$ be the interval in the secondary partitioning of Theorem 2 such that $i \le |P^*| \le j$. Each occurrence of P^* in the text is inserted as a candidate in the algorithm of Theorem 2 directly into Q_{I^*} Thus, the entrance prefixes of candidates in the queues match prefixes of P that are longer than P^* and, by maximality of P^*, these prefixes of P have large wildcard-period length. This means that the average distance between each two consecutive candidates is at least d^δ, and so combined with a lazy approach we are able to obtain an $\tilde{O}(d^{1-\delta})$ amortized time cost per character.

In the rest of this section we describe an overview of the specialized algorithm for dealing with patterns whose wildcard-period length is at most d^δ. The rest of the details for the tradeoff algorithm appear in the full version of this paper.

6.1 Patterns with Small Wildcard-period Length

Let P be a pattern of length m with $\pi_P < d^\delta$. Let q be an integer, which for simplicity is assumed to divide m. Consider the conceptual matrix $M^q = \{m^q_{x,y}\}$ of size $\left\lceil \frac{m}{q} \right\rceil \times q$ where $m^q_{x,y} = p_{(x-1)\cdot q+y-1}$. For any integer $0 \le r < q$ the rth column corresponds to an *offset pattern* $P_{q,r} = p_r p_{r+q} p_{r+2q} \dots p_{m-q+r}$. Notice that some offset patterns might be equal. Let $\Gamma_q = \{P_{q,r} | 0 \le r < q, '?' \notin P_{q,r}\}$ be the set of all the offset patterns that do not contain any wildcards. Each unique offset pattern is associated with a unique id. The set of unique ids is denoted by ID_q. We say that index i in P is *covered* by q if the column containing p_i is given a unique id. The columns of M^q define a *column pattern* P_q of length q, where the i'th character is the unique id of the i'th column, or '?' if no such id exists (the column has wildcards).

We partition T into q *offset texts*, where for every $0 \le r < q$ we define $T_{q,r} = t_r t_{r+q} t_{r+2q} \dots$. Using the dictionary matching streaming (DMS) algorithm of Clifford et al. [13] we find occurrences of offset patterns from Γ_q in each of the offset texts. Notice that we do *not* only find occurrences of $P_{q,r}$ in $T_{q,r}$ (since we cannot guarantee that the offset of T synchronizes with an occurrence of P). When the character t_α arrives, the algorithm passes t_α to the DMS algorithm for $T_{q,\alpha \bmod q}$. We also create a streaming *column text* T_q whose characters correspond to the ids of offset patterns as follows. If one of the offset patterns is found in $T_{q,\alpha \bmod q}$, then its unique id is the αth character in T_q. Otherwise, we use a dummy character for the αth character in T_q.

Notice that on occurrence of P in T necessarily creates an occurrence of P_q in T_q. Such occurrences are found via the black box algorithm of Clifford et al. [10]. However, an occurrence of P_q in T_q does not necessarily mean there was an occurrence of P in T, since some characters in P are not covered by q. In order to avoid such false positives we run the process in parallel with several choices of q, while guaranteeing that each non wildcard character in P is covered by at least one of those choices. Thus, if there is an occurrence of P_q at location i in T_q for all the choices of q, then it must be that P appears in T at location i. The choices of q are given by the following lemma.

▶ **Lemma 16.** *There exists a set Q of $O(\log d)$ prime numbers such that any index of a non-wildcard character in P is covered by at least one prime number $q \in Q$, and $\forall q \in Q : q = \tilde{O}(d)$.*

From a space usage perspective, we need the size of $|\Gamma_q|$ to be small, since this directly affects the space usage of the DMS algorithm which uses $\tilde{O}(k)$ space, where k is the number of patterns in the dictionary. In our case $k = |\Gamma_q|$. In order to bound the size of Γ_q we use the following lemma.

▶ **Lemma 17.** *If $q = \tilde{O}(d)$ and $\pi_P \le d^\delta$ then $|\Gamma_q| = O(d^\delta)$.*

Proof. Since $\pi_P \le d^\delta$, there exists a string $S = s_0 \dots s_{2m-2}$ that contains $\Omega(\frac{m}{d^\delta})$ occurrences of P. Using this string we show that $|\Gamma_q| = O(d^\delta)$.

For each id in ID_q we pick an index of a representative column in M_q that has this id, and denote this set by R_q. Let r_1 be the minimum index in R_q. For every index $0 \le i < m$

let $S_i = s_i \dots s_{i+m-1}$. For every $0 \le r < q$ let $S_{i,q,r} = s_{i+r}s_{i+r+q} \dots s_{i+m-q+r}$, and so for any integer $0 \le \Delta < q - r$ we have $S_{i,q,r+\Delta} = S_{i+\Delta,q,r}$. Notice that if S_i matches P then $P_{q,r} = S_{i,q,r}$ for each $r \in R_q$.

Let i be an index of an occurrence of P in S. For any distinct $r, r' \in R_q$, it must be that $S_{i,q,r} = P_{q,r} \neq P_{q,r'} = S_{i,q,r'}$. Thus, for any $r \in R_q$ such that $r > r_1$, we have $P_{q,r_1} = S_{i,q,r_1} \neq S_{i,q,r} = S_{i+r-r_1,q,r_1}$. This implies that $i + r - r_1$ is not occurrence of P. Therefore, every occurrence of P in S is associated with $|R_q| - 1$ indices that cannot be an occurrence of P. We further argue that each index i is associated with an occurrence of P (as just described) at most once. This is because if $S_{i,q,r_1} = P_{q,r}$ for some $r \in R_q$ then i can be associated only with an occurrence of P in index $i - (r - r_1)$. So the maximum number of instances of P in S is at most $\frac{|S|}{|R_q|} = \frac{2m-1}{|R_q|}$. However, S contains at least $\frac{m}{d^\delta}$ instances of P, so $\frac{m}{d^\delta} \le \frac{2m-1}{|R_q|}$ which implies that $|\Gamma_q| = |R_q| \le 2d^\delta = O(d^\delta)$. ◀

Complexities. For a single $q \in Q$, the algorithm creates $q = \tilde{O}(d)$ offset patterns and texts. For each such offset text the algorithm applies an instance of the DMS algorithm with a dictionary of $O(d^\delta)$ strings (by Lemma 17). Since each instance of the DMS algorithm uses $\tilde{O}(d^\delta)$ words of space [13], the total space usage for all instances of the DMS algorithm is $\tilde{O}(d^{1+\delta})$ words. Moreover, the time per character in each DMS algorithm is $\tilde{O}(1)$ time, and each time a character appears we inject it into only one of the DMS algorithms (for this specific q). In addition, the algorithm uses an instance of the black box algorithm for T_q, with a pattern of length q. This uses another $O(q) = \tilde{O}(d)$ space and another $\tilde{O}(1)$ time per character [10]. Thus the total space usage due to one element in Q is $\tilde{O}(d^{1+\delta})$ words. Since $|Q| = O(\log d)$ the total space usage for all elements in Q is $\tilde{O}(d^{1+\delta})$ words, and the total time per arriving character is $\tilde{O}(1)$. Thus we have proven the following.

▶ **Theorem 18.** *For any $0 \le \delta \le 1$ the online d wildcards pattern matching problem can be solved for patterns P with $\pi_P < d^\delta$ with a randomized Monte Carlo algorithm, in $\tilde{O}(1)$ time per arriving text character and using $\tilde{O}(d^{1+\delta})$ words of space.*

References

1 Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999. doi:10.1006/jcss.1997.1545.

2 Amihood Amir, Moshe Lewenstein, and Ely Porat. Faster algorithms for string matching with k mismatches. *J. Algorithms*, 50(2):257–275, 2004. doi:10.1016/S0196-6774(03)00097-X.

3 Jean Berstel and Luc Boasson. Partial words and a theorem of fine and wilf. *Theor. Comput. Sci.*, 218(1):135–141, 1999. doi:10.1016/S0304-3975(98)00255-2.

4 Francine Blanchet-Sadri. *Algorithmic Combinatorics on Partial Words.* Discrete mathematics and its applications. CRC Press, 2008. URL: http://www.crcpress.com/product/isbn/9781420060928.

5 Francine Blanchet-Sadri and Robert A. Hegstrom. Partial words and a theorem of fine and wilf revisited. *Theor. Comput. Sci.*, 270(1-2):401–419, 2002. doi:10.1016/S0304-3975(00)00407-2.

6 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.

7 Dany Breslauer, Roberto Grossi, and Filippo Mignosi. Simple real-time constant-space string matching. *Theor. Comput. Sci.*, 483:2–9, 2013. doi:10.1016/j.tcs.2012.11.040.

8 Sabin Cautis, Filippo Mignosi, Jeffrey Shallit, Ming-wei Wang, and Soroosh Yazdani. Periodicity, morphisms, and matrices. *Theor. Comput. Sci.*, 295:107–121, 2003. doi:10.1016/S0304-3975(02)00398-5.

9 Peter Clifford and Raphaël Clifford. Simple deterministic wildcard matching. *Inf. Process. Lett.*, 101(2):53–54, 2007. doi:10.1016/j.ipl.2006.08.002.

10 Raphaël Clifford, Klim Efremenko, Benny Porat, and Ely Porat. A black box for online approximate pattern matching. *Inf. Comput.*, 209(4):731–736, 2011. doi:10.1016/j.ic.2010.12.007.

11 Raphaël Clifford, Klim Efremenko, Ely Porat, and Amir Rothschild. From coding theory to efficient pattern matching. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 778–784, 2009. URL: http://dl.acm.org/citation.cfm?id=1496770.1496855.

12 Raphaël Clifford, Klim Efremenko, Ely Porat, and Amir Rothschild. Pattern matching with don't cares and few errors. *J. Comput. Syst. Sci.*, 76(2):115–124, 2010. doi:10.1016/j.jcss.2009.06.002.

13 Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana A. Starikovskaya. Dictionary matching in a stream. In Nikhil Bansal and Irene Finocchi, editors, *Proc. 23rd Annual European Symposium on Algorithms (ESA'15)*, volume 9294 of *LNCS*, pages 361–372. Springer, 2015. doi:10.1007/978-3-662-48350-3_31.

14 Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana A. Starikovskaya. The k-mismatch problem revisited. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 2039–2052, 2016. doi:10.1137/1.9781611974331.ch142.

15 Raphaël Clifford, Markus Jalsenius, Ely Porat, and Benjamin Sach. Space lower bounds for online pattern matching. *Theor. Comput. Sci.*, 483:68–74, 2013. doi:10.1016/j.tcs.2012.06.012.

16 Raphaël Clifford and Ely Porat. A filtering algorithm for k-mismatch with don't cares. In *String Processing and Information Retrieval, 14th International Symposium, SPIRE 2007, Santiago, Chile, October 29-31, 2007, Proceedings*, pages 130–136, 2007. doi:10.1007/978-3-540-75530-2_12.

17 Raphaël Clifford and Benjamin Sach. Pseudo-realtime pattern matching: Closing the gap. In Amihood Amir and Laxmi Parida, editors, *Combinatorial Pattern Matching, 21st Annual Symposium, CPM 2010, New York, NY, USA, June 21-23, 2010. Proceedings*, volume 6129 of *Lecture Notes in Computer Science*, pages 101–111. Springer, 2010. doi:10.1007/978-3-642-13509-5_10.

18 Richard Cole and Ramesh Hariharan. Verifying candidate matches in sparse and wildcard matching. In John H. Reif, editor, *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 592–601. ACM, 2002. doi:10.1145/509907.509992.

19 Funda Ergün, Hossein Jowhari, and Mert Saglam. Periodicity in streams. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 13th International Workshop, APPROX 2010, and 14th International Workshop, RANDOM 2010, Barcelona, Spain, September 1-3, 2010. Proceedings*, pages 545–559, 2010. doi:10.1007/978-3-642-15369-3_41.

20 Nathan J Fine and Herbert S Wilf. Uniqueness theorems for periodic functions. *Proceedings of the American Mathematical Society*, 16(1):109–114, 1965.

21 Michael J Fischer and Michael S Paterson. String-matching and other products. Technical report, DTIC Document, 1974.

22 Zvi Galil and Joel I. Seiferas. Time-space-optimal string matching. *J. Comput. Syst. Sci.*, 26(3):280–294, 1983. doi:10.1016/0022-0000(83)90002-8.

23 Pawel Gawrychowski. Optimal pattern matching in LZW compressed strings. *ACM Transactions on Algorithms*, 9(3):25, 2013. doi:10.1145/2483699.2483705.

24 Shay Golan, Tsvi Kopelowitz, and Ely Porat. Streaming pattern matching with d wildcards. *CoRR*, abs/1605.16729, 2015. URL: http://arxiv.org/abs/1605.16729.

25 Monika Rauch Henzinger, Prabhakar Raghavan, and Sridar Rajagopalan. *External Memory Algorithms*, chapter Computing on data streams, pages 107–118. American Mathematical Society, Boston, USA, 1999.

26 Piotr Indyk. Faster algorithms for string matching problems: Matching the convolution bound. In *39th Annual Symposium on Foundations of Computer Science, FOCS'98, November 8-11, 1998, Palo Alto, California, USA*, pages 166–173. IEEE Computer Society, 1998. doi:10.1109/SFCS.1998.743440.

27 Markus Jalsenius, Benny Porat, and Benjamin Sach. Parameterized matching in the streaming model. In *30th International Symposium on Theoretical Aspects of Computer Science, STACS 2013, February 27 – March 2, 2013, Kiel, Germany*, pages 400–411, 2013. doi:10.4230/LIPIcs.STACS.2013.400.

28 Adam Kalai. Efficient pattern-matching with don't cares. In David Eppstein, editor, *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA.*, pages 655–656. ACM/SIAM, 2002. URL: http://dl.acm.org/citation.cfm?id=545381.545468.

29 Daniel M. Kane, Jelani Nelson, Ely Porat, and David P. Woodruff. Fast moment estimation in data streams in optimal space. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 745–754, 2011. doi:10.1145/1993636.1993735.

30 Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, 1987. doi:10.1147/rd.312.0249.

31 Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(2):323–350, 1977. doi:10.1137/0206024.

32 Gad M. Landau and Uzi Vishkin. Efficient string matching with k mismatches. *Theor. Comput. Sci.*, 43:239–249, 1986. doi:10.1016/0304-3975(86)90178-7.

33 Lap-Kei Lee, Moshe Lewenstein, and Qin Zhang. Parikh matching in the streaming model. In *String Processing and Information Retrieval – 19th International Symposium, SPIRE 2012, Cartagena de Indias, Colombia, October 21-25, 2012. Proceedings*, pages 336–341, 2012. doi:10.1007/978-3-642-34109-0_35.

34 S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005. doi:10.1561/0400000002.

35 S. Muthukrishnan and H. Ramesh. String matching under a general matching relation. In R. K. Shyamasundar, editor, *Foundations of Software Technology and Theoretical Computer Science, 12th Conference, New Delhi, India, December 18-20, 1992, Proceedings*, volume 652 of *Lecture Notes in Computer Science*, pages 356–367. Springer, 1992. doi:10.1007/3-540-56287-7_118.

36 Benny Porat and Ely Porat. Exact and approximate pattern matching in the streaming model. In *50th Annual IEEE Symp. on Foundations of Computer Science, FOCS 2009, Oct. 25-27, 2009, Atlanta, Georgia, USA*, pages 315–323, 2009. doi:10.1109/FOCS.2009.11.

37 Ely Porat and Ohad Lipsky. Improved sketching of hamming distance with error correcting. In *Combinatorial Pattern Matching, 18th Annual Symposium, CPM 2007, London, Canada, July 9-11, 2007, Proceedings*, pages 173–182, 2007. doi:10.1007/978-3-540-73437-6_19.

38 William F. Smyth and Shu Wang. A new approach to the periodicity lemma on strings with holes. *Theor. Comput. Sci.*, 410(43):4295–4302, 2009. doi:10.1016/j.tcs.2009.07.010.

How Hard is it to Find (Honest) Witnesses?

Isaac Goldstein*[1], Tsvi Kopelowitz†[2], Moshe Lewenstein‡[3], and Ely Porat[4]

1 **Bar-Ilan University, Ramat Gan, Israel**
goldshi@cs.biu.ac.il
2 **University of Michigan, Ann Arbor, USA**
kopelot@gmail.com
3 **Bar-Ilan University, Ramat Gan, Israel**
moshe@cs.biu.ac.il
4 **Bar-Ilan University, Ramat Gan, Israel**
poratelyp@cs.biu.ac.il

Abstract

In recent years much effort has been put into developing polynomial-time conditional lower bounds for algorithms and data structures in both static and dynamic settings. Along these lines we introduce a framework for proving conditional lower bounds based on the well-known 3SUM conjecture. Our framework creates a *compact representation* of an instance of the 3SUM problem using hashing and domain specific encoding. This compact representation admits false solutions to the original 3SUM problem instance which we reveal and eliminate until we find a true solution. In other words, from all *witnesses* (candidate solutions) we figure out if an *honest* one (a true solution) exists. This enumeration of witnesses is used to prove conditional lower bounds on *reporting* problems that generate all witnesses. In turn, these reporting problems are then reduced to various decision problems using special search data structures which are able to enumerate the witnesses while only using solutions to decision variants. Hence, 3SUM-hardness of the decision problems is deduced.

We utilize this framework to show conditional lower bounds for several variants of convolutions, matrix multiplication and string problems. Our framework uses a strong connection between all of these problems and the ability to find *witnesses*.

Specifically, we prove conditional lower bounds for computing partial outputs of convolutions and matrix multiplication for sparse inputs. These problems are inspired by the open question raised by Muthukrishnan 20 years ago [22]. The lower bounds we show rule out the possibility (unless the 3SUM conjecture is false) that almost linear time solutions to sparse input-output convolutions or matrix multiplications exist. This is in contrast to standard convolutions and matrix multiplications that have, or assumed to have, almost linear solutions.

Moreover, we improve upon the conditional lower bounds of Amir et al. [5] for histogram indexing, a problem that has been of much interest recently. The conditional lower bounds we show apply for both reporting and decision variants. For the well-studied decision variant, we show a full tradeoff between preprocessing and query time for every alphabet size > 2. At an extreme, this implies that no solution to this problem exists with subquadratic preprocessing time and $\tilde{O}(1)$ query time for every alphabet size > 2, unless the 3SUM conjecture is false. This is in contrast to a recent result by Chan and Lewenstein [9] for a binary alphabet.

While these specific applications are used to demonstrate the techniques of our framework, we believe that this novel framework is useful for many other problems as well.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity

* This research is supported by the Adams Foundation of the Israel Academy of Sciences and Humanities.
† This research is supported by NSF grants CCF-1217338, CNS-1318294, and CCF-1514383.
‡ This research is supported by a BSF grant 2010437 and a GIF grant 1147/2011.

Keywords and phrases 3SUM, convolutions, matrix multiplication, histogram indexing

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.45

1 Introduction

In recent years much effort has been invested towards developing polynomial time lower bounds for algorithms and data structures in both static and dynamic settings. This effort is directed towards obtaining a better understanding of the complexity class P for well-studied problems which seem hard in the polynomial sense. The seminal paper by Gajentaan and Overmars [13] set the stage for this approach by proving lower bounds for many problems in computational geometry conditioned on the 3SUM conjecture. In the 3SUM problem we are given a set A of n integers and we need to establish if there are $a, b, c \in A$ such that $a+b+c = 0$. This problem has a simple $O(n^2)$ algorithm (and some poly-logarithmic improvements in [6, 17]) but no truly subquadratic algorithm is known, where truly subquadratic means $O(n^{2-\epsilon})$ for some $\epsilon > 0$. The 3SUM conjecture states that no truly subquadratic algorithm exists for the 3SUM problem.

Based on this conjecture, there has been a recent extensive line of work establishing conditional lower bounds (CLBs) for many problems in a variety of fields other than computational geometry, including many interesting dynamic problems, see e.g. [1, 2, 3, 4, 19, 23].

1.1 Decision and Reporting Problems

Algorithmic problems come in many flavors. The classic one is the *decision* variant. In this variant, we are given an instance of a problem and we are required to decide if it has some property or not. Some examples include: (1) given a 3-CNF formula we may be interested in deciding if it satisfiable by some truth assignment; (2) given a bipartite graph we may be interested in deciding if the graph has a perfect matching; (3) given a text T and a pattern P we may be interested in deciding if P occurs in T. It is well-known that the first example is NP-complete while the two others are in P. An instance that has the property in question has at least one *witness* that proves the existence of the property. In the examples above a witness is: (1) a satisfying assignment; (2) a perfect matching in the graph; (3) a position of an occurrence of P in T. Sometimes, we are not only interested in understanding if a witness *exists*, but rather we wish to *enumerate* all of the witnesses. This is the *reporting* variant of the problem. In the examples mentioned above the goal of the reporting variant is to: (1) enumerate all satisfying assignments; (2) enumerate all perfect matchings; (3) enumerate all occurrences of P in T. For the first two examples it is known from complexity theory that it is most likely hard to count the number of witnesses (not to mention reporting them) (these are #P-complete problems), while the third example can be solved by classic linear time algorithms.

In this paper we investigate the interplay between the decision and reporting variants of algorithmic problems and present a systematic framework that is used for proving CLBs for these variants. We expect this framework to be useful for proving CLBs on other problems not considered here.

1.2 Our Framework

We introduce and follow a framework that shows 3SUM-hardness of decision problems via their reporting versions. The high-level idea is to reduce an instance of 3SUM to an instance of a reporting problem, and then reduce the instance of a reporting problem to several instances of its decision version using a sophisticated search structure. The outline of this framework is described next.

- **Compact Representation**. One of the difficulties in proving CLBs based on the 3SUM conjecture is that the input universe for 3SUM could be too large for accommodating a reduction to a certain problem. To tackle this, we *embed* the universe using special hashing techniques. This is sometimes coupled with a secondary problem-specific encoding scheme in order to match the problem at hand.
- **Reporting**. The embedding in the first step may introduce false-positives. To tackle this, we report *all* the candidate solutions (witnesses) for the embedded 3SUM instance, in order to verify if a true solution (an honest witness) to 3SUM really exists. This is where we are able to say something about the difficulty of solving reporting problems. This is done by reducing the embedded 3SUM instance to an instance of such a reporting problem, if it provides an efficient way to find all the false-positives. In some cases, such reductions reveal tradeoff relationships between the preprocessing time and reporting/query time.
- **Reporting via Decision**. In this step the goal is to establish 3SUM-hardness of a decision problem. To do so we reduce an instance of the reporting version of the problem to instances of the decision version by creating a data structure on top of the many instances of the decision version. This data structure allows us to efficiently report all of the elements in the output of the instance of the reporting version. By constructing the data structure in different ways we obtain varying CLBs for the decision variants depending on the specific structure that we use.

By following this route we introduce new CLBs for some important problems which are discussed in detail in Section 2. We point out that the embedding in the first step follows along the lines of [23] and [19]. However, in some cases we also add an additional encoding scheme to fit the needs of the specific problem at hand.

Implications. In Section 2 we discuss three applications from two different domains which utilize our framework for proving CLBs, thereby demonstrating the usefulness of our framework. Table 1 summarizes these results. Of particular interest are new results on Histogram Indexing (defined in Section 2) which, together with the algorithm of [9], demonstrate a sharp separation when allowing truly subquadratic preprocessing time between binary and trinary alphabet settings. Moreover, our framework is the first to obtain a CLB for the reporting version, which, as opposed to the decision variant, also holds for the binary alphabet case.

2 Applications

Convolution Problems

The *convolution* of two vectors $u, v \in \{\mathbb{R}^+ \cup \{0\}\}^n$ is a vector w, such that $w[k] = \sum_{i=0}^{k} u[i]v[k-i]$ for $0 \leq k \leq 2n-2$. Computing the convolution of u and v takes $O(n \log n)$ time using the celebrated FFT algorithm. Convolutions are used extensively in many areas including signal processing, communications, image compression, pattern matching, etc. A *convolution witness* for the kth entry in w is a pair (a, b) such that $a+b = k$ and $u[a] \cdot v[b] > 0$. In other words, the witnesses of entry k in w are all values i that contribute a non-zero value to the summation $w[k] = \sum_{i=0}^{k} u[i]v[k-i]$. The first convolution problem we consider is the *convolution witnesses problem* which is defined as follows.

▶ **Definition 1.** In the **convolution witnesses problem** we preprocess two vectors $u, v \in \{\mathbb{R}^+ \cup \{0\}\}^n$ and their convolution vector w, so that given a query integer $0 \leq k \leq 2n-2$, we list *all* convolution witnesses of index k in w.

We prove the following CLB for the convolution witnesses problem that holds even if u and v are binary vectors and all numbers in w are non-negative integers.

▶ **Theorem 2.** *Assume the 3SUM conjecture is true. Then for any constant $0 < \alpha < 1$, there is no algorithm solving the convolution witnesses problem with $O(n^{2-\alpha})$ expected preprocessing time and $O(n^{\alpha/2-\Omega(1)})$ expected amortized query time* ***per witness****.*

Theorem 2 implies that when using only truly subquadratic preprocessing time one is required to spend a significant polynomial amount of time on every single witness. In particular, this means that, assuming the 3SUM conjecture, one cannot expect to find witnesses much faster than following the naive algorithm for computing convolution naïvely according to the convolution definition. This is in contrast to the decision version of the problem, where we only ask if a witness exists. This variant is easily solved using constant query time after a near linear time preprocessing procedure (computing the convolution itself).

Another variation of the convolution problem which we consider is the *sparse convolution problem.* There are two different problems named sparse convolution, both appearing as open questions in a paper by Muthukrishnan [22]. In the first, which is now well understood, we are given Boolean vectors u and v of lengths N and M, where $M < N$. There are n ones in u, m ones in v and z ones in w, where w is the Boolean convolution vector of u and v. The goal is to report the non-zero elements in w in $\tilde{O}(z)$ time. This problem has been extensively studied, and the goal has been achieved; see for example [9, 11, 15]. The second variant which we call *partial convolutions* is as follows.

▶ **Definition 3.** The **partial convolution problem** on two vectors u and v of real numbers (of length N and M respectively, where $M < N$) and a set S of indices is to compute, for each $i \in S$, the value of the i-th element in the convolution of u and v.

Muthukrishnan in [22] asked if it is possible to compute a partial convolution significantly faster than the time needed to compute a (classic) convolution. We prove a CLB based on the 3SUM conjecture, that holds also for the special case of Boolean vectors, and, therefore, also for the special case in which we only want to know if the output values at indices in S are zero or more. Moreover, we focus on the important variant of this problem that deals with the case where the two input vectors have only $n = O(N^{1-\Omega(1)})$ ones and are both given implicitly (specifying only the indices of the ones). Our results also extend to the indexing version of the partial convolution problem, which we call the *partial convolution indexing problem*, and is defined as follows.

▶ **Definition 4.** The partial convolution indexing problem is to preprocess an N-length vector u of real numbers and a set of indices S to support the following queries: given an M-length vector v ($M < N$) of real numbers, for each $i \in S$ compute the value of the i-th element of the convolution of u and v.

Once again this variant already relevant when the input is Boolean and sparse, i.e. u and v have $n = O(N^{1-\Omega(1)})$ ones and are represented implicitly by specifying their indices.

We prove the following CLBs for these problems with the help of our framework.

▶ **Theorem 5.** *Assume the 3SUM conjecture is true. Then there is no algorithm for the partial convolution problem with $O(N^{1-\Omega(1)})$ time, even if $|S|$ and the number of ones in both input vectors are less than $N^{1-\Omega(1)}$.*

▶ **Theorem 6.** *Assume the 3SUM conjecture is true. Then there is no algorithm for the partial convolution indexing problem with $O(N^{2-\Omega(1)})$ preprocessing time and $O(N^{1-\Omega(1)})$ query time, even if both $|S|$ and the number of ones of the input vectors are $O(N^{1-\Omega(1)})$.*

As mentioned above, the convolution of vectors of length N can be computed in $\tilde{O}(N)$ time with the FFT algorithm. However, in the partial convolution problem and partial convolution indexing problem, despite the input vectors being sparse and represented sparsely (specifying only the $O(N^{1-\Omega(1)})$ indices of the ones in each vector), and despite the portion of the output we need to compute being sparse ($|S| = O(N^{1-\Omega(1)})$), no linear time algorithm (in $n = O(N^{1-\Omega(1)})$) exists, unless the 3SUM conjecture is false.

Notice that the partial convolution problem and its indexing variant are *decision* problems, since they require a *decision* for each location $i \in S$, whether $w[i] > 0$ or not. This is in contrast to the convolution witnesses problem, which is a reporting problem, as it requires the reporting of *all* of the witnesses for $w[i]$.

To prove CLBs for the convolution problems we follow our framework. That is, we first use a hash function to embed a 3SUM instance to a *smaller* universe. This mapping introduces false-positives, which we enumerate by utilizing the reporting problem of convolution witnesses. To solve the reporting version we reduce it to several instances of a decision problem, partial convolution or its indexing variant, by constructing a suitable data structure. Tying it all together leads to CLBs for both the reporting and decision problems.

Matrix Problems

We also present some similar CLBs for matrices.

▶ **Definition 7.** The **partial matrix multiplication problem** on two $N \times N$ matrices A and B of real numbers and a set of entries $S \subseteq N \times N$ is to compute, for each $(i, j) \in S$, the value $(A \times B)[i, j]$.

The indexing variant of this problem is defined as follows.

▶ **Definition 8.** The **partial matrix multiplication indexing problem** is to preprocess an $N \times N$ matrix A of real numbers and a collection $\mathcal{S} = \{S_1, S_2, ..., S_k\}$ of sets of entries, where $S_i \subseteq N \times N$, so that given a sequence $B_1, \ldots, B_k$ of $N \times N$ matrices of real numbers, we enumerate the entries of $A \times B_i$ that correspond to S_i.

For $\mathcal{S} = \{S_1, S_2, ..., S_k\}$ let $SIZE(\mathcal{S}) = \sum_{i=1}^{k} |S_i|$. We prove the following CLBs, which hold also for the special case of Boolean multiplication assuming that the input is given implicitly by specifying only the indices of the ones.

▶ **Theorem 9.** *Assume the 3SUM conjecture is true. Then there is no algorithm for the partial matrix multiplication problem running in $O(N^{2-\Omega(1)})$ expected time, even if $|S|$ and the number of ones in the input matrices is $O(N^{2-\Omega(1)})$.*

▶ **Theorem 10.** *Assume the 3SUM conjecture is true. Then there is no algorithm for the partial matrix multiplication indexing problem with $O(SIZE(\mathcal{S}))$ preprocessing time and $O(N^{2-\Omega(1)})$ query time.*

Matrix multiplication, and in particular Boolean matrix multiplication, can be solved in $\tilde{O}(n^\omega)$ time, where $\omega \approx 2.373$ [14, 25]. Many researchers believe that the true value of ω is 2. This belief implies that the running time for computing the product of two Boolean matrices is proportional to the size of the input matrices and the resulting output. However, our results demonstrate that such a result is unlikely to exist for sparse versions of the problem, where the number of ones in the matrices is $O(N^{2-\Omega(1)})$ and we are interested in only a partial output matrix (only $O(N^{2-\Omega(1)})$ entries of the matrix product).

To prove Theorem 9 and 10 we follow our framework. The process is very similar to the path for proving CLBs for convolution problems. In fact, instead of considering a reporting version of the partial matrix multiplication problem for proving these CLBs, we once again utilize the reporting problem of convolution witnesses. However, this time we transform the convolution witnesses to the matrix multiplication problems using a more elaborate data structure. The main difficulty in this transformation is to guarantee the sparsity of both the input and the required output. This transformation illustrates how a reporting version of a problem can be used to prove CLBs for decision versions of other problems, by changing the way we look for honest witnesses.

String Problems

Another application of our framework, which is seemingly unrelated to the previous two, is the problem of histogram indexing. A *histogram*, also called a *Parikh vector*, of a string T over alphabet Σ is a $|\Sigma|$-length vector containing the character count of T. For example, for $T = abbbacab$ the histogram is $\psi(T) = (3, 4, 1)$.

▶ **Definition 11.** In the **histogram indexing problem** we preprocess a string T to support the following queries: given a query Parikh vector ψ, return whether there is a substring T' of T such that $\psi(T') = \psi$.

▶ **Definition 12.** In the **histogram indexing reporting problem** we preprocess a string T to support the following queries: given a query Parikh vector ψ, report indices of T at which a substring T' of T begins such that $\psi(T') = \psi$.

The problem of histogram indexing (not the reporting version) is sometimes called *jumbled indexing.* It has received much attention in recent years. For example, for binary alphabets – that is histograms of length 2 – there is a straightforward algorithm with $O(n^2)$ preprocessing time and constant query time, see [10]. Burcsi et al. [8] and Moosa and Rahman [20] improved the preprocessing time to $O(n^2/\log n)$. Using the four-Russian trick a further improvement was achieved by Moosa and Rahman [21]. Then, using a connection to the recent improvement of all-pairs-shortest path by Williams [24], as observed by Bremner et al. [7] and by Hermelin et al. [16], the preprocessing time was further reduced to $O(\frac{n^2}{2^{\Omega(\log n)^{0.5}}})$. Finally, Chan and Lewenstein [9] presented an $O(n^{1.859})$ preprocessing time algorithm for the problem with constant query time. For non-binary alphabets some progress was achieved in the work by Kociumaka et al. [18] and even further achievement was shown in [9]. On the negative side, some CLBs were recently shown by Amir et al. [5].

We follow our framework and first obtain CLBs for the reporting version of histogram indexing. This is the first time CLBs are shown for the reporting version. Moreover, these CLBs apply to binary alphabets, as opposed to the decision version in which there currently is no CLB known for binary alphabets. The CLBs for the reporting version admit a full tradeoff between preprocessing and query time. For the decision variant, we improve upon the CLB by Amir et al. [5] by presenting full-tradeoffs between preprocessing and query

Table 1 Summary of CLBs proved in this paper. In this table N is the size of vectors, strings and the dimension of matrices. #1 refers to the number of ones in the input. The rows in this table are interpreted to mean that there is no data structure that beats these preprocessing, query, and reporting (if exists) complexities at the same time. For partial convolution and matrix multiplication the CLB on the preprocessing time should be interpreted as a CLB on the total running time as these are offline problems.

Problem *(Type)*	Preprocessing Time	Query Time	Reporting Time	Remarks
Conv. Witnesses *(Reporting)*	$\Omega(N^{2-\alpha})$	$\Omega(N^{1-\alpha/2})$	$\Omega(N^{\alpha/2-o(1)})$	[Theorem 2] $0 < \alpha < 1$
Partial Conv. *(Decision)*	$\Omega(N^{1-o(1)})$	—	—	[Theorem 5] Sparse input: $\#1 < N^{1-\Omega(1)}$; Sparse required output: $\|S\| < N^{1-\Omega(1)}$
Partial Conv. Indexing *(Decision)*	$\Omega(N^{2-o(1)})$	$\Omega(N^{1-o(1)})$	—	[Theorem 6] Sparse input: $\#1 < N^{1-\Omega(1)}$; Sparse required output: $\|S\| < N^{1-\Omega(1)}$
Partial Matrix Mult. *(Decision)*	$\Omega(N^{2-o(1)})$	—	—	[Theorem 9] Sparse input: $\#1 < N^{2-\Omega(1)}$; Sparse required output: $\|S\| < N^{2-\Omega(1)}$
Partial Matrix Mult. Indexing *(Decision)*	$\Omega(SIZE(S))$	$\Omega(N^{2-o(1)})$	—	[Theorem 10] Sparse input: $\#1 < N^{2-\Omega(1)}$; Sparse required output: $\|S_i\| < N^{2-\Omega(1)}$; $SIZE(S) = \sum_{i=1}^{k} \|S_i\|$
Histogram Reporting *(Reporting)*	$\Omega(N^{2-\frac{2\gamma}{\ell+\gamma}-o(1)})$	$\Omega(N^{1-\frac{\gamma}{\ell+\gamma}-o(1)})$	$\Omega(N^{\frac{\gamma\ell}{\ell+\gamma}-\frac{2\gamma}{\ell+\gamma}-o(1)})$	[Theorem 13] alphabet size: $\ell \geq 2$; $0 < \gamma < \ell$
Histogram Indexing *(Decision)*	$\Omega(N^{2-\frac{2(1-\alpha)}{\ell-1-\alpha}-o(1)})$	$\Omega(N^{1-\frac{1+\alpha(\ell-3)}{\ell-1-\alpha}-o(1)})$	—	[Theorem 14] alphabet size: $\ell > 2$; $0 \leq \alpha \leq 1$

time based on the standard 3SUM conjecture. Specifically, our new CLB implies that no solution to the histogram indexing problem exists with subquadratic preprocessing time and $\tilde{O}(1)$ query time for every alphabet size bigger than 2, unless the 3SUM conjecture is false. This demonstrates a sharp separation between binary and trinary alphabets, since Chan and Lewenstein [9] introduced an algorithm for histogram indexing on binary alphabets with $\tilde{O}(n^{1.859})$ preprocessing time and constant query time.

The CLBs are summarized by the following theorems.

▶ **Theorem 13.** *Assume the 3SUM conjecture is true. Then the histogram reporting problem for an N-length string and constant alphabet size $\ell \geq 2$ cannot be solved using $O(N^{2-\frac{2\gamma}{\ell+\gamma}-\Omega(1)})$ preprocessing time, $O(N^{1-\frac{\gamma}{\ell+\gamma}-\Omega(1)})$ query time and $O(N^{\frac{\gamma\ell}{\ell+\gamma}-\frac{2\gamma}{\ell+\gamma}-\Omega(1)})$ reporting time per item, for any $0 < \gamma < \ell$.*

▶ **Theorem 14.** *Assume the 3SUM conjecture holds. Then the histogram indexing problem for a string of length N and constant alphabet size $\ell \geq 3$ cannot be solved with $O(N^{2-\frac{2(1-\alpha)}{\ell-1-\alpha}-\Omega(1)})$ preprocessing time and $O(N^{1-\frac{1+\alpha(\ell-3)}{\ell-1-\alpha}-\Omega(1)})$ query time.*

The main structure of these proofs follows our framework. We first embed a 3SUM instance and encode it in a string with limited length. We then report the false-positives using the reporting variant of the histogram indexing problem, which implies CLBs for this variant. Finally, we reduce the reporting version to the decision version thereby obtaining CLBs for the decision version. The reduction utilizes a sophisticated data structure for reporting witnesses using many instances of the decision version.

3 Preliminaries

In the basic 3SUM problem we are given a set A of n integers and we need to answer whether there are $a, b, c \in A$ such that $a + b + c = 0$. In a common variant of the classic problem, which we also denote by 3SUM, three arrays A, B and C are given and we need to answer whether there are $a \in A, b \in B, c \in C$ such that $a + b + c = 0$. Both versions have the same computational cost (see [13]). There are some other variants of the 3SUM problem shown to be as hard as 3SUM up to poly-logarithmic factors. One such variant is Convolution3SUM, shown to be hard by Pătraşcu [23], see also [19]. In Convolution3SUM A is an ordered set and we need to answer whether there exist indices $0 \leq i, j \leq n-1$ such that $A[i] + A[j] = A[i+j]$. We also define DiffConv3SUM, in which we are given an ordered set A and we need to verify whether there exists $0 \leq i, k \leq n-1$ such that $A[k] - A[i] = A[k-i]$. It is easy to see that this is equivalent to Convolution3SUM.

Let $\mathcal{H}$ be a family of hash functions from $[u] \to [m]$.

$\mathcal{H}$ 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 $h : [u] \to [m]$ 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 $O(m)$. See [19] for constructions of families that are almost-linear and almost-balanced (see also [6, 12]).

For simplicity of presentation, and following the footsteps of previous papers that have used such families of functions [6, 23], we assume for the rest of the paper that almost linearity implies that for any $h \in \mathcal{H}$ and any $x, x' \in [u]$ we have $h(x) + h(x') \equiv h(x + x') \pmod{m}$.

There are actually two assumptions taking place here. The first is that there is only one option of so-called linearity. Overcoming this assumption imposes only a constant factor overhead. The second assumption is that $c_h = 0$. However, the constant c_h only affects offsets in our algorithm in a straightforward and not meaningful way, so we drop it in order to avoid clutter in our presentation.

4 Convolution Witnesses

We first prove a CLB for the convolution witnesses problem. We begin with a lemma which has elements from the proof of Pătraşcu's reduction [23] and from [6]. However, the lemma diverges from [23] by treating the hashed subsets differently. Specifically, many special 3SUM subproblems are created and then reduced to convolution witnesses.

We say that a binary vector of length n is *r-sparse* if it contains at most r 1's. An instance of convolution witnesses problem (u, v, w) is *(n, R)-sparse* if u and v are both of length n and n/R-sparse.

▶ **Lemma 15.** *Let sequence $A = \langle x_1, \cdots, x_n\rangle$ be an instance of Convolution3SUM. Let $R = O(n^\delta)$, where $0 < \delta < 0.5$ is a constant. There exists a truly subquadratic reduction from the instance A to $O(R^2)$ (n, R)-sparse instances of convolution witnesses problem for which we need to report $O(n^2/R)$ witnesses (over all instances).*

Proof. We use an almost-linear, almost-balanced, hash function $h : U \to [R]$ and create R buckets $B_0, \cdots, B_{R-1}$ where each B_a contains the indices of all elements $x_i \in A$ for which $h(x_i) = a$. Since h is almost-balanced the expected overall number of elements in buckets with more than $3n/R$ elements is $O(R)$. For each index i in an overflowed bucket, we verify whether $x_i + x_j = x_{i+j}$ for every other j in $O(n)$ time. Hence, we verify whether any index in an overflowed bucket is part of a Convolution3SUM solution in $O(nR)$ expected time. Since $R = O(n^{1-\Omega(1)})$ the expected time is truly subquadratic time.

We now assume that every bucket contains at most $3n/R$ elements. From the properties of almost-linear hashing, if $x_i + x_j = x_{i+j}$ then $h(x_i) + h(x_j) \bmod R = h(x_{i+j}) \bmod R$. Hence, if $x_i + x_j = x_{i+j}$ then $i \in B_a, j \in B_b$ implies that $i + j \in B_{a+b \bmod R}$.

Every three buckets form an instance of 3SUM and are uniquely defined by a and b. Hence, there are $R(R-1)/2 = O(R^2)$ 3SUM subproblems each on $O(n/R)$ elements from the small universe $[n]$. However, h may generate false positives. So, we must be able to verify that any 3SUM solution (a witness) for any instance is indeed a solution (an honest witness) for the problem on A. The number of false positives is expected to be $O(n^2/R)$ over all $O(R^2)$ instances, see [6]. So, we need an efficient tool to report each such witness in order to be able to solve Convolution3SUM.

To obtain such a tool, we reduce the problem to the convolution setting in the following way. We generate a characteristic vector v_a of length n for every set B_a ($v_a[i] = 1$ if $i \in B_a$ and $v_a[i] = 0$ otherwise, for $0 \le i < n$). This vector will be $3n/R$-sparse, since $|B_a| \le 3n/R$. Note that: $i \in B_a, j \in B_b$ and $\mathrm{i+j \in B_{a+b \bmod R} \iff v_a[i] = 1, v_b[j] = 1}$ and $\mathrm{v_{a+b \bmod R}[i+j] = 1}$.

Now, for each pair of vectors, v_a and v_b, we generate their convolution. Let $v = v_a * v_b$ be the convolution of v_a and v_b, and let $\ell = v[i + j]$. If $v_{a+b \bmod R}[i + j] = 1$, then we need to extract the ℓ witnesses of $v[i + j]$. For each witness (i, j) we check whether $x_i + x_j = x_{i+j}$. We note that if, while verifying, we discover that the overall number of the false-positives exceeds expectation (cn^2/R, for some constant c) by more than twice we rehash.

Thus, we see that Convolution3SUM can be solved by generating $O(R^2)$ (n, R)-sparse instances of convolution witnesses problem. These instances are computed in $O(nR^2)$ time, which is truly subquadratic as $R = O(n^\delta)$ for $\delta < 1/2$. ◀

It now follows that:

▶ **Theorem 2** (restated). *Assume the 3SUM conjecture is true. Then for any constant $0 < \alpha < 1$, there is no algorithm solving the convolution witnesses problem with $O(n^{2-\alpha})$ expected preprocessing time and $O(n^{\alpha/2-\Omega(1)})$ expected amortized query time **per witness**.*

Proof. We make use of Lemma 15 and its parameter R. In particular, the total cost of solving Convolution3SUM is at most $O(R^2 \cdot P(n,R) + n^2/R \cdot Q(n,R))$ expected time, where $P(n,R)$ is the time needed to preprocess an (n,R)-sparse instance of a convolution witness and $Q(n,R)$ is the time per witness query for an (n,R)-sparse instance of a convolution witness.

If we choose $R = n^{\alpha/2-\Omega(1)}$ we have that for $P(n) = O(n^{2-\alpha})$ and $Q(n) = O(n^{\alpha/2-\Omega(1)})$ we solve Convolution3SUM in $O(n^{2-\Omega(1)})$ time which is truly subquadratic. ◀

5 From Reporting to Decision I: Hardness of Partial Convolutions

We further consider the problem of reporting witnesses for convolutions. However, now we use the third step of our framework. We will construct a search data structure over decision problems which will allow us to efficiently search for witnesses. This will be our method for proving CLBs for the decision problems of partial convolutions [22]. Specifically, we intend to generate a data structure that uses convolutions on small sub-vectors of the input vectors in order to solve the problem. However, the data structure cannot be fully constructed as it will be too large. Hence, the construction is partial and we defer some of the work to the query phase.

We start with Lemma 15, and focus on an (n,R)-sparse instance of the convolution witnesses problem(u,v,w). We generate a specialized search tree for efficiently finding witnesses, which is created in an innovative way exploiting the sparsity of the input.

5.1 Search Tree Construction

Assume, without loss of generality, that n is a power of 2. We construct a binary tree in the following way. First, we generate the root of the tree with the convolution of v and u. Then we split u into 2 sub-vectors, say u_1 and u_2, each containing exactly $n/(2R)$ 1s. For each sub-vector we generate nodes that are children of the root, where the first node contains the convolution of v and u_1 and the second node contains the convolution of v and u_2. We continue this construction recursively so that at the ith recursive level we partition u into 2^i sub-vectors each containing $n/(2^iR)$ 1s. A vertex at level i represents the convolution of v and a sub-vector u_A containing $n/2^iR$ 1s. The vertex has two children, one represents the convolution of v and the sub-vector of u_A with the first $n/2^{i+1}R$ 1s of u_A (denoted by $u_{A,1}$). The other represents the convolution of v and the rest of u_A with the other $n/2^{i+1}R$ 1s (denoted by $u_{A,2}$). We stop the construction at the leaf level in which u is split to sub-vectors that each one of them contains X/R 1s from u, for some $X < n$ to be determined later. Calculating the convolution in each vertex is done bottom-up. First, we calculate the convolution for each vertex in the leaf level. Then, we use these results to calculate the convolution of the next level upwards. Specifically, if we have vertex that represent the convolution v and some sub-vector u_A and it has two children one which represents the convolution of v and $u_{A,1}$ and the other which represents the convolution of v and $u_{A,2}$, then $(v * u_A)[k] = (v * u_{A,1})[k] + (v * u_{A,2})[k - l_1]$ for every $k \in [0, n + l_1 + l_2 - 1]$, where l_1 and l_2 are the lengths of $u_{A,1}$ and $u_{A,2}$ respectively, and we consider the value of out of range entries as zero. This way we continue to calculate all the convolutions in the tree until reaching its root.

Construction Time. It is straightforward to verify that the total cost of the construction procedure is dominated by the time of constructing the lowest level of the binary tree. In this level, we have n/X sub-vectors of u as each of them has X/R 1's and the total number of 1s in u is n/R. We calculate the convolution of v with each of these sub-vectors, which can be done in $\tilde{O}(n)$ time. Thus, the total time needed to build the tree is $\tilde{O}(n^2/X)$. herefore, the total time for calculating the binary trees for all $O(R^2)$ (n, R)-sparse instances of the convolution witnesses problem is $\tilde{O}(R^2n^2/X)$.

Witness Search. To search for a witness we begin from the root of the binary tree and traverse down to a leaf containing a non-zero value in the result of the convolution at the query index (adjusting the index as needed while moving down the structure). The search for a leaf costs logarithmic time per query (as the tree has logarithmic height and in each level we just need to find a child with a non-zero value in the convolution it represents in the specific index of interest). Within the leaf, representing the convolution of v and some sub-vector u_A of u we can simply find a witness in $\tilde{O}(X/R)$ time as u_A contains just X/R 1s. Thus, as we have $O(n^2/R)$ false-positives over all $O(R^2)$ instances, the total time for finding all them is $\tilde{O}(n^2X/R^2)$.

Consequently, using the binary tree for solving Convolution3SUM will cost $\tilde{O}(R^2n^2/X + n^2X/R^2)$ time, which for $X = R^2$ is $\tilde{\Theta}(n^2)$ time. Since the tradeoff between the preprocessing time and query time meets at n^2, any improvement to the running time of either of them will imply a subquadratic solution for the Convolution3SUM problem.

5.2 Conditional Lower Bounds for Partial Convolution

As a consequence of our discussion above we obtain the following results regarding partial convolution and its indexing variant:

▶ **Theorem 5** (restated). *Assume the 3SUM conjecture is true. Then there is no algorithm for the partial convolution problem with $O(N^{1-\Omega(1)})$ time, even if $|S|$ and the number of ones in both input vectors are less than $N^{1-\Omega(1)}$.*

Proof. We make use of Lemma 15. In order to construct the binary tree as described in Section 5.1, we need to be able compute the convolution of v with some sub-vector of u for each leaf in the tree (all other convolution can be calculated efficiently from the convolutions in the leaves as described in the previous section). Recall that both input vectors have length $N = n$, n/R 1s (which is $O(N^{1-\Omega(1)})$ for $R = n^a$, where a is a positive constant), and we are interested in finding their convolution result only at the $O(n/R)$ indices (that is, $|S| = O(N^{1-\Omega(1)})$). If we preprocess the input for partial convolution in truly sublinear time (for example, proportional to n/R) then the total time for constructing all the search trees will be $O(R^2n^{2-\Omega(1)}/X)$ while the total query time will remain $O(n^2X/R^2)$. Choosing $X = n^c$ for small enough constant c and setting $R = X$, we obtain a subquadratic solution to Convolution3SUM. ◀

▶ **Theorem 6** (restated). *Assume the 3SUM conjecture is true. Then there is no algorithm for the partial convolution indexing problem with $O(N^{2-\Omega(1)})$ preprocessing time and $O(N^{1-\Omega(1)})$ query time, even if both $|S|$ and the number of ones of the input vectors are $O(N^{1-\Omega(1)})$.*

Proof. Use Lemma 15 and the previous discussion. If the preprocessing time for the partial convolution indexing problem is truly subquadratic and queries are answered in truly sublinear time then the total time for constructing all the structures for all $O(R^2)$ instances is $O(R^2[n^{2-\Omega(1)} + n^{1-\Omega(1)} \cdot n/X])$ while the total time for all of the queries remains $O(n^2X/R^2)$ (note that $N = n$). Choosing $X = n^c$ for small enough constant c and setting $R = X$, we obtain a subquadratic algorithm for Convolution3SUM. ◀

6 From Reporting to Decision II: Hardness of Partial Matrix Multiplication

We present another transformation from the reporting problem of convolution witnesses to decision problems. This time we prove CLBs for the partial matrix multiplication and its indexing variant. The main difficulty in this transformation is to ensure the sparsity of both input and required output. The CLBs that we prove are stated as follows (full details and proofs will appear in the full version of this paper).

▶ **Theorem 9** (restated). *Assume the 3SUM conjecture is true. Then there is no algorithm for the partial matrix multiplication problem running in $O(N^{2-\Omega(1)})$ expected time, even if $|S|$ and the number of ones in the input matrices is $O(N^{2-\Omega(1)})$.*

▶ **Theorem 10** (restated). *Assume the 3SUM conjecture is true. Then there is no algorithm for the partial matrix multiplication indexing problem with $O(SIZE(S))$ preprocessing time and $O(N^{2-\Omega(1)})$ query time.*

7 Hardness of Data Structures for Histogram Indexing

In order to prove a CLB for both the histogram indexing problem and the histogram (indexing) reporting problem, we will first focus on reducing 3SUM to the histogram reporting problem, and then turn our focus to reducing the the histogram reporting problem to the histogram indexing problem.

7.1 Reducing Convolution3SUM to Histogram Reporting

We are given an ordered set A of integers $x_1, x_2, ..., x_n$ for which we want to solve DiffConv3SUM. Our methodology here is to encode the input integers into a compact string S so that histogram indexing with carefully chosen query patterns implies a solution to DiffConv3SUM. Since the size of the universe of the input integers can be as large as n^3, we hash down the universe size while (almost) maintaining the linearity property of the input. To do this, we make use of an almost-linear almost-balanced hash function $h : U \to [R]$ as defined in Section 3, and apply h to all of the input integers.

After utilizing h to compress the input range, we are ready to encode the input and create the string S. To do this, we encode each $h(x_k)$ separately, and then concatenate the encodings in the same order as their corresponding original integers in A. We use the following encoding scheme, using an alphabet $\Sigma = \{\sigma_0, \sigma_1, , ..., \sigma_{\ell-1}\}$. Some other encoding schemes, which surprisingly provide the same bounds, will be presented in the full version of this paper.

Encoding 1. The encoding will consist of two separate partial encodings concatenated together. The first partial encoding is partitioned into ℓ parts which together will represent $h(x_k)$ in base $R^{1/\ell}$. For $0 \le j \le \ell - 1$ the jth part of this first partial encoding is a unary representation of $p_{j,h(x_k)} = \lfloor h(x_k)/R^{j/\ell} \rfloor \bmod R^{1/\ell}$ using σ_j, and is denoted by $enc(j, h(x_k)) = \sigma_j^{p(j,h(x_k))}$. The first partial encoding of $h(x_k)$, which we also call a *regular encoding* of $h(x_k)$, is $enc_\ell(h(x_k)) = enc(0, h(x_k))enc(1, h(x_k)) \cdots enc(\ell - 1, h(x_k))$ $= \sigma_0^{p_{0,h(x_k)}} \sigma_1^{p_{1,h(x_k)}} \cdots \sigma_{\ell-1}^{p_{\ell-1,h(x_k)}}$.

For the second partial encoding we encode the *complement* of each $enc(j, h(x_k))$ which is the unary representation of $\bar{p}_{j,h(x_k)} = R^{1/\ell} - (\lfloor h(x_k)/R^{j/\ell} \rfloor \bmod R^{1/\ell})$ using σ_j, and is denoted by $\overline{enc}(j, h(x_k))$. The second partial encoding of $h(x_k)$, which we also call a *complement encoding* of $h(x_k)$, is $\overline{enc_\ell}(h(x_k)) = \overline{enc}(0, h(x_k))\overline{enc}(1, h(x_k)) \cdots \overline{enc}(\ell - 1, h(x_k)) = \sigma_0^{\bar{p}_{0,h(x_k)}} \sigma_1^{\bar{p}_{1,h(x_k)}} \cdots \sigma_{\ell-1}^{\bar{p}_{\ell-1,h(x_k)}}$.

The full encoding of $h(x_k)$ is the concatenation of $\overline{enc_\ell}(h(x_k))$ and $enc_\ell(h(x_k))$ which we denote by $ENC_\ell(h(x_k))$. Finally, the string S is set to be $ENC_\ell(h(x_1))ENC_\ell(h(x_2)) \cdots ENC_\ell(h(x_n))$. The size of S is clearly $N = O(\ell \cdot R^{\frac{1}{\ell}} n)$. We denote the substring of S starting at the location of the beginning of $enc_\ell(h(x_i))$ and ending at the location of the end of $\overline{enc_\ell}(h(x_j))$ by $S_{i,j}$.

Consider a Parikh vector v_k obtained from x_k and h where the rth element has a count of $\bar{p}_{r,h(x_k)} + R^{1/\ell} \cdot (k - 1)$. We say that v_k *represents* x_k. For a vector $w = (w_0, w_1, ..., w_m)$ we define $w^{>>1} = (0, w_0, w_1, ..., w_{m-1})$. We also define the *carry set* of v_k to be $V_k = \{v_k + R^{1/\ell}u - u^{>>1} \mid u = (u_0, u_1, ..., u_{\ell-2}, 0),\ u_i \in \{0, 1\}\ 0 \le i < \ell - 1\}$. It is easy to see that $|V_k| = 2^{\ell-1}$ and that V_k can be obtained from v_k in $O(\ell \cdot 2^{\ell-1})$ time. We call v_k the *base* of V_k. We have the following lemma regarding V_k:

▶ **Lemma 16.** *If there exists a pair x_i, x_j such that $x_k = x_j - x_i$ and $k = j - i$, then the Parikh vector of $S_{i,j}$ must be in V_k.*

Proof. Since h is linear we know that $h(x_k) = h(x_j) - h(x_i)$. This is equivalent to saying that $R + R^{\frac{\ell-1}{\ell}} - h(x_k) = R + R^{\frac{\ell-1}{\ell}} - [h(x_j) - h(x_i)] = (R + R^{\frac{\ell-1}{\ell}} - h(x_j)) + h(x_i)$. In $S_{i,j}$ we have the full encoding of all integers $x_{i+1}, ..., x_{j-1}$. There are exactly $k - 1$ integers between x_i and x_j. Therefore, each of them adds $R^{1/\ell}$ occurrences of each σ_r $(0 \le r \le l - 1)$ to $S_{i,j}$. In addition to the full encodings of these integers we have two more partial encodings: $enc_\ell(h(x_i))$ and $\overline{enc_\ell}(h(x_j))$. Notice that $enc_\ell(h(x_i))$ and $\overline{enc_\ell}(h(x_j))$ represent $h(x_i)$ and $R + R^{\frac{\ell-1}{\ell}} - h(x_j)$, respectively, in base $R^{1/\ell}$. If we look at the vector v_k (the base of V_k) after subtracting $(k - 1)R^{1/\ell}$ from the count of each character, we obtain the representation of $R + R^{\frac{\ell-1}{\ell}} - h(x_k)$ in base $R^{1/\ell}$, which intuitively implies that v_k is the Parikh vector that we are looking for. However, it is possible to generate a carry at each of the ℓ digits of the base $R^{1/\ell}$ during the addition of $(R + R^{\frac{\ell-1}{\ell}} - h(x_j)) + h(x_i)$. To handle these carries we consider all possible 2^ℓ carry scenarios and generate a vector for each of the $2^{\ell-1}$ scenarios. These carry scenarios are exactly represented by the vectors in V_k, as each vector u in the definition of V_k specifies the indices in which we have a carry. Hence, the Parikh vector of $S_{i,j}$ must be one of the vectors in V_k. ◀

Thus, we preprocess S with an algorithm for histogram reporting, and then query the resulting data structure with all the vectors in V_k, whose base v_k represents some x_k, in an attempt to decide if x_k is part of a solution to DiffConv3SUM. The reported locations are classified into two types:

Candidates: Locations where the histogram match begins and ends exactly between the complement and regular encodings of two input integers. All these locations correspond to x_i and x_j such that for the particular $h(x_k)$ for which the query was constructed, we have $h(x_k) = h(x_j) - h(x_i)$ and also $k = j - i$.

Encoding Errors: All matches that are not candidates.

While encoding errors clearly do not provide a solution for DiffConv3SUM on A, candidates may also not be suitable for a solution since the function h introduces false-positives. The following lemma bounds the total expected number of false-positives (both from false-positive candidates and encoding errors) that can be reported by a single query vector (and the vectors in the carry set that it serves as it base).

▶ **Lemma 17.** *The expected number of false positives that are reported when considering all vectors in V_k (whose base represents x_k) as queries is $O(2^{\ell-1}N/R^{1-\frac{1}{\ell}})$.*

Proof. We focus on $v \in V_k$ that is queried when considering x_k. This vector v implies the value of m which is the length of substrings of S that can have v as their Parikh vector. Clearly, there are at most N such substrings. We focus on the substring from location α to location $\alpha + m - 1$ in S. Due to our encoding scheme, this substring contains a (possibly empty) suffix of $ENC_\ell(h(x_i))$, for some x_i, followed by $k-1$ full encodings of some integers from A, and then a (possibly empty) prefix of $ENC_\ell(h(x_j))$, for some integers x_i and x_j. The only way in which we may falsely report location α as a match is if for each $\sigma \in \Sigma$ the number of σ characters in the substring of S, denoted by $f(\sigma, \alpha, m)$, is equal to the count of σ in v, denoted by v_σ. For a given σ, since the substring contains $k-1$ complete encodings, we can consider $v_\sigma - (k-1)R^{1/\ell}$ which is a function of $\bar{p}_{r,h(x_k)}$, compared to $f(\sigma, \alpha, m) - (k-1)R^{1/\ell}$. Now, since $\bar{p}_{r,h(x_k)}$ is uniformly random (due to h) in the range $[R^{1/\ell}]$, the probability that they are equal is $R^{-1/\ell}$. This is true for every character σ on its own, but when considering all of the ℓ characters, once we set the count for the first $\ell - 1$ characters the count for the last character completely depends on the other counts. Therefore, the probability that the comparison passes for all of the characters only depends on the first $\ell - 1$ characters, and is $1/R^{1-1/\ell}$. By linearity of expectation over all possible locations in S and all $2^{\ell-1}$ vectors in V_k, the expected number of false positives is $O(2^{\ell-1}N/R^{1-\frac{1}{\ell}})$. ◀

7.2 Hardness of Histogram Reporting

Utilizing the reduction we have described in the previous section, that transforms an ordered set A to a string S, we can prove the following CLB.

▶ **Theorem 13** (restated). *Assume the 3SUM conjecture holds. The histogram reporting problem for an N-length string and constant alphabet size $\ell \geq 2$ cannot be solved using $O(N^{2-\frac{2\gamma}{\ell+\gamma}-\Omega(1)})$ preprocessing time, $O(N^{1-\frac{\gamma}{\ell+\gamma}-\Omega(1)})$ query time and $O(N^{\frac{\gamma\ell}{\ell+\gamma}-\frac{2\gamma}{\ell+\gamma}-\Omega(1)})$ reporting time per item, for any $0 < \gamma < \ell$.*

Proof. We follow the reduction in Section 7.1. For an instance of the histogram reporting problem on a string of length N denote the preprocessing time by $O(N^\alpha)$, the query time by $O(N^\beta)$ and the reporting time per item by $O(N^\delta)$. The total expected running time used by our reduction to solve DiffConv3SUM is $O(N^\alpha) + n \cdot O(N^\beta) + E_{fp} \cdot O(N^\delta)$, where E_{fp} is the expected total number of false positives. This running time must be $\Omega(n^{2-\Omega(1)})$, unless 3SUM conjecture is false.

Since $N = O(\ell \cdot R^{\frac{1}{\ell}}n)$ and $E_{fp} = O(n2^\ell N/R^{1-\frac{1}{\ell}})$, then either $(\ell \cdot R^{\frac{1}{\ell}}n)^\alpha = \Omega(n^{2-o(1)})$, $(\ell \cdot R^{\frac{1}{\ell}}n)^\beta = \Omega(n^{1-o(1)})$, or $n2^\ell(\ell \cdot R^{\frac{1}{\ell}}n)/R^{1-\frac{1}{\ell}} \cdot (\ell \cdot R^{\frac{1}{\ell}}n)^\delta = \Omega(n^{2-o(1)})$. Set R to be n^γ. By straightforward calculations following our choice of R we get that $\alpha = 2 - \frac{2\gamma}{\ell+\gamma} - \Omega(1)$, $\beta = 1 - \frac{\gamma}{\ell+\gamma} - \Omega(1)$, and $\delta = \frac{\gamma\ell}{\ell+\gamma} - \frac{2\gamma}{\ell+\gamma} - \Omega(1)$. ◀

7.3 From Reporting to Decision: Hardness of Histogram Indexing

We make use of Theorem 13 to obtain a CLB on the decision variant of the problem. Amir et al. [5] proved similar lower bounds based on a stronger 3SUM conjecture. Our proof here shows that this stronger assumption is not needed and that the common 3SUM conjecture suffices. The idea of the proof is to make the expected number of false-positives small by a suitable choice of R.

▶ **Lemma 18.** *Assume the 3SUM conjecture holds. The histogram indexing problem for a string of length N and constant alphabet size $\ell \geq 3$ cannot be solved with $O(N^{2-\frac{2}{\ell-1}-\Omega(1)})$ preprocessing time and $O(N^{1-\frac{1}{\ell-1}-\Omega(1)})$ query time.*

Proof. We follow the reduction in Section 7.1. In order to use histogram indexing we will reduce the probability of a false positive for any query to be less than 1/2. From Lemma 17 we know that the expected number of false positives due to query is at most $O(\frac{2^{\ell-1}(\ell R^{\frac{1}{\ell}} n)}{R^{1-\frac{1}{\ell}}})$. By setting R to be $c_1 n^{\frac{\ell}{\ell-2}}$ for sufficiently large constant c_1 the number of false positives is strictly smaller than 1/2, which implies immediately that the probability of a false positive is strictly smaller than 1/2. Therefore, if we were to solve histogram indexing instead of histogram reporting on the same input as in Theorem 13, the probability of a false positive is less than 1/2. We can make this probability smaller by repeating the process $O(\log n)$ times, each time using a different hash function h. This way, the probability that all of the queries that are due to a specific x_k return false positives is less than $1/poly(n)$. If a given x_k passes all of the query processes (that is, a positive answer is received by each one of them), then we can verify that there is indeed a match with this x_k in $O(n)$ time, which will add a negligible cost to the expected running time in the case it is indeed a false positive. Thus, the total expected running time of this procedure is $O(\log n(P(N,\ell)+nQ(N,\ell)))$, where $P(N,\ell)$ is the preprocessing time (for input string of length N and alphabet size ℓ) and $Q(N,\ell)$ is the query time (for the same parameters). Therefore, unless the 3SUM conjecture is false, there is no solution for histogram indexing such that $P(N,\ell) = O(n^{2-\Omega(1)})$ and $Q(N,\ell) = O(n^{1-\Omega(1)})$. If we plug-in the value of R we have chosen and follow the calculations in the proof of Theorem 13 (with $\gamma = \frac{\ell}{\ell-2}$), then we obtain that there is no solution for the histogram indexing problem with $P(N,\ell) = O(N^{2-\frac{2}{\ell-1}-\Omega(1)})$ and $Q(N,\ell) = O(N^{1-\frac{1}{\ell-1}-\Omega(1)})$. ◀

We generalize this CLB by presenting a full-tradeoff between preprocessing and query time. The proof will appear in the full version of this paper. The idea of the proof is to artificially split the encoded string S to smaller parts, so we can have many false positives in S, but the probability for a false positive in each part will be small.

▶ **Theorem 14** (restated). *Assume the 3SUM conjecture holds. The histogram indexing problem for a string of length N and constant alphabet size $\ell \geq 3$ cannot be solved with $O(N^{2-\frac{2(1-\alpha)}{\ell-1-\alpha}-\Omega(1)})$ preprocessing time and $O(N^{1-\frac{1+\alpha(\ell-3)}{\ell-1-\alpha}-\Omega(1)})$ query time, for any $0 \leq \alpha \leq 1$.*

References

1 Amir Abboud and Kevin Lewi. Exact weight subgraphs and the k-sum conjecture. In *Int'l 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 Amihood 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.

6 Ilya Baran, Erik D. Demaine, and Mihai Patrascu. Subquadratic algorithms for 3SUM. In *Workshop on Algorithms and Data Structures, WADS 2005*, pages 409–421, 2005.

7 David Bremner, Timothy M. Chan, Erik D. Demaine, Jeff Erickson, Ferran Hurtado, John Iacono, Stefan Langerman, Mihai Patrascu, and Perouz Taslakian. Necklaces, convolutions, and X+Y. *Algorithmica*, 69(2):294–314, 2014.

8 Peter Burcsi, Ferdinando Cicalese, Gabriele Fici, and Zsuzsanna Lipták. Algorithms for jumbled pattern matching in strings. *Int. J. Found. Comput. Sci.*, 23(2):357–374, 2012.

9 Timothy Chan and Moshe Lewenstein. Clustered integer 3SUM via additive combinatorics. In *Symposium on Theory of Computing, STOC 2015*, pages 31–40, 2015.

10 Ferdinando Cicalese, Gabriele Fici, and Zsuzsanna Lipták. Searching for jumbled patterns in strings. In *Prague Stringology Conference*, pages 105–117, 2009.

11 Richard Cole and Ramesh Hariharan. Verifying candidate matches in sparse and wildcard matching. In *Symposium on Theory of Computing, STOC 2002*, pages 592–601, 2002.

12 Martin Dietzfelbinger. Universal hashing and k-wise independent random variables via integer arithmetic without primes. In *Symposium on Theoretical Aspects of Computer Science, STACS 1996*, pages 569–580, 1996.

13 Anka Gajentaan and Mark H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Comput. Geom.*, 5:165–185, 1995.

14 François Le Gall. Powers of tensors and fast matrix multiplication. In *International Symposium on Symbolic and Algebraic Computation, ISSAC 2014*, pages 296–303, 2014.

15 Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price. Nearly optimal sparse fourier transform. In *Symposium on Theory of Computing Conference, STOC 2012*, pages 563–578, 2012.

16 Danny Hermelin, Gad M. Landau, Yuri Rabinovich, and Oren Weimann. Binary jumbled pattern matching via all-pairs shortest paths. *CoRR*, abs/1401.2065, 2014.

17 Allan Grønlund Jørgensen and Seth Pettie. Threesomes, degenerates, and love triangles. In *Foundations of Computer Science, FOCS 2014*, pages 621–630, 2014.

18 Tomasz Kociumaka, Jakub Radoszewski, and Wojciech Rytter. Efficient indexes for jumbled pattern matching with constant-sized alphabet. In *ESA*, pages 625–636, 2013.

19 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.

20 Tanaeem M. Moosa and M. Sohel Rahman. Indexing permutations for binary strings. *Inf. Process. Lett.*, 110(18-19):795–798, 2010.

21 Tanaeem M. Moosa and M. Sohel Rahman. Sub-quadratic time and linear space data structures for permutation matching in binary strings. *J. Discrete Algorithms*, 10:5–9, 2012.

22 S. Muthukrishnan. New results and open problems related to non-standard stringology. In *CPM 1995*, pages 298–317, 1995.

23 Mihai Patrascu. Towards polynomial lower bounds for dynamic problems. In *Symposium on Theory of Computing Conference, STOC 2010*, pages 603–610, 2010.

24 Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *Symposium on Theory of Computing, STOC 2014*, pages 664–673, 2014.

25 Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *Symposium on Theory of Computing Conference, STOC 2012*, pages 887–898, 2012.

Incremental Exact Min-Cut in Poly-logarithmic Amortized Update Time*

Gramoz Goranci[1], Monika Henzinger[†2], and Mikkel Thorup[‡3]

1 **University of Vienna, Faculty of Computer Science, Vienna, Austria**
gramoz.goranci@univie.ac.at
2 **University of Vienna, Faculty of Computer Science, Vienna, Austria**
monika.henzinger@univie.ac.at
3 **Faculty of Computer Science, University of Copenhagen, Copenhagen, Denmark**
mikkel2thorup@gmail.com

Abstract

We present a deterministic incremental algorithm for *exactly* maintaining the size of a minimum cut with $\widetilde{O}(1)$ amortized time per edge insertion and $O(1)$ query time. This result partially answers an open question posed by Thorup [Combinatorica 2007]. It also stays in sharp contrast to a polynomial conditional lower-bound for the fully-dynamic weighted minimum cut problem. Our algorithm is obtained by combining a recent sparsification technique of Kawarabayashi and Thorup [STOC 2015] and an exact incremental algorithm of Henzinger [J. of Algorithm 1997].

We also study space-efficient incremental algorithms for the minimum cut problem. Concretely, we show that there exists an $O(n \log n/\varepsilon^2)$ space Monte-Carlo algorithm that can process a stream of edge insertions starting from an empty graph, and with high probability, the algorithm maintains a $(1+\varepsilon)$-approximation to the minimum cut. The algorithm has $\widetilde{O}(1)$ amortized update-time and constant query-time.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Dynamic Graph Algorithms, Minimum Cut, Edge Connectivity

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.46

1 Introduction

Computing a minimum cut of a graph is a fundamental algorithmic graph problem. While most of the focus has been on designing static efficient algorithms for finding a minimum cut, the dynamic maintenance of a minimum cut has also attracted increasing attention over the last two decades. The motivation for studying the dynamic setting is apparent, as important networks such as social or road network undergo constant and rapid changes.

Given an initial graph G, the goal of a dynamic graph algorithm is to build a data-structure that maintains G and supports update and query operations. Depending on the types of update operations we allow, dynamic algorithms are classified into three main

* This work was done in part while M. Henzinger and M. Thorup were visiting the Simons Institute for the Theory of Computing.
† The research leading to these results has received funding from the European Research Council under the European Union's 7th Framework Programme (FP/2007-2013) / ERC Grant Agreement no. 340506 for M. Henzinger.
‡ M. Thorup's research is partly supported by Advanced Grant DFF-0602-02499B from the Danish Council for Independent Research under the Sapere Aude research career programme.

categories: (i) *fully dynamic*, if update operations consist of both edge insertions and deletions, (ii) *incremental*, if update operations consist of edge insertions only and (iii) *decremental*, if update operations consist of edge deletions only. In this paper, we study incremental algorithms for maintaining the size of a minimum cut of an unweighted, undirected graph (denoted by $\lambda(G) = \lambda$) supporting the following operations:

- Insert(u, v): insert the edge (u, v) in G.
- QuerySize: return the size of a minimum cut of the current G.

For any $\alpha \geq 1$, we say that an algorithm is an α-approximation of λ if QuerySize returns a positive number k such that $\lambda \leq k \leq \alpha \cdot \lambda$. Our problem is characterized by two time measures; *query time*, which denotes the time needed to answer each query and *total update time*, which denotes the time needed to process *all* edge insertions. We say that an algorithm has an $O(t(n))$ amortized update time if it takes $O(m(t(n)))$ total update time for m edge insertions starting from an empty graph. We use $\widetilde{O}(\cdot)$ to hide poly-logarithmic factors.

Related Work

For over a decade, the best known static and deterministic algorithm for computing a minimum cut was due to Gabow [10] which runs in $O(m + \lambda^2 \log n)$ time. Recently, Kawarabayashi and Thorup [19] devised a $\widetilde{O}(m)$ time algorithm which applies only to simple, unweighted and undirected graphs. Randomized Monte Carlo algorithms in the context of static minimum cut were initiated by Karger [17]. The best known randomized algorithm is due to Karger [18] and runs in $O(m \log^3 n)$ time.

Karger [16] was the first to study the dynamic maintenance of a minimum cut in its full generality. He devised a fully dynamic, albeit randomized, algorithm for maintaining a $\sqrt{1 + 2/\varepsilon}$-approximation of the minimum cut in $\widetilde{O}(n^{1/2+\varepsilon})$ expected amortized time per edge operation. In the incremental setting, he showed that the update time for the same approximation ratio can be further improved to $\widetilde{O}(n^{\varepsilon})$. Thorup and Karger [28] improved upon the above guarantees by achieving an approximation factor of $\sqrt{2 + o(1)}$ and an $\widetilde{O}(1)$ expected amortized time per edge operation.

Henzinger [14] obtained the following guarantees for the incremental minimum cut; for any $\varepsilon \in (0, 1]$, (i) an $O(1/\varepsilon^2)$ amortized update-time for a $(2 + \varepsilon)$-approximation, (ii) an $O(\log^3 n/\varepsilon^2)$ expected amortized update-time for a $(1 + \varepsilon)$-approximation and (iii) an $O(\lambda \log n)$ amortized update-time for the exact minimum cut.

For minimum cut up to some poly-logarithmic size, Thorup [27] gave a fully dynamic Monte-Carlo algorithm for maintaining exact minimum cut in $\widetilde{O}(\sqrt{n})$ time per edge operation. He also showed how to obtain an $1 + o(1)$-approximation of an arbitrary sized minimum cut with the same time bounds. In comparison to previous results, it is worth pointing out that his work achieves *worst-case* update times.

Łącki and Sankwoski [21] studied the dynamic maintenance of the exact size of the minimum cut in planar graphs with arbitrary edge weights. They obtained a fully dynamic algorithm with $\widetilde{O}(n^{5/6})$ query and update time.

There has been a growing interest in proving conditional lower bounds for dynamic problems in the last few years [1, 13]. A recent result of Nanongkai and Saranurak [24] shows the following conditional lower-bound for the *exact weighted* minimum cut assuming the Online Matrix-Vector Multiplication conjecture: for any $\varepsilon > 0$, there are no fully-dynamic algorithms with polynomial-time preprocessing that can simultaneously achieve $O(n^{1-\varepsilon})$ update-time and $O(n^{2-\varepsilon})$ query-time.

Results and Technical Overview

We present two new incremental algorithms concerning the maintenance of the size of a minimum cut. Both algorithms apply to undirected, unweighted simple graphs.

Our first and main result, presented in Section 4, shows that there is a deterministic incremental algorithm for *exactly* maintaining the size of a minimum cut with $\widetilde{O}(1)$ amortized time per operation and $O(1)$ query time. This result allows us to partially answer in the affirmative a question regarding efficient dynamic algorithms for exact minimum cut posed by Thorup [27]. Meanwhile, it also stays in sharp contrast to the recent polynomial conditional lower-bound for the fully-dynamic weighted minimum cut problem [24].

We obtain our result by heavily relying on a recent sparsification technique developed in the context of static minimum cut. Specifically, for some given simple graph G, Kawarabayashi and Thorup [19] designed an $\widetilde{O}(m)$ procedure that contracts vertex sets of G and produces a multigraph H with considerably less vertices and edges while preserving some family of cuts of size up to $3/2\lambda(G)$. Motivated by the properties of H, we crucially observe that it is "safe" to escape from G and work entirely with graph H as long as the sequence of newly inserted edges have not increased the size of a minimum cut in H by more than $3/2\lambda(G)$. If the latter occurs, we then recompute a new multigraph H for the current graph G. Since $\lambda(G) \leq n$, we note the number of such re-computations can be at most $O(\log n)$. For maintaining the minimum-cut of H, we appeal to the exact incremental algorithm due to Henzinger [14]. Though the combination of this two algorithms might seem immediate at first sight, we remark that it is not alone sufficient for achieving the claimed bounds. Our main contribution is to overcome some technical obstacles and formally argue that such combination indeed leads to our desirable guarantees.

Motivated by the recent work on *space-efficient* dynamic algorithms [5, 12], we next study the efficient maintenance of the size of a minimum cut using only $\widetilde{O}(n)$ space. Concretely, we present a $O(n \log n/\varepsilon^2)$ space Monte-Carlo algorithms that can process a stream of edge insertions starting from an empty graph, and with high probability, the algorithm maintains an $(1+\varepsilon)$-approximation to the minimum cut in $O(\alpha(n)\log^3 n/\varepsilon^2)$ amortized update-time and constant query-time. Note that none of the existing streaming algorithms for $(1+\epsilon)$-approximate minimum cut [2, 20, 3] achieve these update and query times.

2 Preliminary

Let $G = (V, E)$ be an undirected, unweighted multigraph with no self-loops. Two vertices x and y are *k-edge connected* if there exist k edge-disjoint paths connecting x and y. A graph G is *k-connected* if every pair of vertices is k-edge connected. The *local edge connectivity* $\lambda(G, x, y)$ of vertices x and y is the largest k such that x and y are k-edge connected in G. The *edge connectivity* $\lambda(G)$ of G is the largest k such that G is k-edge connected.

For a subset $S \subseteq V$, the *edge cut* $E(S, V \setminus S)$ is a set of edges that have one endpoint in S and the other in $V \setminus S$. If S is a singleton, we refer to such cut as *trivial* cut. Two vertices x and y are *separated* from $E(S, V \setminus S)$ if they do not belong to the same connected component induced by the edge cut. A *minimum edge cut* of x and y is a cut of minimum size among all cuts separating x and y. A *global minimum cut* $\lambda(G)$ for G is the minimum edge cut over all pairs of vertices. By Menger's Theorem [22], (a) the size of the minimum edge cut separating x and y is $\lambda(G, x, y)$, and (b) the size of the global minimum cut is equal to $\lambda(G)$.

Let n, m_0 and m_1 be the number of vertices, initial edges and inserted edges, respectively. The total number of edges m is the sum of the initial and inserted edges. Moreover, let

λ and δ denote the size of the global minimum cut and the minimum degree in the final graph, respectively. Note that the minimum degree is always an upper bound on the edge connectivity, i.e., $\lambda \leq \delta$ and $m = m_0 + m_1 = \Omega(\delta n)$.

A subset $U \subseteq V$ is *contracted* if all vertices in U are identified with some element from U and all edges between them are discarded. Note that this may not correspond to edge contractions, since we do not know whether U is connected. For $G = (V, E)$ and a collection of vertex sets, let $H = (V_H, E_H)$ denote the graph obtained by contracting such vertex sets. Such contractions are associated with a mapping $h : V \to V_H$. For an edge subset $N \subseteq E$, let $N_h = \{(h(a), h(b)) : (a, b) \in N\} \subseteq E_H$ be its corresponding edge subset induced by h.

3 Sparse certificates

In this section we review a useful sparsification tool, introduced by Nagamochi and Ibaraki [23].

▶ **Definition 1** ([4]). A *sparse k-connectivity certificate*, or simply a *k-certificate*, for an unweighted graph G with n vertices is a subgraph G' of G such that

1. G' consists of at most $k(n-1)$ edges, and
2. G' contains all edges crossing cuts of size at most k.

Given an undirected graph $G = (V, E)$, a *maximal spanning forest decomposition (msfd)* of order k is a decomposition of G into k edge-disjoint spanning forests F_i, $1 \leq i \leq k$, such that F_i is a maximal spanning forest of $G \setminus (F_1 \cup F_2 \ldots \cup F_{i-1})$. If we let $G' = (V, \bigcup_{i \leq k} F_i)$, then G' is a k-certificate. An msfd that fulfills the following additional properties is called a DA-msfd of order k: For a multigraph G, (1) for all $1 \leq i \leq k$, if x and y are connected in F_i, then they are i-edge connected in G; (2) G is k-edge connected iff G' is k-edge connected; (3) for any $1 \leq i \leq k$ and $x, y \in V$, $\lambda(\bigcup_{j \leq i} F_j, x, y) \geq \min(\lambda(G, x, y), i)$. As G' is a subgraph of G, $\lambda(G') \leq \lambda(G)$. This implies that $\lambda(G') = \min(k, \lambda(G))$. Nagamochi and Ibaraki [23] presented a $O(m + n)$ time algorithm to construct a DA-msfd, of order k.

4 Incremental Exact Minimum Cut

In this section we present a deterministic incremental algorithm that exactly maintains $\lambda(G)$. The algorithm has an $\widetilde{O}(1)$ update-time, an $O(1)$ query time and it applies to any undirected, unweighted, simple graph $G = (V, E)$. The result is obtained by carefully combining a recent result of Kawarabayashi and Thorup [19] on static min-cut and the incremental exact min-cut algorithm of Henzinger [14]. We start by describing the maintenance of non-trivial cuts, that is, cuts with at least two vertices on both sides.

Maintaining non-trivial cuts

Kawarabayashi and Thorup [19] devised a near-linear time algorithm that contracts vertex sets of a simple input graph G and produces a sparse multi-graph preserving all non-trivial minimum cuts of G. In the following theorem, we state a slightly generalized version of this algorithm.

▶ **Theorem 2** (KT-Sparsifier [19]). *Given an undirected, unweighted graph G with n vertices, m edges, and min-cut λ, in $\widetilde{O}(m)$ time, we can contract vertex sets and produce a multigraph H which consists of only $m_H = \widetilde{O}(m/\lambda)$ edges and $n_H = \widetilde{O}(n/\lambda)$ vertices, and which preserves all non-trivial minimum cuts along with the non-trivial cuts of size up to $3/2\lambda$ in G.*

As far as non-trivial cuts are concerned, the above theorem implies that it is safe to abandon G and work on H as long as the sequence of newly inserted edges satisfies $\lambda_H \leq 3/2\lambda$. To incrementally maintain the correct λ_H, we apply Henzinger's algorithm [14] on top of H. The basic idea to verify the correctness of the solution is to compute and store all min-cuts. Clearly, a solution is correct as long as an edge insertion does not increase the size of all min-cuts. If all min-cuts have increased, a new solution is computed using information about the previous solution. We next show how to do this efficiently.

To store all minimum edge cuts we use the *cactus tree* representation by Dinitz, Karzanov and Lomonosov [7]. A cactus tree of a graph $G = (V, E)$ is a weighted graph $G_c = (V_c, E_c)$ defined as follows: There is a mapping $\phi : V \to V_c$ such that:

1. Every node in V maps to exactly one node in V_c and every node in V_c corresponds to a (possibly empty) subset of V.
2. $\phi(x) = \phi(y)$ iff x and y are $(\lambda(G) + 1)$-edge connected.
3. Every minimum cut in G_c corresponds to a min-cut in G, and every min-cut in G corresponds to at least one min-cut in G_c.
4. If λ is odd, every edge of E_c has weight λ and G_c is a tree. If λ is even, G_c consists of paths and simple cycles sharing at most one vertex, where edges that belong to a cycle have weight $\lambda/2$ while those not belonging to a cycle have weight λ.

Dinitz and Westbrook [8] showed that given a cactus tree, we can use the data structures from [11, 25] to maintain the cactus tree for minimum cut size λ under u insertions, reporting when the minimum cut size increases to $\lambda + 1$ in $O(u + n)$ total time.

To quickly compute and update the cactus tree representation of a given multigraph G, we use an algorithm due to Gabow [9]. The algorithm computes first a subgraph of G, called a *complete λ-intersection* or $I(G, \lambda)$, with at most λn edges, and uses $I(G, \lambda)$ to compute the cactus tree. Given some initial graph with m_0 edges, the algorithm computes $I(G, \lambda)$ and the cactus tree in $\widetilde{O}(m_0 + \lambda^2 n)$ time. Moreover, given $I(G, \lambda)$ and a sequence of edge insertions that increase the minimum cut by 1, the new $I(G, \lambda)$ and the new cactus tree can be computed in $\widetilde{O}(m')$, where m' is the number of edges in the current graph (this corresponds to one execution of Round Robin subroutine [10]).

Maintaining trivial cuts

We remark that the multigraph H from Theorem 2 preserves only non-trivial cuts of G. If $\lambda = \delta$, then we also need a way to keep track of a trivial minimum cut. We achieve this by maintaining a minimum heap $\mathcal{H}_G$ on the vertices, where each vertex is stored with its degree. If an edge insertion is performed, the values of the edge endpoints are updated accordingly in the heap. It is well known that constructing $\mathcal{H}_G$ takes $O(n)$ time. The supported operations $\textsc{Min}(\mathcal{H}_G)$ and $\textsc{UpdateEndpoints}(\mathcal{H}_G, e)$ can be implemented in $O(1)$ and $O(\log n)$ time, respectively (see [6]).

This leads to the following Algorithm 1.

Correctness

Let G be some current graph throughout the execution of the algorithm and let H be the corresponding multigraph maintained by the algorithm. Recall that H preserves some family of cuts from G. We say that H is *correct* if and only if there exists a minimum cut from G that is contained in the union of (a) all trivial cuts of G and (b) all cuts in H. Note that we consider H to be correct even in the `Special Step` (i.e., when $\lambda_H > 3/2\lambda^*$), where H is not

Algorithm 1 INCREMENTAL EXACT MINIMUM CUT

```
1: Compute the size $\lambda_0$ of the min-cut of $G$ and set $\lambda^* = \lambda_0$.
   Build a heap $\mathcal{H}_G$ on the vertices, where each vertex stores its degree as a key.
   Compute a multigraph $H$ by running KT-SPARSIFIER on $G$ and a mapping $h : V \to V_H$.
   Compute the size $\lambda_H$ of the min-cut of $H$, a DA-msfd $F_1, \ldots, F_m$ of order $m$ of $H$,
   $I(H, \lambda_H)$, and a cactus-tree of $\bigcup_{i \le \lambda_H + 1} F_i$.
2: Set $N_h = \emptyset$.
   while there is at least one minimum cut of size $\lambda_H$ do
      Receive the next operation.
      if it is a query then return $\min\{\lambda_H, \text{MIN}(\mathcal{H}_G)\}$
      else it is the insertion of an edge $(u, v)$, then
      update the cactus tree according to the insertion of the new edge $(h(u), h(v))$,
      add the edge $(h(u), h(v))$ to $N_h$ and update the degrees of $u$ and $v$ in $\mathcal{H}_G$.
      endif
   endwhile
   Set $\lambda_H = \lambda_H + 1$.
3: if $\min\{\lambda_H, \text{MIN}(\mathcal{H}_G)\} > 3/2\lambda^*$ then
      // Full Rebuild Step
      Compute $\lambda(G)$ and set $\lambda^* = \lambda(G)$.
      Compute a multigraph $H$ by running KT-SPARSIFIER on the current graph $G$.
      Update $\lambda_H$ to be the min-cut of $H$, compute a DA-msfd $F_1, \ldots, F_m$ of order $m$ of $H$,
      and then $I(H, \lambda_H)$ and a cactus tree of $\bigcup_{i \le \lambda_H + 1} F_i$.
   else if $\lambda_H \le 3/2\lambda^*$ then
         // Partial Rebuild Step
         Compute a DA-msfd $F_1, \ldots, F_m$ of order $m$ of $\bigcup_{i \le \lambda_H + 1} F_i \cup N_h$ and
         call the resulting forests $F_1, \ldots, F_m$.
         Let $H' = (V_H, E')$ be a graph with $E' = I(H, \lambda_H - 1) \cup \bigcup_{i \le \lambda_H + 1} F_i$.
         Compute $I(H', \lambda_H)$ and a cactus tree of $H'$.
      else // Special Step
         while $\text{MIN}(\mathcal{H}_G) \le 3/2\lambda^*$ do
            if the next operation is a query then return $\text{MIN}(\mathcal{H}_G)$
            else update the degrees of the edge endpoints in $\mathcal{H}_G$.
            endif
         endwhile
         Goto 3.
      endif
   endif
   Goto 2.
```

updated anymore since we are certain that the smallest trivial cut is smaller than any cut in H.

To prove the correctness of the algorithm we will show that (1) it correctly maintains a trivial min-cut at any time, (2) H is correct as long as $\min\{\textsc{Min}(\mathcal{H}_G), \lambda_H\} \leq 3/2\lambda^*$ (and when this condition fails we rebuild H), and (3) as long as $\lambda_H \leq 3/2\lambda^*$, the algorithm correctly maintains all cuts of size up to $\lambda_H + 1$ of H.

Let N be the set of recently inserted edges in G that the algorithm maintains during the execution of the **while** loop in Step 2. Similarly, let N_h be the corresponding edge set in H.

▶ **Lemma 3.** *Let $H = (V_H, E_H)$ be a multigraph with minimum cut λ_H and let N_h be a set with $N_h \subseteq E_H$. Further, let $F_1, \ldots, F_m$ be a DA-msfd of order $m \geq \lambda_H + 1$ of $H \setminus N_h$, and let $H' = (V_H, E')$ be a graph with $E' = N_h \cup \bigcup_{i \leq \lambda_H + 1} F_i$. Then, a cut is a min-cut in H' iff it is a min-cut in H.*

Proof. We first show that every non-min cut in H is a non-min cut in H'. By contrapositive, we get that a min-cut in H' is a min-cut in H.

To this end, let $(S, V_H \setminus S)$ be a cut with $|E_H(S, V_H \setminus S)| \geq \lambda_H + 1$ in H. Define $E_H(S, V_H \setminus S) \cap N_h = S_{N_h}$ and $E_H(S, V_H \setminus S) \cap (E_H \setminus N_h) = S_{H \setminus N_h}$ such that $E_H(S, V_H \setminus S) = S_{N_h} \uplus S_{H \setminus N_h}$ and $|E_H(S, V_H \setminus S)| = |S_{N_h}| + |S_{H \setminus N_h}|$. Letting $F' = \bigcup_{i \leq \lambda_H + 1} F_i$, we similarly define edge sets S'_{N_h} and $S'_{F'}$ partitioning the edges $E'(S, V_H \setminus S)$ that cross the cut $(S, V_H \setminus S)$ in H'. First, observe that $|S_{N_h}| = |S'_{N_h}|$ since edges of N_h are always included in H'. In addition, by second property of Definition 1, we know that F' preserves all cuts of $H \setminus N_h$ up to size $\lambda_H + 1$. Thus, if $|S_{H \setminus N_h}| \leq \lambda_H + 1$, we get that $|E'(S, V_H \setminus S)| = |E_H(S, V_H \setminus S)| \geq \lambda_H + 1$. If $|S_{H \setminus N_h}| > \lambda_H + 1$, then F' must contain at least $\lambda_H + 1$ edges crossing such cut and thus $|S'_{F'}| \geq \lambda_H + 1$. The latter implies that $|E'(S, V_H \setminus S)| \geq \lambda_H + 1$. But H' being a subgraph of H implies that $\lambda(H') \leq \lambda_H$, thus $(S, V_H \setminus S)$ cannot be a min-cut in H'.

For other direction, let $(S, V_H \setminus S)$ be a min-cut in H. Since H' is a subgraph of H, we know that $|E'(S, V_H \setminus S)| \leq \lambda_H$. Therefore, showing that $|E'(S, V_H \setminus S)| \geq \lambda_H$ implies that $(S, V_H \setminus S)$ is also a min cut in H'. Fix x, y and consider a min-cut $(D, V_H \setminus D)$ of size $\lambda(H', x, y)$ separating x and y. Using the above notation and considering the cut $(D, V_H \setminus D)$ in H, we know that $|E_H(D, V_H \setminus D)| = |D_{N_h}| + |D_{H \setminus N_h}| \geq \lambda_H$. We first note that $|D_{N_h}| = |D'_{N_h}|$ since edges of N_h are always included in H'. Then, similarly as above, by second property of Definition 1 we know that if $|D_{H \setminus N_h}| \leq \lambda_H + 1$, then $|E'(D, V_H \setminus D)| = |E_H(D, V_H \setminus D)| \geq \lambda_H$. If $|D_{H \setminus N_H}| > \lambda_H + 1$, then F' must contain at least $\lambda_H + 1$ edges crossing such cut and thus $|E'(D, V_H \setminus D)| \geq \lambda_H + 1$. Combining both bounds we obtain that $\lambda(H', x, y) = |E'(D, V_H \setminus D)| \geq \lambda_H$. Since the later is valid for any x and y, we get that $\lambda(H') \geq \lambda_H$ must hold and in particular, $|E'(S, V_H \setminus S)| \geq \lambda_H$. ◀

▶ **Lemma 4.** *The algorithm correctly maintains a trivial min-cut in G.*

Proof. This follows directly from the min-heap property of $\mathcal{H}_G$. ◀

To simplify our notation, in the following we will refer to Step 1 as a `Full Rebuild Step` (namely the initial `Full Rebuild Step`).

▶ **Lemma 5.** *For some current graph G, let H be the multigraph obtained from G and assume that $\lambda_H \leq 3/2\lambda^*$, where λ^* denotes the value of min-cut at the last* `Full Rebuild Step`. *Then the algorithm correctly maintains $\lambda_H = \lambda(H)$.*

Proof. At the time of the last `Full Rebuild Step`, the algorithm applies KT-sparsifier on G, which yields a multigraph H that preserves all non-trivial min-cuts of G. The value of

λ_H is updated to $\lambda(H)$ and a DA-msfd and a cactus tree are constructed for H. The latter preserve all cuts of H of size up to $\lambda_H + 1$. Thus, the value of λ_H is correct at this step.

Now, suppose that the graph after the last `Full Rebuild Step` has undergone a sequence of edge insertions, which resulted in the current graph G. During these insertions, as long as $\lambda_H \leq 3/2\lambda^*$, a sequence of k `Partial Rebuild Steps` is executed, for some $k \geq 1$. Let $\lambda_H^{(i)}$ be the value of λ_H after the i-th execution of `Partial Rebuild Step`, where $1 \leq i \leq k$. Since, $\lambda_H^{(k)} = \lambda(H)$, it suffices to show that $\lambda_H^{(k)}$ is correct. We proceed by induction.

For the base case, we show that $\lambda_H^{(1)}$ is correct. First, using the fact that λ_H and the cactus tree are correct at the last `Full Rebuild Step` and that the incremental cactus tree algorithm correctly tell us when to increment λ_H, we conclude that incrementing the value of λ_H in Step 2 is valid. Thus, $\lambda_H^{(1)}$ is correct. Next, in a `Partial Rebuild Step`, the algorithm sparsifies the graph while preserving all cuts of size up to $\lambda_H^{(1)} + 1$ and producing a new cactus tree for the next insertions. The correctness of the sparsification follows from Lemma 3.

For the induction step, let us assume that $\lambda_H^{(k-1)}$ is correct. Then, similarly to the base case, the correctness of $\lambda_H^{(k-1)}$, the cactus tree from the $(k-1)$-th `Partial Rebuild Step` and the correctness of the incremental cactus tree algorithm give that incrementing the value of $\lambda_H^{(k-1)}$ in Step 2 is valid and yields a correct $\lambda_H^{(k)}$. ◀

Note that when $\lambda_H > 3/2\lambda^*$, the above lemma is not guaranteed to hold. However, we will show below that this is not necessary for the correctness of the algorithm. The fact that we do not need to update the cactus tree in this setting is crucial for achieving our time bound.

▶ **Lemma 6.** *If* $\min\{\textsc{Min}(\mathcal{H}_G), \lambda_H\} \leq 3/2\lambda^*$*, then H is correct.*

Proof. Let C' be any non-trivial cut in G that is not in H. Such a cut must have cardinality strictly greater than $3/2\lambda^*$ since otherwise it would be contained in H. We show that C' cannot be a minimum cut as long as $\min\{\textsc{Min}(\mathcal{H}_G), \lambda_H\} \leq 3/2\lambda^*$ holds. We distinguish two cases.

1. If $\lambda_H \leq 3/2\lambda^*$, then by Lemma 5 the algorithm maintains λ_H correctly. Since H is obtained from G by contracting vertex sets, there is a cut C in H, and thus in G, of value λ_H. It follows that C' cannot be a minimum cut of G since $|C'| > 3/2\lambda^* \geq \lambda_H = \lambda(H) \geq \lambda(G)$, where the last inequality follows from the fact that H is a contraction of G.
2. If $\textsc{Min}(\mathcal{H}_G) \leq 3/2\lambda^*$, then by Lemma 4 there is a cut of size $\textsc{Min}(\mathcal{H}_G) = \delta$ in G. Similarly, C' cannot be a minimum cut of G since $|C'| > 3/2\lambda^* \geq \delta \geq \lambda(G)$.

Appealing to the above cases, we conclude H is correct since a min-cut of G is either contained in H or it is a trivial cut of G. ◀

▶ **Lemma 7.** *The algorithm correctly maintains $\lambda(G)$, i.e., $\lambda(G) = \min\{\textsc{Min}(\mathcal{H}_G), \lambda_H\}$.*

Proof. Let G be some current graph. If $\min\{\textsc{Min}(\mathcal{H}_G), \lambda_H\} \leq 3/2\lambda^*$, then by Lemma 6, H is correct. Thus, if $\lambda_H \leq 3/2\lambda^*$, then Lemma 5 ensures that λ_H is also maintained correctly and, hence, $\min\{\textsc{Min}(\mathcal{H}_G), \lambda_H\} = \lambda(G)$. If, however, $\lambda_H > 3/2\lambda^*$ but $\min\{\textsc{Min}(\mathcal{H}_G), \lambda_H\} \leq 3/2\lambda^*$, then $\lambda_H > \min\{\textsc{Min}(\mathcal{H}_G), \lambda_H\}$ which implies that $\min\{\textsc{Min}(\mathcal{H}_G), \lambda_H\} = \textsc{Min}(\mathcal{H}_G)$. As the algorithm correctly maintains $\textsc{Min}(\mathcal{H}_G)$ at any time by Lemma 4, it follows that the algorithm maintains λ correctly in this case as well.

The only case that remains to consider is $\textsc{Min}(\mathcal{H}_G) > 3/2\lambda^*$ and $\lambda_H > 3/2\lambda^*$. But this implies that $\min\{\textsc{Min}(\mathcal{H}_G), \lambda_H\} > 3/2\lambda^*$, and the algorithm computes a H and $\lambda(G)$ from scratch and sets λ_H correctly. After this full rebuild $\lambda(G) = \min\{\textsc{Min}(\mathcal{H}_G), \lambda_H\}$ trivially holds. ◀

Running Time Analysis

▶ **Theorem 8.** *Let G be a simple graph with n nodes and m_0 edges. Then the total time for inserting m_1 edges and maintaining a minimum edge cut of G is $\widetilde{O}(m_0 + m_1)$. If we start with an empty graph, the amortized time per edge insertion is $\widetilde{O}(1)$. The size of the minimum cut can be answered in constant time.*

Proof. We first analyse Step 1. Building the heap $\mathcal{H}_G$ and computing λ_0 take $O(n)$ and $\widetilde{O}(m_0)$ time, respectively. The total running time for constructing H, $I(H, \lambda_H)$ and the cactus tree is dominated by $\widetilde{O}(m_0 + \lambda_0^2 \cdot (n/\lambda_0)) = \widetilde{O}(m_0)$. Thus, the total time for Step 1 is $\widetilde{O}(m_0)$.

Let $\lambda_H^0, \ldots, \lambda_H^f$ be the values that λ_H assumes in Step 2 during the execution of the algorithm in increasing order. We define Phase i to be all steps executed after Step 1 while $\lambda_H = \lambda_H^i$, excluding Full Rebuild Steps and Special Steps. Additionally, let $\lambda_0^*, \ldots, \lambda_{O(\log n)}^*$ be the values that λ^* assumes during the algorithm. We define *Superphase* j to consist of the j-th `Full Rebuild Step` along with all steps executed while $\min\{\textsc{Min}(\mathcal{H}_G), \lambda_H\} \leq 3/2\lambda_j^*$, where λ_j^* is the value of $\lambda(G)$ at the `Full Rebuild Step`. Note that a superphase consists of a sequence of phases and potentially a final `Special Step`. Moreover, the algorithm runs a phase if $\lambda_H \leq 3/2\lambda^*$.

We say that λ_H^i *belongs* to superphase j, if the i-th phase is executed during superphase j and $\lambda_H^i \leq 3/2\lambda_j^*$. We remark that the number of vertices in H changes only at the beginning of a superphase, and remains unchanged during its lifespan.

Let n_j denote the number of vertices in some superphase j. We bound this quantity as follows:

▶ **Fact 9.** *Let j be a superphase during the execution of the algorithm. Then, we have*

$$n_j = \widetilde{O}(n/\lambda_H^i), \text{ for all } \lambda_H^i \text{ belonging to superphase } j.$$

Proof. From Step 3 we know that $n_j = \widetilde{O}(n/\lambda_j^*)$. Moreover, observe that $\lambda_j^* \leq \lambda_H^i$ and a phase is executed whenever $\lambda_H^i \leq 3/2\lambda_j^*$. Thus, for all λ_H^i's belonging to superphase j, we get the following relation

$$\lambda_j^* \leq \lambda_H^i \leq 3/2\lambda_j^*, \tag{1}$$

which in turn implies that $n_j = \widetilde{O}(n/\lambda_j^*) = \widetilde{O}(n/\lambda_H^i)$. ◀

For the remaining steps, we divide the running time analysis into two parts (one part corresponding to phases, and the other to superphases).

Part 1

For some superphase j, the i-th phase consists of the i-th execution of a `Partial Rebuild Step` followed by the execution of Step 2. Let u_i be the number of edge insertions in Phase i. The total time for Step 2 is $O(n_j + u_i \log n) = \widetilde{O}(n + u_i)$. Using Fact 9, we observe that $\bigcup_{i \leq \lambda_H + 1} F_i \cup N_h$ has size $O(u_{i-1} + \lambda_H^i n_j) = \widetilde{O}(u_{i-1} + n)$. Thus, the total time for computing DA-msfd in a `Partial Rebuild Step` is $\widetilde{O}(u_{i-1} + n)$. Similarly, since H' has $O(\lambda_H^i n_j) = \widetilde{O}(n)$ edges, it takes $\widetilde{O}(n)$ time to compute $I(H', \lambda_H^i)$ and the new cactus tree.

The total time spent in Phase i is $\widetilde{O}(u_{i-1} + u_i + n)$. Let λ and λ_H denote the size of the minimum cut in the final graph and its corresponding multigraph, respectively. Note that

$\sum_{i=1}^{\lambda} u_i \leq m_1$, $\lambda n \leq m_0 + m_1$ and recall Eqn. (1). This gives that the total work over all phases is

$$\sum_{i=1}^{\lambda_H} \widetilde{O}\left(u_{i-1} + u_i + n\right) = \sum_{i=1}^{\lambda} \widetilde{O}\left(u_{i-1} + u_i + n\right) = \widetilde{O}(m_0 + m_1).$$

Part 2

The j-th superphase consists of the j-th execution of a `Full Rebuild Step` along with a possible execution of a `Special Step`, depending on whether the condition is met. In a `Full Rebuild Step`, the total running time for constructing H, $I(H, \lambda_j^*)$ and the cactus tree is dominated by $\widetilde{O}(m_0 + m_1 + (\lambda_j^*)^2 \cdot (n/\lambda_j^*)) = \widetilde{O}(m_0 + m_1)$. The running time of a Special Step is $\widetilde{O}(m_1)$.

Throughout its execution, the algorithm begins a new superphase whenever $\lambda(G) = \min \{\textsc{Min}(\mathcal{H}_G), \lambda_H\} > 3/2\lambda^*$. This implies that $\lambda(G)$ must be at least $3/2\lambda^*$, where λ^* is the value of $\lambda(G)$ at the last `Full Rebuild Step`. Thus, a new superphase begins whenever $\lambda(G)$ has increased by a factor of $3/2$, i.e., only $O(\log n)$ times over all insertions. This gives that the total time over all superphases is $\widetilde{O}(m_0 + m_1)$. ◀

5 Incremental $(1 + \varepsilon)$ Minimum Cut with $\widetilde{O}(n)$ space

In this section we present two $\widetilde{O}(n)$ space incremental Monte-Carlo algorithms that w.h.p maintain the size of a min-cut up to a $(1+\varepsilon)$-factor. Both algorithms have $\widetilde{O}(1)$ update-time and $\widetilde{O}(1)$, resp. $O(1)$ query-time.

5.1 An $O(n \log^2 n/\varepsilon^2)$ space algorithm

Our first algorithm follows an approach that was used in several previous works [14, 28, 27], where the space requirement is not considered. The basic idea is to maintain the min-cut up to some size k using small space. We achieve this by maintaining a sparse k-certificate and incorporating it into the incremental exact min-cut algorithm due to Henzinger [14], as described in Section 4. Finally we apply the well-known randomized sparsification result due to Karger [17] to obtain our result.

Maintaining min-cut up to size k using $O(kn)$ space

We incrementally maintain a DA-msfd for an unweighted multigraph G using k union-find data structures $\mathcal{F}_1, \ldots, \mathcal{F}_k$ (see [6]). Each $\mathcal{F}_i$ maintains a spanning forest F_i of G. Recall that $F_1, \ldots, F_k$ are edge-disjoint. When a new edge $e = (u, v)$ is inserted into G, we define i to be the first index such that $\mathcal{F}_i.\textsc{Find}(u) \neq \mathcal{F}_i.\textsc{Find}(v)$. If we found such an i, we append the edge e to the forest F_i by setting $\mathcal{F}_i.\textsc{Union}(u, v)$ and return i. If such an i cannot be found after k steps, we simply discard edge e and return NULL. We refer to such procedure as k-$\textsc{Connectivity}(e)$.

It is easy to see that the forests maintained by k-$\textsc{Connectivity}(e)$ for every newly inserted edge e are indeed edge-disjoint. Combining this procedure with techniques from Henzinger [14] leads to the following Algorithm 2.

The space requirement of the above algorithm is only $O(kn)$, since we always maintain at most k spanning forests during its execution. The total running time for testing the k-connectivity of the endpoints of the newly inserted edges in Step 2 is $O(km\alpha(n))$, where

Algorithm 2 Incremental Exact Min-Cut up to size k

```
1: Set $\lambda = 0$, initialize $k$ union-find data structures $\mathcal{F}_1, \ldots, \mathcal{F}_k$,
   $k$ empty forests $F_1, \ldots, F_k$, $I(\lambda)$, and an empty cactus tree.
2: while there is at least one minimum cut of size $\lambda$ do
       Receive the next operation.
       if it is a query then return $\lambda$
       else it is the insertion of an edge $e$, then
       Set $i = k$-Connectivity$(e)$.
           if $i \neq$ NULL then
             Set $F_i = F_i \cup \{e\}$.
             Update the cactus tree according to the insertion of the edge $e$.
           endif
       endif
   endwhile
3: Set $\lambda = \lambda + 1$.
   Let $G' = (V, E')$ be a graph with $E' = I(\lambda - 1) \cup \bigcup_{i \leq \lambda+1} F_i$.
   Compute $I(\lambda)$ and a cactus tree of $G'$.
   Goto 2.
```

$\alpha(n)$ stands for the inverse of Ackermann function. These guarantees combined with the arguments from Theorem 8 of Henzinger [14] give the following corollary.

▶ **Corollary 10.** *For $k > 0$, there is an $O(kn)$ space algorithm that processes a stream of edge insertions starting from any empty graph G and maintains an exact value of* $\min\{\lambda(G), k\}$. *The total time for inserting m edges is $O(km\alpha(n)\log n)$ and queries can be answered in constant time.*

Dealing with min-cuts of arbitrary size

We observe that Corollary 10 gives polylogarithmic amortized update time only for min-cuts up to some polylogarithmic size. For dealing with min-cuts of arbitrary size, we use the well-known sampling technique due to Karger [17]. This allows us to get an $(1+\varepsilon)$-approximation to the value of min-cut with high probability.

▶ **Lemma 11** ([17]). *Let G be any graph with minimum cut λ and let $p \geq 10(\log n)/(\varepsilon^2\lambda)$. Let $S(p)$ be a subgraph of G obtained by including each of edge of G to $S(p)$ with probability p independently. Then the probability that the value of any cut of $S(p)$ has value more than $(1+\varepsilon)$ or less than $(1-\varepsilon)$ times its expected value is $O(1/n^3)$.*

For some integer $i \geq 1$, let G_i denote a subgraph of G obtained by including each edge of G to G_i with probability $1/2^i$ independently. We now have all necessary tools to present our incremental algorithm:

1. For $i = 0, \ldots, \lfloor \log n \rfloor$, let G_i be the initially empty sampled subgraphs.
2. If an edge e is inserted into G, include e to each G_i with probability $1/2^i$ and maintain the exact minimum cut of G_i up to size $k = 40\log n/\varepsilon^2$ using Algorithm 2.
3. If the operation is a query, find the minimum j such that the min-cut of G_j is at most k. Return $2^j\lambda(G_j)$.

▶ **Theorem 12.** *There is an $O(n \log^2 n/\varepsilon^2)$ space randomized algorithm that processes a stream of edge insertions starting from an empty graph G and maintains a $(1+\varepsilon)$-approximation to the min-cut of G with high probability. The amortized update time per operation is $O(\alpha(n)\log^3 n/\varepsilon^2)$ and queries can be answered in $O(\log n)$ time.*

Proof. We first prove the correctness of the algorithm. For an integer $t \geq 0$, let $G^{(t)} = (V, E^{(t)})$ be the graph after the first t edge insertions. Further, let $\lambda(G^{(t)})$ denote the min-cut of $G^{(t)}$ and $p^{(t)} = 10(\log n)/(\varepsilon^2\lambda^{(t)})$. For any integer $i \leq \lfloor \log_2 1/p^{(t)} \rfloor$, Lemma 11 implies that $2^i\lambda(G_i^{(t)})$ is an $(1 \pm \varepsilon)$-approximation to $\lambda(G^{(t)})$. Setting $i = \lfloor \log_2 1/p^{(t)} \rfloor$, we get that:

$$\mathbb{E}[\lambda(G_i^{(t)})] \leq \lambda(G^{(t)})/2^i \leq 2p^{(t)}\lambda(G^{(t)}) \leq 20\log n/\varepsilon^2.$$

The later along with Lemma 11 imply that for any $\varepsilon \in (0,1)$, the size of the minimum cut in $G_i^{(t)}$ is at most $(1+\varepsilon)20\log n/\varepsilon^2 \leq 40\log n/\varepsilon^2$ with probability $1 - O(1/n^3)$. Thus, $j \leq \lfloor \log_2 1/p^{(t)} \rfloor$ and the algorithm returns a $(1 \pm \varepsilon)$-approximation to the minimum cut of $G^{(t)}$ with probability $1 - O(1/n^3)$. Note that for any t, $\lfloor \log_2 1/p^{(t)} \rfloor \leq \lfloor \log n \rfloor$, and thus it is sufficient to maintain only $O(\log n)$ sampled subgraphs.

Since our algorithm applies to unweighted simple graphs, we know that $t \leq O(n^2)$. Now applying union bound over all $t \in \{1, \ldots O(n^2)\}$ gives that the probability that the algorithm does not maintain a $(1 \pm \varepsilon) \leq 1 + O(\varepsilon)$-approximation is at most $O(1/n)$.

The total expected time for maintaining a sampled subgraph is $O(m\alpha(n)\log^2 n/\varepsilon^2)$ and the required space is $O(n \log n/\varepsilon^2)$ (Corollary 10). Maintaining $O(\log n)$ such subgraphs gives an $O(\alpha(n)\log^3 n/\varepsilon^2)$ amortized time per edge insertion and an $O(n\log^2 n/\varepsilon^2)$ space requirement. The $O(\log n)$ query time follows as in the worst case we scan at most $O(\log n)$ subgraphs, each answering a min-cut query in constant time. ◀

5.2 Improving the space to $O(n \log n/\varepsilon^2)$

We next show how to bring down the space requirement of the previous algorithm to $O(n\log n/\varepsilon^2)$ without degrading its running time. The main idea is to keep a single sampled subgraph instead of $O(\log n)$ of them.

Let $G = (V, E)$ be an unweighted undirected graph and assume each edge is given some random weight p_e chosen uniformly from $[0,1]$. We call the resulting weighted graph G^w. For any $p > 0$, we denote by $G(p)$ the unweighted subgraph of G that consists of all edges that have weight at most p. We state the following lemma due to Karger [15]:

▶ **Lemma 13.** *Let $k = 40\log n/\varepsilon^2$. Given a connected graph G, let p be a value such that $p \geq k/(4\lambda(G))$. Then with high probability, $\lambda(G(p)) \leq k$ and $\lambda(G(p))/p$ is an $(1+\varepsilon)$-approximation to the min-cut of G.*

Proof. Since the weight of every edge is uniformly distributed, the probability that an edge has weight at most p is exactly p. Thus, $G(p)$ can be viewed as taking G and including each edge with probability p. The claim follows from Lemma 11. ◀

For any graph G and some appropriate weight p, the above lemma tells us that the min-cut of $G(p)$ is bounded by k with high probability. Thus, instead of considering the graph G along with its random edge weights, we build a collection of k minimum edge-disjoint spanning forests (using those edge weights). We note that such a collection is a DA-msfd of order k for G with $O(kn)$ edges and by Lemma 3, it preserves all minimum cuts of G up to size k.

Our algorithm uses the following two data structures:

(1) NI-Sparsifier(k) data-structure: Given a graph G, where each edge e is assigned some weight p_e and some parameter k, we maintain an insertion-only data-structure that maintains a collection of k minimum edge-disjoint spanning forests $S_1, \dots, S_k$ with respect to the edge weights. Let $S = \bigcup_{i=1}^{k} S_i$. Since we are in the incremental setting, it is known that the problem of maintaining a single minimum spanning forest can be solved in time $O(\log n)$ per insertion using the dynamic tree structure of Sleator and Tarjan [26]. Specifically, we use this data-structure to determine for each pair of nodes (u, v) the maximum weight of an edge in the cycle that the edge (u, v) induces in the minimum spanning forest S_i. Let max-weight$(S_i(u, v))$ denote such a maximum weight. The update operation works as follows: when a new edge $e = (u, v)$ is inserted into G, we first use the dynamic tree data structure to test whether u and v belong to the same tree. If no, we link their two trees with the edge (u, v) and return the pair (TRUE, NULL) to indicate that e was added to S_i and no edge was evicted from S_i. Otherwise, we check whether $p_e >$ max-weight$(S_i(e))$. If the latter holds, we make no changes in the forest and return (FALSE, e). Otherwise, we replace one of the maximum edges, say e', on the path between u and v in the tree by e and return (TRUE, e'). The boolean value that is returned indicates whether e belongs to S_i or not, the second value that is returned gives an edge that does not (or no longer) belong to S_i. Note that each edge insertion requires $O(\log n)$ time. We refer to this insert operation as INSERT-MSF(S_i, e, p_e).

Now, the algorithm that maintains the weighted minimum spanning forests implements the following operations:

- INITIALIZE-NI(k): initializes the data structure for k empty minimum spanning forests.
- INSERT-NI(e, p_e): Set $i = 1$, $e' = e$, taken = FALSE.

```
while (($i \le k$) and $e' \neq$ NULL) do
    Set ($t'$, $e''$) = INSERT-MSF($S_i, e', p_{e'}$).
    if ($e' = e$) then set taken = $t'$ endif
    Set $e' = e''$ and $i = i + 1$.
endwhile
if ($e' \neq e$) then return (taken, $e'$) else return (taken, NULL).
```

Recall that $S = \bigcup_{i \le k} S_i$. We use the abbreviation NI-SPARSIFIER(k) to refer to this data-structure. By slight abuse of notation we will associate a weight with each edge in S and use S^w to refer to this weighted version of S.

▶ **Lemma 14.** *For $k > 0$ and any graph G,* NI-SPARSIFIER(k) *maintains a weighted* DA-msfd of order k of G under edge insertions. The algorithm uses $O(kn)$ space and the total time for inserting m edges is $O(km \log n)$.

(2) Limited Exact Min-Cut(k) data-structure: We use Algorithm 2 to implement the following operations for any unweighted graph G and parameter k,

- INSERT-LIMITED(e): executes the insertion of edge e into Algorithm 2.
- QUERY-LIMITED(): returns λ
- INITIALIZE-LIMITED(G, k): builds a data structure for G with parameter k by calling INSERT-LIMITED(e) for each edge e in G.

We use the abbreviation LIM(k) to refer to such data-structure.

Combining the above data-structures leads to the following algorithm:

Correctness and Running Time Analysis

Let S denote the unweighted version of S^w. Throughout the execution of Algorithm 3, S corresponds exactly to the DA-msfd of order k of G maintained by NI-SPARSIFIER(k). In

Algorithm 3 $(1+\varepsilon)$-Min-Cut with $O(n \log n/\varepsilon^2)$ Space

```
1: Set $k = 40 \log n/\varepsilon^2$.
   Set $p = 10 \log n/\varepsilon^2$.
   Let $H$ and $S^w$ be empty graphs.
2: Initialize-Limited$(H, k)$.
   while Query-Limited() $< k$ do
       Receive the next operation.
       if it is a query then return Query-Limited()$/\min\{1, p\}$.
       else it is the insertion of an edge $e$, then
       Sample a random weight from $[0, 1]$ for the edge $e$ and denote it by $p_e$.
       if $p_e \leq p$ then Insert-Limited$(e)$ endif
       Set (taken, $e'$) = Insert-NI$(e, p_e)$.
           if taken then
             Insert $e$ into $S^w$ with weight $p_e$.
             if ($e' \neq$ NULL) then remove $e'$ from $S^w$.
           endif
       endif
   endwhile
3: // Rebuild Step
   Set $p = p/2$.
   Let $H$ be the unweighted subgraph of $S^w$ consisting of all edges of weight at most $p$.
   Goto 2.
```

the following, let H be the graph that is given as input to Lim(k). Thus, by Corollary 10, Query-Limited() returns $\min\{k, \lambda(H)\}$, i.e., it returns $\lambda(H)$ as long as $\lambda(H) \leq k$. We now formally prove the correctness.

▶ **Lemma 15.** *Let $\epsilon \leq 1$. If $\lambda(G) < k$, then $H = G$, $p = k/4$, and* Query-Limited() *returns $\lambda(G)$. The first rebuild step is triggered after the first insertion that increases $\lambda(G)$ to k and let $\lambda(G) = \lambda(H) = k$ at that time.*

Proof. The algorithm starts with an empty graph G, i.e., initially $\lambda(G) = 0$. Throughout the sequence of edge insertions $\lambda(G)$ never decreases. We show by induction on the number m of edge insertions that $H = G$ and $p = k/4$ as long as $\lambda(G) < k$.

Note that $k/4 \geq 1$ by our choice of ϵ. For $m = 0$, the graphs G and H are both empty graphs and p is set to $k/4$. For $m > 0$, consider the m-th edge insertion, which inserts an edge e. Let G and H denote the corresponding graphs after the insertion of e. By the inductive assumption, $p = k/4$ and $G \setminus \{e\} = H \setminus \{e\}$. As $p \geq 1$, e is added to H and, thus, it follows that $G = H$. Hence, $\lambda(H) = \lambda(G)$. If $\lambda(G) < k$, no rebuild is performed and p is not changed. If $\lambda(G) = k$, then the last insertion was exactly the insertion that increased $\lambda(G)$ from $k - 1$ to k. As $H = G$ before the rebuild, Query-Limited() returns k, triggering the first execution of the rebuild step. ◀

We next analyze the case that $\lambda(G) \geq k$. In this case, both H and p are random variables, as they depend on the randomly chosen weights for the edges. Let $S(p)$ be the unweighted subgraph of S^w that contains all edge of weight at most p.

▶ **Lemma 16.** *Let $N_h(p)$ be the graph consisting of all edges that were inserted after the last rebuild and have weight at most p and let $S^{old}(p)$ be $S(p)$ right after the last rebuild. Then the graph $H = S^{old}(p) \cup N_h(p)$.*

▶ **Lemma 17.** *At the time of a rebuild $S(p)$ is a DA-msfd of order k of $G(p)$.*

By Lemma 13, in order to show that $\lambda(H)/\min\{1,p\}$ is an $(1+\varepsilon)$-approximation of $\lambda(G)$ with high probability, we need to show that if $\lambda(G) \geq k$ then (a) the random variable p is at least $k/(4\lambda(G))$ w.h.p., which implies that $\lambda(G(p))$ is a $(1+\varepsilon)$-approximation of $\lambda(G)$ w.h.p. , and (b) that $\lambda(H) = \lambda(G(p))$.

▶ **Lemma 18.** *Let $\varepsilon \leq 1$. If $\lambda(G) \geq k$, then (1) $p \geq k/(4\lambda(G))$ w.h.p. and (2) $\lambda(H) = \lambda(G(p))$.*

Proof. For any $i \geq 0$, after the i-th rebuild we have $p = p^{(i)} := 10 \log n/(2^i\varepsilon^2)$. We will show by induction on i that (1) $p^{(i)} = 10 \log n/(2^i\varepsilon^2) \geq 10 \log n/(\varepsilon^2\lambda(G))$ with high probability, which is equivalent to showing that $\lambda(G) \geq 2^i$ and that (2) at any point between the $i-1$-st and the i-th rebuild, $\lambda(H) = \lambda(G(p^{(i-1)}))$.

We first analyse $i = 1$. Assume that the insertion of edge e caused the first rebuild. Lemma 15 showed that (1) at the first rebuild $\lambda(G) = k \geq 2^1 = 2$ and (2) that up to the first rebuild $G(p) = G = H$.

For the induction step ($i > 1$), we inductively assume that (1) at the $(i-1)$-st rebuild, $p^{(i-1)} \geq 10 \log n/\varepsilon^2\lambda(G^{\text{old}})$ with high probability, where G^{old} is the graph G right before the insertion that triggered the i-th rebuild (i.e., at the last point in time when Query-Limited() returned a value less than k), and (2) that $\lambda(H) = \lambda(G(p^{(i-2)}))$ at any time between the $(i-2)$-nd and the $(i-1)$-st rebuild. Let e be the edge whose insertion caused the i-th rebuild. Define $G^{\text{new}} = G^{\text{old}} \cup \{e\}$. Note that w.h.p. $p^{(i-1)} \geq 10 \log n/(\varepsilon^2\lambda(G^{\text{old}})) \geq 10 \log n/(\varepsilon^2\lambda(G^{\text{new}}))$ as $\lambda(G^{\text{old}}) \leq \lambda(G^{\text{new}})$. Thus, by Lemma 13, we get that $\lambda(G^{\text{new}}(p^{(i-1)}))/p^{(i-1)} \leq (1+\varepsilon)\lambda(G^{\text{new}})$ with high probability.

We show below that $\lambda(G^{\text{new}}(p^{(i-1)})) = \lambda(H^{\text{new}})$, where H^{new} is the graph stored in Lim(k) right before the i-th rebuild. Thus, $\lambda(H^{\text{new}}) = k$, which implies that

$$\begin{aligned}\lambda(G^{\text{new}}(p^{(i-1)})) = k = 40 \log n/\varepsilon^2 &\leq (1+\varepsilon)\lambda(G^{\text{new}}) \cdot p^{(i-1)} \\ &= (1+\varepsilon)\lambda(G^{\text{new}}) \cdot 10 \log n/(2^{i-1}\varepsilon^2),\end{aligned}$$

w.h.p.. This in turn implies that $\lambda(G^{\text{new}}) \geq 2^{i+1}/(1+\varepsilon) \geq 2^i$ w.h.p. by our choice of ε.

It remains to show that $\lambda(G^{\text{new}}(p^{(i-1)})) = \lambda(H^{\text{new}})$. Note that this is a special case of (2), which claims that at any point between that $(i-1)$-st and the i-th rebuild $\lambda(H) = \lambda(G(p^{(i-1)}))$, where H and G are the current graphs. Thus, to complete the proof of the lemma it suffices to show (2).

As H is a subgraph of $G(p^{(i-1)})$, we know that $\lambda(G(p^{(i-1)})) \geq \lambda(H)$. Thus, we only need to show that $\lambda(G(p^{(i-1)})) \leq \lambda(H)$. Let G^{i-1}, resp. S^{i-1}, resp. H^{i-1}, be the graph G, resp. S, resp. H, right after rebuild $i-1$ and let N_h be the set of edges inserted since, i.e., $G = G^{(i-1)} \cup N_h$. As we showed in Lemma 16, $H = S^{i-1}(p^{(i-1)}) \cup N_h(p^{(i-1)})$. Thus, $H^{i-1} = S^{i-1}(p^{(i-1)})$. Additionally, by Lemma 17, $S^{i-1}(p^{(i-1)})$ is a DA-msfd of order k of $G^{i-1}(p^{(i-1)})$. Thus by Property (3) of a DA-msfd of order k, for every cut $(A, V \setminus A)$ of value at most k in H^{i-1}, $\lambda(H^{i-1}, A) = \lambda(S^{i-1}(p^{(i-1)}), A) = \lambda(G^{i-1}(p^{(i-1)}), A)$, where $\lambda(G, A)$ denotes the number of edges crossing from A to $V \setminus A$ in G. Now assume by contradiction that $\lambda(G(p^{(i-1)})) > \lambda(H)$ and consider a minimum cut $(A, V \setminus A)$ in H, i.e., $\lambda(H) = \lambda(H, A)$. We know that at any time $k \geq \lambda(H)$. Thus $k \geq \lambda(H) = \lambda(H, A)$, which implies $k \geq \lambda(H^{i-1}, A)$. By Property (3) of DA-msfd it follows that $\lambda(H^{i-1}, A) = \lambda(G^{i-1}(p^{(i-1)}), A)$. Note that $H = H^{i-1} \cup N_h(p^{(i-1)})$ and $G(p^{(i-1)}) = G^{i-1}(p^{(i-1)}) \cup E_H(p^{(i-1)})$. Let x be the number of edges of $N_h(p^{(i-1)})$ that cross the cut $(A, V \setminus A)$. Then $\lambda(H) = \lambda(H, A) = \lambda(H^{i-1}, A) + x = \lambda(G^{i-1}(p^{(i-1)}), A) + x = \lambda(G(p^{(i-1)}), A)$, which contradicts the assumption that $\lambda(G(p^{(i-1)})) > \lambda(H)$. ◀

Since our algorithm is incremental and applies only to unweighted graphs, we know that there can be at most $O(n^2)$ edge insertions. Thus, by the above lemma and an union bound over these $O(n^2)$ different graphs, we get that throughout its execution, our algorithm maintains a $(1+\varepsilon)$-approximation to the min cut with high probability.

▶ **Theorem 19.** *There is an $O(n \log n/\varepsilon^2)$ space randomized algorithm that processes a stream of edge insertions starting from an empty graph G and maintains a $(1+\varepsilon)$-approximation to the min-cut of G with high probability. The total time for insertiong m edges is $O(m\alpha(n) \log^3 n/\varepsilon^2)$ and queries can be answered in constant time.*

Proof. The space requirement is $O(n \log n/\varepsilon^2)$ since at any point of time, the algorithm keeps H, S^w, Lim(k), and NI-Sparsifier (k), each of size at most $O(n \log n/\varepsilon^2)$ (Corollary 10 and Lemma 14).

When Algorithm 3 executes a `Rebuild Step`, only the Lim(k) data-structure is rebuilt, but not NI-Sparsifier(k). During the whole algorithm m Insert-NI operations are performed. Thus, by Lemma 14, the total time for all operations involving NI-Sparsifier(k) is $O(m \log^2 n/\varepsilon^2)$.

It remains to analyze Steps 2 and 3. In Step 2, Initialize-Limited(H, k) takes at most $O(m\alpha(n) \log^2 n/\varepsilon^2)$ total time (Corollary 10). The running time of Step 3 is $O(m)$ as well. Since the number of `Rebuild Steps` is at most $O(\log n)$, it follows that the total time for all Initialize-Limited(H, k) calls in Steps 2 and the total time of Step 3 throughout the execution of the algorithm is $O(m\alpha(n) \log^3 n/\varepsilon^2)$.

We are left with analyzing the remaining part of Step 2. Each query operation executes one Query-Limited() operation, which takes constant time. Each insertion executes one Insert-NI(e, p_e) operation, which takes amortized time $O(\log^2 n/\varepsilon)$. We maintain the edges of S^w in a binary tree so that each insertion and deletion takes $O(\log n)$ time. As there are m edge insertions the remaining part of Step 2 takes total time $O(m \log^2 n/\varepsilon^2)$. Combining the above bounds gives the theorem. ◀

References

1 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *Proc. of the 55th FOCS*, pages 434–443. IEEE, 2014.

2 Kook Jin Ahn and Sudipto Guha. Graph sparsification in the semi-streaming model. In *Proc. of the 36th ICALP*, pages 328–338, 2009.

3 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Proc. of the 32nd PODS*, pages 5–14, 2012.

4 András A. Benczúr and David R. Karger. Randomized approximation schemes for cuts and flows in capacitated graphs. *SIAM J. Comput.*, 44(2):290–319, 2015.

5 Sayan Bhattacharya, Monika Henzinger, Danupon Nanongkai, and Charalampos E. Tsourakakis. Space- and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams. In *Proc. of the 47th STOC*, pages 173–182, 2015.

6 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms (3. ed.)*. MIT Press, 2009.

7 E. A. Dinitz, A. V. Karzanov, and M. V. Lomonosov. On the structure of a family of minimum weighted cuts in a graph. *Studies in Discrete Optimization*, pages 290–306, 1976.

8 Yefim Dinitz and Jeffery Westbrook. Maintaining the classes of 4-edge-connectivity in a graph on-line. *Algorithmica*, 20(3):242–276, 1998.

9 Harold N. Gabow. Applications of a poset representation to edge connectivity and graph rigidity. In *Proc. of the 32nd FOCS*, pages 812–821, 1991.

10 Harold N. Gabow. A matroid approach to finding edge connectivity and packing arborescences. *J. Comput. Syst. Sci.*, 50(2):259–273, 1995.

11 Zvi Galil and Giuseppe F. Italiano. Maintaining the 3-edge-connected components of a graph on-line. *SIAM J. Comput.*, 22(1):11–28, 1993.

12 David Gibb, Bruce M. Kapron, Valerie King, and Nolan Thorn. Dynamic graph connectivity with improved worst case update time and sublinear space. *CoRR*, abs/1509.06464, 2015.

13 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proc. of the 47th STOC*, pages 21–30, 2015.

14 Monika Rauch Henzinger. A static 2-approximation algorithm for vertex connectivity and incremental approximation algorithms for edge and vertex connectivity. *Journal of Algorithms*, 24(1):194–220, 1997.

15 David Karger. *Random Sampling in Graph Optimization Problems.* PhD thesis, Stanford University, Stanford, 1994.

16 David R. Karger. Using randomized sparsification to approximate minimum cuts. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms. 23-25 January 1994, Arlington, Virginia.*, pages 424–432, 1994.

17 David R. Karger. Random sampling in cut, flow, and network design problems. *Math. Oper. Res.*, 24(2):383–413, 1999.

18 David R. Karger. Minimum cuts in near-linear time. *J. ACM*, 47(1):46–76, 2000.

19 Ken-ichi Kawarabayashi and Mikkel Thorup. Deterministic global minimum cut of a simple graph in near-linear time. In *Proc. of the 47th STOC*, pages 665–674, 2015.

20 Jonathan A. Kelner and Alex Levin. Spectral sparsification in the semi-streaming setting. *Theory Comput. Syst.*, 53(2):243–262, 2013.

21 Jakub Lacki and Piotr Sankowski. Min-cuts and shortest cycles in planar graphs in $O(n \log\log n)$ time. In *Proc. of the 19th ESA*, pages 155–166, 2011.

22 Karl Menger. Zur allgemeinen kurventheorie. *Fundamenta Mathematicae*, 1(10):96–115, 1927.

23 Hiroshi Nagamochi and Toshihide Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, 7(5&6):583–596, 1992.

24 Danupon Nanongkai and Thatchaphol Saranurak. Dynamic cut oracle. under submission, 2016.

25 Johannes A. La Poutré. Maintenance of 2- and 3-edge-connected components of graphs II. *SIAM J. Comput.*, 29(5):1521–1549, 2000.

26 Daniel Dominic Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983.

27 Mikkel Thorup. Fully-dynamic min-cut. *Combinatorica*, 27(1):91–127, 2007.

28 Mikkel Thorup and David R Karger. Dynamic graph algorithms with applications. In *Algorithm Theory-SWAT 2000*, pages 1–9. Springer, 2000.

Packing and Covering with Non-Piercing Regions

Sathish Govindarajan[1], Rajiv Raman[2], Saurabh Ray[3], and Aniket Basu Roy[4]

1 **Indian Institute of Science, Bangalore, India**
gsat@iisc.ernet.in
2 **IIIT-Delhi, Delhi, India**
rajiv@iiitd.ac.in
3 **NYU Abu Dhabi, UAE**
saurabh.ray@nyu.edu
4 **Indian Institute of Science, Bangalore, India**
aniket.basu@csa.iisc.ernet.in

Abstract

In this paper, we design the first polynomial time approximation schemes for the Set Cover and Dominating Set problems when the underlying sets are non-piercing regions (which include pseudodisks). We show that the local search algorithm that yields PTASs when the regions are disks [5, 19, 28] can be extended to work for non-piercing regions. While such an extension is intuitive and natural, attempts to settle this question have failed even for pseudodisks. The techniques used for analysis when the regions are disks rely heavily on the underlying geometry, and do not extend to topologically defined settings such as pseudodisks. In order to prove our results, we introduce novel techniques that we believe will find applications in other problems.

We then consider the Capacitated Region Packing problem. Here, the input consists of a set of points with capacities, and a set of regions. The objective is to pick a maximum cardinality subset of regions so that no point is covered by more regions than its capacity. We show that this problem admits a PTAS when the regions are k-admissible regions (pseudodisks are 2-admissible), and the capacities are bounded. Our result settles a conjecture of Har-Peled (see Conclusion of [20]) in the affirmative. The conjecture was for a weaker version of the problem, namely when the regions are pseudodisks, the capacities are uniform, and the point set consists of all points in the plane.

Finally, we consider the Capacitated Point Packing problem. In this setting, the regions have capacities, and our objective is to find a maximum cardinality subset of points such that no region has more points than its capacity. We show that this problem admits a PTAS when the capacity is unity, extending one of the results of Ene et al. [16].

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Local Search, Set Cover, Dominating Set, Capacitated Packing, Approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.47

1 Introduction

Geometric packing and covering problems have received wide attention in the last decade, especially in the context of approximation algorithms. Besides the inherent aesthetic appeal, the interest in the geometric setting arises from the fact that in many applications, the packing and covering problems involve geometric objects. For example, see [3, 4, 14, 24, 30]. Several tools and techniques have been developed for this purpose, but for many fundamental problems there are still large gaps between the known approximation factors, and the existing hardness results.

Classic techniques for solving packing and covering problems rely on grid-shifting techniques introduced by Hochbaum and Maass [22], and extensions by Erlebach et al. [17] and Chan [10]. All these algorithms are restricted to the setting where the regions are *fat*. Recent progress has been based mainly on two paradigms. The first is algorithms that use LP rounding [7, 9, 12, 18], and the other is *local search* (albeit only in the unweighted setting). Local Search has been used to obtain PTASs[1] for several problems besides packing and covering. For example, see [5, 12, 19, 23, 28] and [8, 13] for more recent work.

Har-Peled and Chan [12], and Mustafa and Ray [28] obtained PTASs for the Independent Set and Hitting Set problems respectively, via local search when the underlying regions are non-piercing[2]. Non-piercing regions constitute a general setting studied widely, examples of which include disks, homothets of convex objects, unit height rectangles, arbitrary sized squares, etc. In contrast, for the Set Cover and Dominating Set problems, PTASs exist only when the underlying regions are disks [5, 19, 28]. Since these are natural and important problems, there have been attempts to extend these results to more general settings. The main difficulty is that the analysis for the case of disks relies heavily on the geometry. Durocher and Fraser [15] showed the existence of a PTAS for the Set Cover problem when the regions are pseudodisks satisfying a *cover-free* condition by *dualizing* and converting the problem to a Hitting Set problem. Further, they show that the approach of dualizing the problem can not be extended to work for a general family of pseudodisks.

In this paper, we develop new techniques to analyze the local search algorithm for problems when the underlying regions are non-piercing. These techniques lead to the first PTAS for the unweighted Set Cover and Dominating Set problems when the underlying regions are non-piercing. In the weighted setting, Chan et al. [11] building on the work of Varadarajan [31] obtained $O(1)$-approximation algorithms for the Set Cover and Dominating Set problems for non-piercing regions with low union complexity. For the Set Cover problem, the current best result is a QPTAS [27] that extends the technique of Adamaszek and Wiese [1, 2] which obtains QPTAS for the Independent Set problem for polygons.

We also develop new techniques for obtaining a PTAS for the Capacitated Region packing problem when the capacities are bounded by a constant and the regions are k-admissible[3]. This result proves a conjecture of Har-Peled [20]. We also consider the dual problem, namely Capacitated Point Packing for non-piercing regions. We show that it admits a PTAS using local search for the special case when the capacity is unity, extending a result of Ene et al. [16], who obtained a PTAS for Capacitated Point Packing for disks with unit capacity in the plane.

2 Preliminaries

Two compact, simply connected regions A, B are said to be *non-piercing* if both $A \setminus B$ and $B \setminus A$ are connected. A set $\mathcal{X}$ of compact, simply connected regions is non-piercing if the regions in $\mathcal{X}$ are pairwise non-piercing. For a region A, let $\partial(A)$ denote the boundary of A. We assume $\partial(A)$ is oriented counter-clockwise. The boundary divides the plane into two

[1] A polynomial time $(1+\epsilon)$-approximation algorithm for any $\epsilon > 0$.
[2] A set of simply connected regions is said to be non-piercing if for any pair A, B of regions, the sets $A \setminus B$ and $B \setminus A$ are connected.
[3] k-admissible regions are non-piercing regions whose boundaries intersect at most k times.

regions the interior of A, denoted $int(A)$, and the exterior of A, denoted $ext(A)$. By the orientation of $\partial(A)$, $int(A)$ lies to the left of $\partial(A)$. We further assume that any pair of regions in $\mathcal{X}$ intersect *properly* by which we mean that for any two regions A, B, $\partial(A) \cap \partial(B)$, i.e., the points of intersection of their boundaries is a finite set, and at each point of intersection, their boundaries cross. In this paper, when we use the term *region*, we implicitly mean that the region is compact and simply connected, and a set of regions is assumed to be properly intersecting.

In some applications considered in this paper, we are given a set $\mathcal{R}$ of regions as well as a set P of points in $\mathbb{R}^2$. In this case, we assume that the regions in $\mathcal{R}$ intersect properly, and each point $p \in P$ is at least at a distance $\epsilon > 0$ away from the boundary of any region in $\mathcal{R}$.

We will use Lemma 6 from [29], which we state here for completeness.

▶ **Lemma 1** ([29]). *Given a set of non-piercing regions $\mathcal{X}$ and a set P of points in the plane, we can construct a plane graph*[4] *$H = (P, E)$ in polynomial time such that for any region $X \in \mathcal{X}$, the induced sub-graph on the set of points in $P \cap X$ is connected. Furthermore, the sub-graph formed by edges of H lying within X also form a connected sub-graph on $P \cap X$.*

The Lemma as stated is slightly more general than the statement in [29]. However, this follows from their proof.

3 Local Search Framework

The local search algorithm we use in this paper is the following:

Local Search Algorithm: For a parameter k, start with a feasible solution. At each iteration, attempt to find a better feasible solution by swaps of a bounded size k of objects in the current solution with objects not in the solution. Stop and return the current solution when no such swap is possible.

In the rest of the paper, when we say the "local search algorithm", we implicitly refer to the algorithm above. The running time of this algorithm is $n^{O(k)}$. We choose the local search parameter k to be an appropriate polynomial in $1/\epsilon$ to achieve the approximation factor of $(1 + \epsilon)$ [5, 12, 28].

Let $\mathcal{R}$ and $\mathcal{B}$ denote an optimal solution and the solution returned by the local search algorithm, respectively. To analyze the approximation factor of the algorithm, we need to construct a suitable bipartite graph on the elements in $\mathcal{R}$ and $\mathcal{B}$. We refer to the exposition by Aschner et al. [5] as well as [12, 28] for a more complete description of this framework and its analysis. The following Theorem then follows.

▶ **Theorem 2** ([5, 12, 28]). *Consider a problem Π.*

1. *Suppose Π is a minimization problem. If there exists a bipartite graph $H = (\mathcal{R} \cup \mathcal{B}, E)$, that belongs to a family of graphs having a balanced vertex separator of sub-linear size, and it satisfies the* local-exchange property*: For any subset $\mathcal{B}' \subseteq \mathcal{B}$, $(\mathcal{B} \setminus \mathcal{B}') \cup N(\mathcal{B}')$ is a feasible solution. Then, the Local Search algorithm is a PTAS for Π. Here, $N(\mathcal{B}')$ denotes the set of neighbors of $\mathcal{B}'$ in H.*
2. *Suppose Π is a maximization problem. If there exists a bipartite graph $H = (\mathcal{R} \cup \mathcal{B}, E)$ that belongs to a family of graphs having a balanced vertex separator of sub-linear size,*

[4] An embedding of a planar graph in the plane such that the vertices are points and edges are continuous curves between the end-points.

and such that it satisfies the local-exchange property*: For any $\mathcal{R}' \subseteq \mathcal{R}$, $(\mathcal{B} \cup \mathcal{R}') \setminus N(\mathcal{R}')$ is a feasible solution. Then, the Local Search algorithm is a PTAS for* Π. *Here, as above $N(\mathcal{R}')$ denotes the set of neighbors of $\mathcal{R}'$ in H.*

Note that it can be assumed that $\mathcal{R} \cap \mathcal{B} = \emptyset$. This is because, otherwise, the common elements can be removed from both the sets and then the analysis can be restricted to the modified sets.

In Sections 6, 7, and 8, we construct graphs satisfying the conditions of Theorem 2 above, and thereby obtain a PTAS for the Dominating Set, Set Cover, Capacitated Region Packing, and Capacitated Point Packing problems. *Note that we only need to show the existence of these graphs and we do not require their construction to be algorithmic as they are only used in the analysis and not in the algorithm.*

4 Our Results

In this paper, we study the following problems.

Set Cover: Given a finite set of non-piercing regions $\mathcal{X}$, and a set of points P covered by the union of the regions in $\mathcal{X}$, compute $\mathcal{Y} \subseteq \mathcal{X}$ of smallest cardinality such that for each $p \in P$, there exists $Y \in \mathcal{Y}$ s.t., $p \in Y$.

Note that when the regions are disks, a PTAS for this problem follows from a PTAS for the Hitting-Set problem of halfspaces in $\mathbb{R}^3$ [28] via lifting. However, this technique does not generalize even for pseudodisks. An appealing approach is to try to *dualize* the problem, and use results for the Hitting Set problem. However, as Durocher and Frazer [15] observed, such a dual does not exist. Currently, the best approximation algorithm is a QPTAS given by Mustafa et al., [27] that works even in the weighed setting. In the unweighted setting, the work of Har-Peled and Quanrud [21] implies a PTAS for the above problem under the assumption that the regions are fat, and that no point is contained in more than a constant number of regions. We obtain a PTAS without these assumptions. The only requirement is that the regions are non-piercing, and in this sense, our work complements the work of Har-Peled and Quanrud [21].

▶ **Theorem 3.** *The Local Search algorithm yields a PTAS for the geometric Set Cover problem when the regions are non-piercing.*

Dominating Set: Given a finite set of non-piercing regions $\mathcal{X}$, find a subset $\mathcal{Y} \subseteq \mathcal{X}$ of smallest cardinality such that for each $X \in \mathcal{X}$, there exists $Y \in \mathcal{Y}$ so that $Y \cap X \neq \emptyset$.

Gibson and Pirwani [19] gave a PTAS for this problem, via local search when restricted to disks. However, their construction of the planar graph uses power diagrams [6] which strongly relies on the fact that the regions are circular disks. It is not clear how one could generalize their result to the setting of non-piercing regions, or even pseudodisks. Har-Peled and Quanrud [21] prove that local search yields a PTAS for *low-density graphs.* However, it is not clear how we could apply these techniques even in the setting where the regions are fat. We obtain a PTAS for this problem when the regions are non-piercing. In Section 6, we prove the following:

▶ **Theorem 4.** *The Local Search algorithm yields a PTAS for the Dominating Set problem in the intersection graph of non-piercing regions.*

Capacitated Region Packing: Given a finite set of non-piercing regions $\mathcal{X}$, a set of points P each having a constant capacity ℓ, compute the largest cardinality set $\mathcal{Y} \subseteq \mathcal{X}$ such that each point of P is contained in at most ℓ regions of $\mathcal{Y}$.

Currently, the best algorithm for this problem is by Ene et al., [16] that is an $O(1)$-approximation. This algorithm works even in the weighted setting as long as the regions have linear union complexity. Aschner et al. [5] gave PTAS for fat objects, Har-Peled [20] gave a QPTAS for family of pseudodisks. The results in [5, 20] is for the special case where $P = \mathbb{R}^2$. Har-Peled [20] conjectured that a PTAS must exist for this problem. Indeed, we obtain a PTAS for the more general problem. In Section 7 we prove the following theorem.

▶ **Theorem 5.** *The Local search algorithm yields a PTAS for the Capacitated Region Packing problem when the regions are k-admissible for $k = O(1)$, and the capacities are bounded above by a constant.*

Capacitated Point Packing: Given a finite set of non-piercing regions $\mathcal{X}$, with a constant capacity ℓ and a set of points P, compute the largest cardinality set $Q \subseteq P$ such that each region in $\mathcal{X}$ contains at most ℓ points of Q.

Ene et al., [16] gave $O(1)$-approximation algorithms for disks in plane with arbitrary capacities. For unit capacities, they show a PTAS for halfspaces in $\mathbb{R}^3$ and disks in the plane. To the best of our knowledge, there is no known $O(1)$-approximation algorithm for pseudodisks, even for unit capacity (standard pack points). We show that the problem admits a PTAS when the regions have unit capacity. In Section 8 we prove the following theorem.

▶ **Theorem 6.** *The Local search algorithm yields a PTAS for the Capacitated Point Packing problem for non-piercing regions when the regions have unit capacity.*

We define the notion of *lens bypassing*, and use this to build graphs with a small separator for the Set Cover and the Dominating Set problems thus obtaining a PTAS for these problems. Lens-bypassing is a finer tool than *lens-cutting* used in [27], and allows us to simplify one intersection at a time, instead of all intersections with one region at a time. This technique may find applications elsewhere. For the Capacitated Region Packing problem, we argue that the natural intersection graph relevant to the problem has a small separator by comparing it to a planar graph on the points.

5 Lens Bypassing

For two regions A and B, each connected component of $A \cap B$ bounded by two arcs, one from the boundary of A and the other from that of B is called a *lens*. Since the boundary of any region is oriented counter-clockwise, observe that the arcs from A and B forming the boundary of a lens are oriented in opposite directions.

Let L_{AB} denote the set of lenses formed by the intersection of regions A and B. We define a *lens bypassing* for a lens $\ell_{AB} \in L_{AB}$ formed by A and B as follows: Leaving B as is, we modify the boundary of A to follow the boundary of B along the arc of B bounding ℓ_{AB}, at an arbitrarily small distance $\beta > 0$ away from this arc. In this case, we say that we do lens bypassing in favor of B. If we did the reverse, we would call this lens bypassing in favor of A. More formally, let D_β be a ball of radius β. Then, bypassing lens ℓ_{AB} in favor of B is the operation of replacing A by $A' = A \setminus (\ell_{AB} \oplus D_\beta)$, where $\oplus$ denotes a Minkowski-sum. Figures 1 and 2 shows the operation of lens-bypassing for non-piercing regions.

▶ Remark. Our goal in lens-bypassing is to preserve the union of the regions while simplifying the arrangement. For this, we need $\beta = 0$ in the definition of lens-bypassing. But, this would

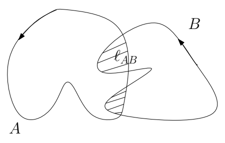


Figure 1 The figure shows lenses in the arrangement of non-piercing regions.

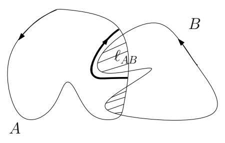


Figure 2 The figure shows the operation of bypassing lens ℓ_{AB} in favor of B

imply that A' and B share a portion of their boundaries and they are no longer properly intersecting. To avoid this technical complication, we take β to be an arbitrarily small positive quantity. However, for the rest of the paper, we do not make this distinction and state our results as if $\beta = 0$ for better readability. For instance, we say that lens-bypassing does not change the union of the regions. This is to be understood as: $A' \cup B$ contains all points in $A \cup B$ that are at least a distance $\delta > 0$ away from $\partial(A)$ and $\partial(B)$. Here, we assume that $\beta < \delta$ for an arbitrarily small positive quantity δ.

▶ **Proposition 6.1.** *Let A and B be two regions in $\mathbb{R}^2$. Let ℓ_{AB} be a lens formed by A and B. Let A', B be the regions obtained by bypassing the lens ℓ_{AB} in favor of B. Then, A' and B are non-piercing regions if and only if A, B are non-piercing.*

For two regions A and B, let $X(A, B)$ denote the intersection points of $\partial(A)$ and $\partial(B)$. Let σ_{AB} denote the cyclic sequence of $X(A, B)$ along $\partial(A)$, i.e., walking in counter-clockwise order along $\partial(A)$. Similarly, let σ_{BA} denote the cyclic sequence of the intersection points $X(A, B)$ along $\partial(B)$. For two points x, y on ∂A, we let $\gamma_{xy}(A)$ denote the arc on $\partial(A)$ from x to y in counter-clockwise direction along ∂A. We use γ_{xy}, when the region A is clear from context. Two cyclic sequences σ and σ' on the same set of elements are said to be reverse-cyclic if σ can be obtained from σ' by reversing the order. For example x_1, x_2, x_3, x_4, x_1 and x_4, x_3, x_2, x_1, x_4 are reverse-cyclic. For a cyclic sequence σ, we say x precedes y in σ, or $x \prec_\sigma y$ if x immediately precedes y in the cyclic sequence σ.

For a pair of reverse-cyclic sequences σ, σ' on the same set of elements, we define a *lens* in the sequences as pair of elements x, y that appear consecutively in σ and appear consecutively in σ', but in reverse-cyclic order; i.e., $x \prec_\sigma y$ and $y \prec_{\sigma'} x$. Bypassing a lens xy in a pair σ, σ' of reverse-cyclic sequences is the operation of removing x and y from σ and σ', i.e., $\pi = \sigma \setminus \{x, y\}$ and $\pi' = \sigma' \setminus \{x, y\}$.

For regions A, B consider the cyclic sequences σ_{AB} and σ_{BA}. If x, y form a lens in the sequences σ_{AB} and σ_{BA}, it is easy to see that the arcs of $\partial(A)$ and $\partial(B)$ between points x and y in $X(A, B)$ form a lens of the regions A and B. If $x \prec_{\sigma_{AB}} y$ and $x \prec_{\sigma_{BA}} y$, then the region bounded by the arcs of $\partial(A)$ and $\partial(B)$ between points x and y forms a region that is contained in either $A \setminus B$ or $B \setminus A$.

We will see that the pair of cyclic sequences σ_{AB} and σ_{BA} of the intersection points of regions A and B are reverse-cyclic if and only if the regions A, B are non-piercing. Further, a lens in the sequences σ_{AB} and σ_{BA} corresponds to a lens in the intersection of A and B, and bypassing a lens in σ_{AB} and σ_{BA} is the operation of lens bypassing in A and B. The following proposition is intuitively clear, and we skip the easy proof.

▶ **Proposition 6.2.** *If σ, σ' are two reverse-cyclic sequences on the same set X of elements, then*

1. *For a lens in the sequences σ, σ' formed by elements x, y, bypassing the lens leaves the sequences reverse-cyclic, i.e., the sequences $\pi = \sigma \setminus \{x, y\}$ and $\pi' = \sigma' \setminus \{x, y\}$, are reverse-cyclic.*

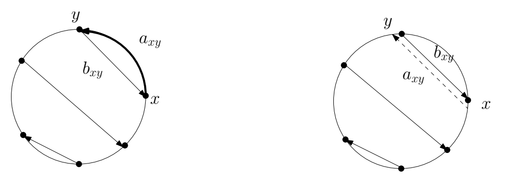


Figure 3 The figure on the left shows the segments of the boundary of B in A. The figure on the right shows the lens bypassed to obtain the modified region for A. This is shown as a dotted line.

2. *π and π' are reverse-cyclic sequences, where π and π' are obtained from σ and σ' respectively, by adding elements $x, y \notin X$ between the same pair of consecutive elements in σ and σ' such that $x \prec_{\pi} y$ and $y \prec_{\pi'} x$.*

We now give a combinatorial characterization of non-piercing regions that will be useful for the rest of the paper.

▶ **Theorem 7.** *Two regions A, B in $\mathbb{R}^2$ are non-piercing if and only if σ_{AB} and σ_{BA} are reverse-cyclic.*

Proof. We prove both directions by induction on $|X(A, B)|$. If A and B are non-piercing, then we show that σ_{AB} and σ_{BA} are reverse-cyclic. The base case is when $|X(A, B)| = 0$; sequences σ_{AB} and σ_{BA} are empty and and are therefore reverse-cyclic. Suppose the theorem holds for $|X(A, B)| < k$. Given two regions A, B with $|X(A, B)| = k$. Consider $S = \partial(B) \cap A$. Since $\partial(B)$ is not self-intersecting, S is a set of non-intersecting chords connecting disjoint pairs of points in $X(A, B)$.

Let $\gamma_{yx}(B)$ be a chord from y to x in S of smallest length, where the length of a chord in S joining x to y is the number of points of $X(A, B)$ encountered when going from y to x in σ_{AB}. We claim that x and y are adjacent in σ_{AB} since points between y and x along σ_{AB} can not be connected by a chord of S to a point outside without intersecting $\gamma_{yx}(B)$, and all such chords have smaller length than $\gamma_{yx}(B)$, contradicting the fact that $\gamma_{yx}(B)$ is the chord of smallest length.

Suppose $x \prec_{\sigma_{AB}} y$. Let $\gamma_{xy}(A)$ be the arc on $\partial(A)$ joining x to y. Since $\gamma_{yx}(B)$ is a chord joining y to x, it follows that $y \prec_{\sigma_{BA}} x$. Then, the region bounded by $\gamma_{yx}(B)$ and $\gamma_{xy}(A)$ forms a lens ℓ_{AB}. Suppose we bypass ℓ_{AB} in favor of B then, by Proposition 6.1 regions A and B remain non-piercing and $|X(A, B)|$ decreases by 2. Figure 3 shows this operation. By the inductive hypothesis, the two sequences are reverse-cyclic. Adding this pair x, y of adjacent points in both sequences σ_{AB} and σ_{BA} (in opposite order), the sequences remain reverse-cyclic by Proposition 6.2.

If $y \prec_{\sigma_{AB}} x$. Let a_{yx} be the arc joining y to x on $\partial(A)$. Since $\gamma_{yx}(B)$ lies in A, the region R bounded by a_{yx} and $\gamma_{yx}(B)$ lies in $A \setminus B$. In this case, we show $|X(A, B)| = 2$. Suppose not. Then, there are points u, v different from x, y in $X(A, B)$. If b_{uv} is a chord of S joining u to v. Then, around this region, we again obtain a region of $A \setminus B$ disconnected from R, contradicting the fact that $A \setminus B$ is connected. If $|X(A, B)| = 2$, then the sequences are reverse-cyclic.

We show the reverse direction again by induction on $|X(A, B)|$. If $|X(A, B)| = 0$, the regions are disjoint and are therefore non-piercing. Suppose that the theorem is true for $|X(A, B)| < k$. Let A, B be two regions with $|X(A, B)| = k$ and such that σ_{AB} and σ_{BA} are reverse-cyclic. By applying the lens-bypassing as before between two points x, y such that $x \prec_{\sigma_{AB}} y$ (in σ_{BA}, $y \prec_{\sigma_{BA}} x$, by virtue of the sequences being reverse-cyclic) the sequences remain reverse-cyclic by Proposition 6.2 and $|X(A, B)|$ has reduced by 2. By the inductive hypothesis, A and B are non-piercing. Adding the two intersection points x, y results in adding the lens between x and y, and this does not change $A \setminus B$ or $B \setminus A$, and the regions are non-piercing. ◀

▶ **Corollary 8.** *For two non-piercing regions A, B if there exist $x \prec_{\sigma_{AB}} y$ and $x \prec_{\sigma_{BA}} y$, then $|X(A, B)| = 2$.*

▶ **Corollary 9.** *For two non-piercing regions A, B, such that one is not contained in the other, $A \cap B$ is a collection of disjoint lenses.*

The lenses formed in an arrangement of a set $\mathcal{X}$ of regions can be ordered as a partial order by inclusion. That is, lenses $\ell_{AB} \prec \ell_{CD}$ if $\ell_{AB} \subseteq \ell_{CD}$. Note that either C or D could be equal to A or B. Now we prove the key lemma about lenses.

▶ **Lemma 10.** *Let $\mathcal{X}$ be a set of non-piercing regions. Let ℓ_{AB} be a minimal lens, defined by regions $A, B \in \mathcal{X}$. Let A' be the region obtained from A by bypassing the lens ℓ_{AB} in favor of B. Then, the regions $\mathcal{X}' = (\mathcal{X} \setminus \{A\}) \cup \{A'\}$ is a set of non-piercing regions.*

Proof. Let x, y be the two vertices of the lens ℓ_{AB} so that the arcs bounding ℓ_{AB} are $\gamma_{xy}(A)$, and $\gamma_{yx}(B)$. Let C be any region intersecting A. We will show that after bypassing the lens ℓ_{AB} in favor of B, the modified region A' remains non-piercing with respect to C. In particular, we will show that $\sigma_{A'C}$ and $\sigma_{CA'}$ remain reverse-cyclic. Let S be the set of chords in ℓ_{AB} formed by $\partial(C)$.

Suppose there is a chord $\gamma_{pq}(C)$ in S, such that p and q both lie on the boundary of the lens ℓ_{AB} defined by B, i.e., on $\gamma_{yx}(B)$. If q precedes p along $\partial(B)$, i.e., on σ_{BC} then $\gamma_{pq}(C)$ and $\gamma_{qp}(B)$ form a lens contained in ℓ_{AB}, contradicting the minimality of ℓ_{AB}. Therefore, it follows that p must precede q along $\partial(B)$, i.e., on σ_{BC}. But, this implies that p precedes q in both reverse-cyclic sequences σ_{BC} and σ_{CB}. By Corollary 8 therefore, $\partial(B)$ and $\partial(C)$ do not have any other points of intersection. In particular, this implies that $\partial(C)$ does not intersect the boundary of the lens ℓ_{AB} at any point other than p or q and so there are no other chords in S. After lens-bypassing therefore, $\sigma_{A'C}$ and $\sigma_{CA'}$ are obtained by inserting consecutive points p, q (in fact by points p', q' arbitrarily close to p and q respectively, but we do not make this distinction) between the same two consecutive points in opposite order into σ_{AC} and σ_{CA}. By Proposition 6.2 the sequences remain reverse-cyclic. This case is shown in Figure 4.

Now, suppose there is a chord $\gamma_{pq}(C)$ in S that joins two points on the boundary of the ℓ_{AB} defined by $\partial(A)$, namely $\gamma_{xy}(A)$. If q precedes p on $\gamma_{xy}(A)$, then this forms a lens contained in ℓ_{AB} contradicting the minimality of ℓ_{AB}. Otherwise, p precedes q in σ_{AC}. But this implies that p precedes q in both reverse-cyclic sequences σ_{AC} and σ_{CA}. Hence, by Corollary 8 $\partial(A)$ and $\partial(C)$ have no more points of intersection. In this case, after lens-bypassing, $\partial(A')$ and $\partial(C)$ do not intersect. Hence, $\sigma_{A'C}$ and $\sigma_{CA'}$ are empty and therefore the regions remain non-piercing. In fact, in this case we will have $A' \subseteq C$. This case is shown in Figure 5.

If none of the above hold, then all chords in S have one end-point on $\gamma_{xy}(A)$ and the other end-point on $\gamma_{yx}(B)$. In this case, after lens-bypassing $\sigma_{A'C}$ and $\sigma_{CA'}$ are obtained by replacing for each chord of C, the end-point of the chord on $\gamma_{xy}(A)$ by its other end-point on $\gamma_{yx}(B)$. The sequences σ_{AC} and $\sigma_{A'C}$ are identical except for renaming of the points in $\gamma_{xy}(A)$ by corresponding points in $\gamma_{yx}(B)$. The same holds for the sequence σ_{CA} and $\sigma_{CA'}$. Therefore the sequences $\sigma_{A'C}$ and $\sigma_{CA'}$ remain reverse-cyclic. This case is shown in Figure 6. ◀

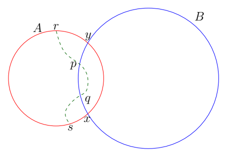


Figure 4 There is a chord of C between two points on $\gamma_{yx}(B)$.

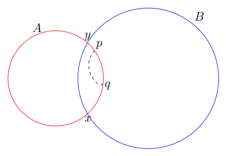


Figure 5 There is a chord of C between two points on $\gamma_{xy}(A)$.

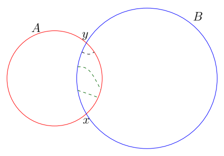


Figure 6 All chords of C connect a point on $\gamma_{xy}(A)$ and a point $\gamma_{yx}(B)$.

▶ **Corollary 11.** *Let ℓ_{AB} be a minimal lens. Let A' be the region obtained after bypassing ℓ_{AB} in favor of B. Then, for any region $C \neq B$, $A \cap C \neq \emptyset$ implies $A' \cap C \neq \emptyset$.*

6 Dominating Set and Set Cover

In this section, we first describe the construction of a graph over a set of non-piercing regions. The graph we construct satisfies the conditions required for Theorem 2, and thereby obtaining a PTAS for the Dominating Set and Set Cover problems.

For the Set Cover problem, in order to satisfy the local-exchange property, we require a bipartite graph $H = (\mathcal{R} \cup \mathcal{B}, E)$ with a small separator so that for any set $\mathcal{S} \subseteq \mathcal{B}$, the set $\mathcal{B}' = (\mathcal{B} \setminus \mathcal{S}) \cup N(\mathcal{S})$ is a Set Cover, where $N(\mathcal{S})$ is the set of neighbors of the vertices in H corresponding to regions in $\mathcal{S}$. In other words, the required graph $H = (\mathcal{R} \cup \mathcal{B}, E)$ must satisfy the following property: For each point $p \in P$, there is an $R \in \textsc{Red}(p)$ and $B \in \textsc{Blue}(p)$ that are adjacent in H, where $\textsc{Red}(p)$ is the set of regions in $\mathcal{R}$ containing p, and $\textsc{Blue}(p)$ is the set of regions in $\mathcal{B}$ containing p.

For the Dominating Set problem, the property that the bipartite graph H is required to satisfy is that for any region $X \in \mathcal{X}$, there is a region $R \in \textsc{Red}(X)$ and a region $B \in \textsc{Blue}(X)$ such that R and B are adjacent in H, where $\textsc{Red}(X)$ is the set of regions in $\mathcal{R}$ intersecting X, and $\textsc{Blue}(X)$ is the set of regions in $\mathcal{B}$ intersecting X. We let $\mathcal{G}$ denote the regions of $\mathcal{X}$ that are not in $\mathcal{R}$ or $\mathcal{B}$. Since we assumed that $\mathcal{R} \cap \mathcal{B} = \emptyset$, the three sets form a partition of the input $\mathcal{X}$.

We give a graph construction of a planar graph for any set $\mathcal{X} = \mathcal{R} \sqcup \mathcal{B} \sqcup \mathcal{G}$ of non-piercing regions from which the construction of the graphs for the Dominating Set and Set Cover problems follow.

Henceforth we refer to the regions in $\mathcal{R}, \mathcal{B}$ and $\mathcal{G}$ as Red, Blue, and Green regions respectively. We call a region $X \in \mathcal{X}$ *bi-chromatic* if and only if X intersects a Red as well as a Blue region. We assume that a region intersects itself. Therefore a Red region is bi-chromatic if it intersects a Blue region. Similarly, a point in the plane is called bi-chromatic if and only if it is contained in both a Red as well as a Blue region. As before, we let $\textsc{Red}(X)$ and $\textsc{Blue}(X)$ denote the Red and Blue regions intersecting X, respectively. Similarly, for any point p, we let $\textsc{Red}(p)$ and $\textsc{Blue}(p)$ denote the Red and Blue regions containing p respectively. We now define the properties that our constructed graph satisfies.

▶ **Definition 12.** Let $\mathcal{X} = \mathcal{R} \sqcup \mathcal{G} \sqcup \mathcal{B}$ be a set of non-piercing regions. A bipartite planar graph $H = (\mathcal{R} \cup \mathcal{B}, E)$ is called a *locality-preserving* graph for $\mathcal{X}$ if it satisfies the following properties.

P1. For each bi-chromatic region $X \in \mathcal{X}$, there exists an $R \in \textsc{Red}(X)$ and a $B \in \textsc{Blue}(X)$ such that R and B are adjacent in H.

P2. For each bi-chromatic point p in the plane, there exists an $R \in \textsc{Red}(p)$ and a $B \in \textsc{Blue}(p)$ such that R and B are adjacent in H.

We can assume that in any instance, all Green regions are bi-chromatic since Green regions that are not bi-chromatic can be removed without changing the problem. We say that a graph H on $\mathcal{R} \cup \mathcal{B}$ *satisfies* a region $X \in \mathcal{X}$ if Property P1 holds for X. We also state this as "region X is *satisfied* by H". We use a similar terminology for the points. If in a given instance $\mathcal{X}$, there are regions $R \in \mathcal{R}, B \in \mathcal{B}$ and $G \in \mathcal{G}$ such that $R \cap B \cap G \neq \emptyset$, we say that $\mathcal{X}$ has a Red-Blue-Green intersection. The main theorem we prove in this section is the following:

▶ **Theorem 13.** *For any set $\mathcal{X} = \mathcal{R} \sqcup \mathcal{G} \sqcup \mathcal{B}$ of non-piercing regions, there is a locality-preserving graph H.*

The broad approach to construct a locality-preserving graph H for $\mathcal{X}$ is as follows. If the instance $\mathcal{X}$ satisfies certain additional conditions, then we can directly describe the construction of such a graph. If the instance $\mathcal{X}$ does not satisfy these additional conditions, we show that can *reduce* the instance to one that does.

In order to do this, we describe a sequence of reduction steps that either remove a region, or bypasses a minimal lens in the arrangement of the regions, thus getting us closer to an arrangement enjoying the additional conditions alluded to above. These reduction steps have the crucial property that if we are given a locality-preserving graph for the reduced instance, we can obtain a locality-preserving graph for the original instance. We start with a construction of a locality-preserving graph for an instance $\mathcal{X}$ satisfying the additional conditions. Then, in a sequence of lemmas, namely Lemma 15, 16 and 17 we describe the reduction steps for an instance not enjoying the additional properties. Finally, we can prove Theorem 13.

▶ **Lemma 14.** *Suppose $\mathcal{X} = \mathcal{R} \sqcup \mathcal{G} \sqcup \mathcal{B}$ is a set of non-piercing regions satisfying the following properties:*

1. *$R \cap R' = \emptyset$, for all $R, R' \in \mathcal{R}$ and $B \cap B' = \emptyset$, for all $B, B' \in \mathcal{B}$, i.e., the* Red *regions are pairwise disjoint, and the* Blue *regions are pairwise disjoint.*
2. *For each $R \in \mathcal{R}, B \in \mathcal{B}$ and $G \in \mathcal{G}$, $R \cap B \cap G = \emptyset$, i.e., there is no* Red-Blue-Green *intersection.*

Then, there is a locality-preserving graph for $\mathcal{X}$.

Proof. In order to construct the graph, we temporarily add Red and Blue points to the arrangement of the regions in the following way: For each intersection $R \cap G$ of a Red region R and a Green region G, we place a Red point in $R \cap G$. Similarly, we place a Blue point for each Blue-Green intersection. Since there are no Red-Blue-Green intersections, observe that in the interior of any Green region the Red and Blue regions are disjoint. Therefore, for any Green region, the point we place corresponding to a Red region does not lie in a Blue region, and vice-versa.

Now, by Lemma 1, applied to the Green regions and the Red and Blue points we place, there is a plane graph K such that for each Green region $G \in \mathcal{G}$, there is an edge in K between a Red point contained in G and a Blue point contained in G lying entirely in G.

For each $G \in \mathcal{G}$, we pick one such edge e_G arbitrarily. Let r and b be the Red and Blue end-points of e_G, respectively. As observed earlier, r lies in a Red region, and b lies in a Blue region. So, walking from b to r along e_G we encounter a Red region R and a Blue region B that are consecutive along e_G. We now extend B along e_G so that R and B now intersect. Note that this can be done in a way that they remain non-piercing. An example is shown in Figures 7 and 8.

We remove any Green region with a Red-Blue-Green intersection, and repeat the operation above on the remaining Green regions. Since the graph K is planar, the extended

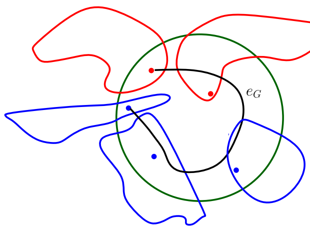


Figure 7 The edge e_G.

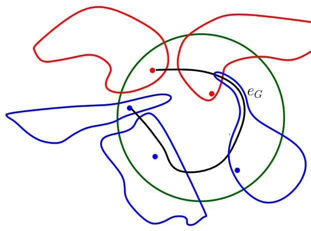


Figure 8 Extending the BLUE region to intersect the adjacent RED region along e_G.

BLUE regions remain disjoint. Extending a BLUE region along an edge of K chosen for a GREEN region may intersect other GREEN regions in an arbitrary fashion. However, this does not matter as the GREEN regions will not play any role henceforth. It is possible that the same BLUE region is extended multiple times to intersect the same RED region. However, the regions remain non-piercing as each extension of the BLUE ensures this property.

By extending the BLUE regions, we have the property that for each $G \in \mathcal{G}$, there is now an $R \in \text{RED}(G)$ and $B \in \text{BLUE}(G)$ such that $R \cap B \neq \emptyset$. Therefore, the intersection graph of the RED and BLUE regions gives us the desired locality-preserving graph.

Since we ensure that the RED regions are pairwise disjoint, the BLUE regions are pairwise disjoint, and $\mathcal{R} \cup \mathcal{B}$ is non-piercing, the intersection graph of $\mathcal{R} \cup \mathcal{B}$ is planar as observed by Chan and Har-Peled [12]. ◀

We now describe the reductions for an instance $\mathcal{X}$ that does not satisfy the conditions of Lemma 14. The reduction steps are the following: We first show that we can remove RED-BLUE-GREEN intersections if any in our instance. Then, we show that if our instance has two regions such that one is contained in another, then we can remove one of them (this statement is not entirely accurate; we do not get rid of containments where a RED or BLUE region is contained in a GREEN region, but such containments do not affect our construction). Then, we show that we can decrease the number vertices in the arrangement by bypassing minimal lenses. The latter two reductions are applied repeatedly until none apply. At that point, we can show that the instance satisfies the conditions of Lemma 14 and a locality-preserving graph can thus be constructed.

▶ **Lemma 15.** *Let $\mathcal{X} = \mathcal{R} \sqcup \mathcal{B} \sqcup \mathcal{G}$ be a set of non-piercing regions. Suppose there exists $R \in \mathcal{R}, B \in \mathcal{B}$ and $G \in \mathcal{G}$ such that $R \cap B \cap G \neq \emptyset$, i.e., the instance contains a* RED-BLUE-GREEN *intersection. Then, a locality-preserving graph for the reduced instance $\mathcal{X}' = \mathcal{X} \setminus G$ is a locality-preserving graph for $\mathcal{X}$.*

Proof. Let H' be a locality-preserving graph for $\mathcal{X}'$. By Property P2, there is an edge between a region $R \in \text{RED}(p)$ and $B \in \text{BLUE}(p)$ for $p \in R \cap B \cap G$. This implies that H' is locality-preserving for $\mathcal{X}$ since $\text{RED}(p) \subseteq \text{RED}(G)$ and $\text{BLUE}(p) \subseteq \text{BLUE}(G)$. ◀

▶ **Lemma 16.** *Let $\mathcal{X} = \mathcal{R} \sqcup \mathcal{B} \sqcup \mathcal{G}$ be a set of non-piercing regions with no* RED-BLUE-GREEN *intersections. If there are two regions P and Q such that $P \subseteq Q$, and either both P and Q are* GREEN*, or both are not* GREEN*, a locality-preserving graph for a suitable reduced instance $\mathcal{X}'$ with one less region than $\mathcal{X}$ implies a locality-preserving graph for $\mathcal{X}$.*

Proof. Note that, as mentioned in the conditions of the lemma, we do not deal with the case where Q is GREEN and P is either RED or BLUE. Thus, we have the following cases.

Case 1: Q is either Red or Blue, and P is Green. This case does not arise, since we assume that there are no Red-Blue-Green intersections and all Green regions are bi-chromatic.

Case 2: Q is either Red or Blue, and P is either Red or Blue. The reduced instance in this case is obtained by removing P, i.e., $\mathcal{X}' = \mathcal{X} \setminus P$. If H' is a locality-preserving graph for $\mathcal{X}'$, then we obtain the locality-preserving graph H for $\mathcal{X}$ by adding an edge between P and Q if they have distinct colors. Otherwise, we add P to any Red or Blue region intersecting P and having a color different from P, if such a region exists. If such a region does not exist, then we set $H = H'$.

Now we argue that the graph H so constructed is locality-preserving for $\mathcal{X}$. In obtaining the graph H, we added at most one vertex of degree 1 to H', and hence H is planar. Since the only regions affected by the removal of P are those that intersect P, we only argue about such regions. Similarly the only points affected are those lying in P and we argue only about these points.

Case 2a: P and Q have the same color. To show that H is locality-preserving for $\mathcal{X}$, note that if P is bi-chromatic, then the edge we added satisfies P. Any other region intersecting P also intersects a region of the same color as P, namely Q. Therefore, such a region is satisfied by H'. Similarly, all bi-chromatic points in P remain bi-chromatic in $\mathcal{X}'$ since they are contained in Q. Therefore, it follows that the edges in H' satisfy all bi-chromatic points with respect to $\mathcal{X}$.

Case 2b: P and Q have different colors. In this case, we connect P to Q. This satisfies the region P. Since $P \subset Q$, all other regions intersecting P also intersect Q and are therefore satisfied by the edge we added. By the same argument, all bi-chromatic points in P are also satisfied by H.

Case 3: P and Q are both Green. In this case, $\mathcal{X}' = \mathcal{X} \setminus Q$ is the desired reduced instance. If H' is a locality-preserving graph, then H' is also locality-preserving with respect to $\mathcal{X}$, since any region intersecting P also intersects Q. ◀

▶ **Lemma 17.** *Let $\mathcal{X} = \mathcal{R} \sqcup \mathcal{B} \sqcup \mathcal{G}$ be a set of non-piercing regions such that there are no* Red-Blue-Green *intersections. Then, a locality-preserving graph H' for a reduced instance $\mathcal{X}'$ is a locality-preserving graph for $\mathcal{X}$. Here, $\mathcal{X}'$ is obtained by bypassing a minimal lens ℓ_{PQ} using the following rules:*

1. *If P and Q have the same color. Bypass ℓ_{PQ} in favor of either P or Q chosen arbitrarily.*
2. *If ℓ_{PQ} is contained in a region $R \in \mathcal{R}$, where R is distinct from P and Q, then bypass the lens in favor of the region that is not* Red.
3. *If ℓ_{PQ} is contained in a region $B \in \mathcal{B}$, where B is distinct from P and Q, then bypass the lens in favor of the region that is not* Blue.

Proof. We assume without loss of generality that we bypass the lens ℓ_{PQ} in favor of Q, and let P' be the resulting region corresponding to P. Then, $\mathcal{X}' = (\mathcal{X} \setminus P) \cup P'$. By Lemma 10, the regions in $\mathcal{X}'$ are non-piercing since we bypass a minimal lens ℓ_{PQ}.

Suppose P and Q have the same color. By Corollary 11, any region $X \neq Q$ intersecting P also intersects P'. This ensures that the set of Red or Blue regions intersecting a region $X \neq Q$ remains unchanged in $\mathcal{X}'$. Thus, H' satisfies all regions except possibly Q. Since P and Q have the same color, Q remains bi-chromatic in $\mathcal{X}'$ if it was bi-chromatic in $\mathcal{X}$. Since

$P \cup Q$ remains unchanged due to lens-bypassing and the fact that they have the same color, all bi-chromatic points in $\mathcal{X}$ remain bi-chromatic in $\mathcal{X}'$. Since no region gains a new intersection and no point is contained in a new region when going from $\mathcal{X}$ to $\mathcal{X}'$, a locality-preserving graph H' for $\mathcal{X}'$ is also locality-preserving for $\mathcal{X}$.

Now suppose that P and Q have distinct colors, then we bypass ℓ_{PQ} only if it lies in a RED or a BLUE region. Let us assume that ℓ_{PQ} is contained in a RED region R. The other case is symmetric. By our assumption that there is no RED-BLUE-GREEN intersection, one of P or Q must be RED. Since we assume that we bypass ℓ_{PQ} in favor of Q, P is RED. We claim that H' is locality-preserving with respect to $\mathcal{X}$.

By Corollary 11, it follows that for every region $X \neq P, Q$, the set of regions intersecting X does not change. These regions are therefore satisfied by H'. For the region Q, while it possibly loses its intersection with P, it continues to intersect a RED region, namely R. Therefore, Q is also satisfied by H'. The region P' may not be bi-chromatic even if P was bi-chromatic. This can happen only when Q is the unique BLUE region intersecting P, which is RED by assumption. However, in that case, consider any point $p \in \ell_{PQ}$. Such a point p is bi-chromatic in $\mathcal{X}'$ since it lies in Q and R. Therefore the point p is satisfied in H'. Now, the fact that P is satisfied in H' follows from the fact that any region containing p also intersects P. ◀

Proof of Theorem 13. Given an instance $\mathcal{X} = \mathcal{R} \sqcup \mathcal{B} \sqcup \mathcal{G}$ of non-piercing regions, we first remove all RED-BLUE-GREEN intersections by applying Lemma 15. Then, we repeatedly apply Lemma 16 followed by Lemma 17 until neither applies. At this point, we claim that the RED regions are pairwise disjoint, and the BLUE regions are pairwise disjoint.

First, note that no RED or BLUE region is contained in another RED or BLUE region by Lemma 16. In particular, this implies that any intersection of a pair P, Q of RED regions is a union of lenses formed by them. However, any such lens ℓ_{PQ} is removed by Lemma 17 so long as it is minimal. We argue that ℓ_{PQ} must be minimal. Suppose not. Then, there is a minimal lens ℓ_{WZ} contained in ℓ_{PQ}. We can check that for all possible colors of W and Z, Lemma 17 applies to ℓ_{WZ} and is thus bypassed. A symmetric argument applies for the BLUE regions.

Thus, when the conditions of Lemma 16 or Lemma 17 do not apply, the RED regions are pairwise disjoint, the BLUE regions are pairwise disjoint, and there are no RED-BLUE-GREEN intersections. Hence, the conditions of Lemma 14 apply and we can obtain a locality-preserving graph H'. This implies a locality-preserving graph H for the instance $\mathcal{X}$. ◀

Now, we can prove that the Dominating Set and Set Cover problem for non-piercing regions admits a PTAS.

Proof of Theorem 4. Let $\mathcal{R}$ denote an optimal solution to the Dominating Set problem. Let $\mathcal{B}$ denote a solution returned by local search. Let $\mathcal{G}$ be the remaining regions. Recall that we assume $\mathcal{R} \cap \mathcal{B} = \emptyset$. Let $\mathcal{X} = \mathcal{R} \sqcup \mathcal{B} \sqcup \mathcal{G}$. Note that since every region in $\mathcal{X}$ is bi-chromatic, Property P1 is precisely the locality condition required for the graph of the Dominating Set problem. Now, Theorem 13 gives us a planar graph satisfying Property P1, and is therefore the desired graph. Since planar graphs have separators of size $O(\sqrt{n})$ [25], where n is the number of regions in the input $\mathcal{X}$, by Theorem 2 a $(1+\epsilon)$-approximation algorithm follows. ◀

Proof of Theorem 3. Let $\mathcal{R}$ denote an optimal solution to the Set Cover problem. Let $\mathcal{B}$ denote a solution returned by local search. Recall that we assume $\mathcal{R} \cap \mathcal{B} = \emptyset$. Let $\mathcal{X} = \mathcal{R} \cup \mathcal{B}$, and let $\mathcal{G} = \emptyset$. Since every point P is in $R \cap B$, for some $R \in \mathcal{R}$ and $B \in \mathcal{B}$, Property P2 is precisely the locality condition required for the Set Cover problem. Now, Theorem 13 gives us a planar graph satisfying the required locality conditions. The graph returned by Theorem 13 is planar. Since planar graphs have separators of size $O(\sqrt{n})$ [25], where n is the number of regions in the input $\mathcal{X}$, by Theorem 2 a $(1+\epsilon)$-approximation follows. ◀

7 Capacitated Region Packing

Recall that in the Capacitated Region Packing problem, we are given a family of r-admissible regions $\mathcal{X}$, a set of points P, and a positive integer constant ℓ. We want to find a maximum sized subset $\mathcal{X}' \subseteq \mathcal{X}$ such that every point $p \in P$ is contained in at most ℓ regions in $\mathcal{X}'$. Unlike the earlier results in this paper, here we require that the regions be r-admissible for a constant r, i.e., they are non-piercing and their boundaries intersect at most a constant number of times.

Let $\mathcal{X} = \mathcal{R} \cup \mathcal{B}$ and $k = 2\ell$. Note that the depth of a point in P with respect to $\mathcal{X}$, that is the number of regions in $\mathcal{X}$ containing it, is at most k. For this problem, the graph we construct is as follows: For each point $p \in P$, add an edge between all regions in $\mathcal{R}$ containing p and all regions in $\mathcal{B}$ containing p. It is easy to check that this graph satisfies the local-exchange property as stated in Theorem 2. We now show that this graph has a small balanced separator. In fact, we show the following super-graph $H(\mathcal{X}, k)$ has a small, balanced separator. For each point $p \in \mathbb{R}^2$ whose depth is at most k, add an edge between all pairs of regions in $\mathcal{R} \cup \mathcal{B}$ containing p.

▶ **Theorem 18.** *Given a set $\mathcal{X}$ of r-admissible regions for a constant r, the graph $H(\mathcal{X}, k)$ on $\mathcal{X}$ has a balanced separator of size $O(k^{3/2}\sqrt{|\mathcal{X}|})$.*

Proof. In order to prove the statement for $H(\mathcal{X}, k)$, we prove it for an isomorphic graph that is the intersection graph of a family of trees $\mathcal{T}$ that we obtain in the following way: We put one point in every cell whose depth is at most k in the arrangement of the regions in $\mathcal{X}$, and call this point set P. By Lemma 1 there exists a plane graph G_P on the point set P such that the subgraph induced by $X \cap P$ is connected, for every $X \in \mathcal{X}$. For each $X \in \mathcal{X}$, consider an arbitrary spanning tree T_X of the subgraph induced by $X \cap P$. Observe that the intersection graph of the family of these spanning trees $\mathcal{T} = \{T_X \mid X \in \mathcal{X}\}$ is isomorphic to $H(\mathcal{X}, k)$. We now claim that such an intersection graph has a small and balanced separator.

▶ **Lemma 19.** *Given a family of trees $\mathcal{T}$ as above, its intersection graph has a balanced separator of size $O(k^{3/2}\sqrt{|\mathcal{T}|})$.*

Proof. We prove this statement as follows. We assign appropriate weights on the points in P, and use the fact that the graph G_P on the points, constructed by applying Lemma 1 is planar, and therefore has a balanced weighted separator.

We assign weights to the points in P as follows: We start by assigning a weight of 0 to each point in P. For each region $X \in \mathcal{X}$, we add $1/|X \cap P|$ to the weight of each point in X. Thus, the weight of a point in P is given by $wt(p) = \sum_{X \mid p \in X} 1/|X \cap P|$.

By the Lipton-Tarjan separator theorem [25], G_P has a separator S of size $O(\sqrt{|P|})$ such that removing S separates the graph into two disjoint sets A and B, each of which has at most 2/3 of the total weight of the points. The separator S for G_P gives a separator for the graph $\mathcal{T}$ by taking $\mathcal{S} = \{T \mid S \cap T \neq \emptyset\}$. We claim that $\mathcal{S}$ is a small, balanced separator. The fact that removing $\mathcal{S}$ separates $\mathcal{T}$ into two parts $\mathcal{A}, \mathcal{B}$ follows since a tree containing a vertex from A and a vertex from B must contain a vertex from S.

To show that $|\mathcal{S}| \leq O(k^{3/2}\sqrt{|\mathcal{T}|})$, we proceed as follows. By our construction, every point in P has depth at most k. This implies that $|\mathcal{S}| \leq k|S|$. However, $|S| \leq O(\sqrt{|P|})$, as S is a separator in G_P. Since r-admissible regions have linear union complexity [32], the Clarkson-Shor technique [26, p. 141] implies that the number of cells in the arrangement is at most $O(k|\mathcal{T}|)$. Thus, $|P| = O(k|\mathcal{T}|)$ and hence, $|\mathcal{S}| \leq O(k\sqrt{k|\mathcal{T}|})$.

Now, we need to show that $\mathcal{S}$ is balanced. To see this, let $\mathcal{T}_A$ be the set of trees whose vertex set is a subset of A. $\mathcal{T}_B$ is defined similarly. Since, the weight of all trees in $\mathcal{T}_A$ were distributed among the points in A, $wt(A) \geq |\mathcal{T}_A|$. Also, from the planar separator theorem we know that $wt(A) \leq \frac{2}{3}|\mathcal{T}|$. Therefore, $|\mathcal{T}_A| \leq \frac{2}{3}|\mathcal{T}|$. The same holds for $\mathcal{T}_B$. Therefore, $\mathcal{S}$ is a balanced separator of size $O(k^{3/2}\sqrt{|\mathcal{T}|})$ of the intersection graph of $\mathcal{T}$. ◀

From Lemma 19, it follows that $H(\mathcal{X}, k)$ has a balanced separator of size $O(k^{3/2}\sqrt{|\mathcal{X}|})$. ◀

Proof of Theorem 5. The graph constructed satisfies the conditions of Theorem 2. Therefore a PTAS follows. ◀

8 Capacitated Point Packing

Recall that in the Capacitated Point Packing problem, we are given a set P of n points, a set $\mathcal{X}$ of non-piercing regions, and a positive integer constant ℓ. The goal is to obtain the maximum sized subset of points $Q \subseteq P$ such that for every region $X \in \mathcal{X}$, $|X \cap Q| \leq \ell$. We consider the Capacitated Point Packing problem when $\ell = 1$. We show that the Local search algorithm yields a PTAS for this special case.

Proof of Theorem 6. We construct a bipartite graph on the union of red and blue points in the following way. We put an edge between a red point and a blue point if they are contained in some region $X \in \mathcal{X}$. It is easy to check that this graph satisfies the local-exchange property as stated in Theorem 2. As both the local search and the optimum solutions are feasible, every region contains at most one red point and at most one blue point. Observe that the graph constructed in Lemma 1 is such a graph, because it ensures that the graph induced by the points contained in an input region is connected. This in turn means that there is always an edge between a red point and a blue point contained in the same region. Lemma 1 states that this graph is planar. Thus, the graph constructed satisfies the condition of Theorem 2. Therefore, a PTAS follows. ◀

References

1 Anna Adamaszek and Andreas Wiese. Approximation schemes for maximum weight independent set of rectangles. In *Proceedings of the 2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, FOCS'13, pages 400–409, Washington, DC, USA, 2013. IEEE Computer Society.

2 Anna Adamaszek and Andreas Wiese. A QPTAS for maximum weight independent set of polygons with polylogarithmically many vertices. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'14, pages 645–656, 2014.

3 Pankaj K. Agarwal and Nabil H. Mustafa. Independent set of intersection graphs of convex objects in 2D. *Computational Geometry*, 34(2):83–95, 2006.

4 Pankaj K. Agarwal, Marc van Kreveld, and Subhash Suri. Label placement by maximum independent set in rectangles. *Computational Geometry*, 11(3):209–218, 1998.

5 Rom Aschner, Matthew J. Katz, Gila Morgenstern, and Yelena Yuditsky. Approximation schemes for covering and packing. In *WALCOM: Algorithms and Computation*, volume 7748 of *Lecture Notes in Computer Science*, pages 89–100. Springer Berlin Heidelberg, 2013.

6 Franz Aurenhammer. Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, 23(3):345–405, 1991.

7 Reuven Bar-Yehuda, Danny Hermelin, and Dror Rawitz. Minimum vertex cover in rectangle graphs. *Computational Geometry*, 44(6):356–364, 2011.

8 Vijay V. S. P. Bhattiprolu and Sariel Har-Peled. Separating a Voronoi Diagram via Local Search. In *Proceedings of the Thirty-second International Symposium on Computational Geometry*, SoCG'16, pages 18:1–18:16, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

9 Hervé Brönnimann and Michael T. Goodrich. Almost optimal set covers in finite VC-dimension. *Discrete & Computational Geometry*, 14(1):463–479, 1995.

10 Timothy M. Chan. Polynomial-time approximation schemes for packing and piercing fat objects. *Journal of Algorithms*, 46(2):178–189, 2003.

11 Timothy M. Chan, Elyot Grant, Jochen Könemann, and Malcolm Sharpe. Weighted capacitated, priority, and geometric set cover via improved quasi-uniform sampling. In *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'12, pages 1576–1585, 2012.

12 Timothy M. Chan and Sariel Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. *Discrete & Computational Geometry*, 48(2):373–392, 2012.

13 Vincent Cohen-Addad and Claire Mathieu. Effectiveness of local search for geometric optimization. In *Proceedings of the Thirty-first International Symposium on Computational Geometry*, SoCG'15, pages 329–343, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

14 Jeffrey S. Doerschler and Herbert Freeman. A rule-based system for dense-map name placement. *Communications of the ACM*, 35(1):68–79, 1992.

15 Stephane Durocher and Robert Fraser. Duality for geometric set cover and geometric hitting set problems on pseudodisks. In *Proceedings of the 27th Canadian Conference on Computational Geometry*, pages 8–16, 2015.

16 Alina Ene, Sariel Har-Peled, and Benjamin Raichel. Geometric packing under non-uniform constraints. In *Proceedings of the Twenty-eighth Annual Symposium on Computational Geometry*, SoCG'12, pages 11–20, New York, NY, USA, 2012. ACM.

17 Thomas Erlebach and Erik Jan van Leeuwen. Approximating geometric coverage problems. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete algorithms*, SODA'08, pages 1267–1276, 2008.

18 Guy Even, Dror Rawitz, and Shimon (Moni) Shahar. Hitting sets when the VC-dimension is small. *Inf. Process. Lett.*, 95(2):358–362, July 2005.

19 Matt Gibson and Imran A. Pirwani. Algorithms for dominating set in disk graphs: Breaking the $\log n$ barrier. In *Algorithms – ESA 2010 - 18th Annual European Symposium, Liverpool, United Kingdom, September 6–8, 2010, Proceedings*, pages 243–254, 2010.

20 Sariel Har-Peled. Quasi-polynomial time approximation scheme for sparse subsets of polygons. In *Proceedings of the Thirtieth Annual Symposium on Computational Geometry*, SoCG'14, pages 120–129, New York, NY, USA, 2014. ACM.

21 Sariel Har-Peled and Kent Quanrud. Approximation algorithms for polynomial-expansion and low-density graphs. In *Algorithms – ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14–16, 2015, Proceedings*, pages 717–728, 2015.

22 Dorit S. Hochbaum and Wolfgang Maass. Fast approximation algorithms for a nonconvex covering problem. *Journal of algorithms*, 8(3):305–323, 1987.

23 Erik Krohn, Matt Gibson, Gaurav Kanade, and Kasturi Varadarajan. Guarding terrains via local search. *Journal of Computational Geometry*, 5(1):168–178, 2014.

24 Brian Lent, Arun Swami, and Jennifer Widom. Clustering association rules. In *Proceedings of the Thirteenth International Conference on Data Engineering*, pages 220–231. IEEE Computer Society, 1997.

25 Richard J. Lipton and Robert E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

26 Jiří Matoušek. *Lectures on Discrete Geometry*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

27 Nabil H. Mustafa, Rajiv Raman, and Saurabh Ray. Quasi-polynomial time approximation scheme for weighted geometric set cover on pseudodisks and halfspaces. *SIAM Journal on Computing*, 44(6):1650–1669, 2015.

28 Nabil H. Mustafa and Saurabh Ray. Improved results on geometric hitting set problems. *Discrete & Computational Geometry*, 44(4):883–895, 2010.

29 Evangelia Pyrga and Saurabh Ray. New existence proofs for ϵ-nets. In *Proceedings of the Twenty-fourth Annual Symposium on Computational Geometry*, SoCG'08, pages 199–207, New York, NY, USA, 2008. ACM.

30 Di Tian and Nicolas D. Georganas. A coverage-preserving node scheduling scheme for large wireless sensor networks. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 32–41. ACM, 2002.

31 Kasturi Varadarajan. Weighted geometric set cover via quasi-uniform sampling. In *Proceedings of the 42nd ACM Symposium on Theory of Computing*, pages 641–648, 2010.

32 Sue Whitesides and Rongyao Zhao. *k-admissible collections of Jordan curves and offsets of circular arc figures*. McGill University, School of Computer Science, 1990.

Incremental and Fully Dynamic Subgraph Connectivity For Emergency Planning*

Monika Henzinger[1] and Stefan Neumann[2]

1 **University of Vienna, Faculty of Computer Science, Vienna, Austria**
monika.henzinger@univie.ac.at

2 **University of Vienna, Faculty of Computer Science, Vienna, Austria**
stefan.neumann@univie.ac.at

Abstract

During the last 10 years it has become popular to study dynamic graph problems in a *emergency planning* or *sensitivity* setting: Instead of considering the general fully dynamic problem, we only have to process a *single* batch update of size d; after the update we have to answer queries.

In this paper, we consider the dynamic subgraph connectivity problem with sensitivity d: We are given a graph of which some vertices are activated and some are deactivated. After that we get a single update in which the states of up to d vertices are changed. Then we get a sequence of connectivity queries in the subgraph of activated vertices.

We present the first fully dynamic algorithm for this problem which has an update and query time only slightly worse than the best decremental algorithm. In addition, we present the first incremental algorithm which is tight with respect to the best known conditional lower bound; moreover, the algorithm is simple and we believe it is implementable and efficient in practice.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases connectivity, emergency planning, sensitivity

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.48

1 Introduction

Dynamic graph algorithms maintain a data structure to answer queries about certain properties of the graph while the underlying graph is changed, e.g., by vertex or edge deletions and additions; such properties could be, for example, the connectivity or the shortest paths between two vertices. The main goal is that after an update the algorithm does not have to recompute the data structure from scratch, but only has to make a small number of changes to it. Due to strong conditional lower bounds for various dynamic graph problems (see [1, 16, 22]), it is necessary to restrict the dynamic model in some way to improve the efficiency of the operations. One model that has become increasingly popular is to study dynamic graph problems in a *sensitivity* or *emergency planning* setting (see, e.g., [10, 23, 7, 8, 4, 3, 21]): Instead of considering the general fully dynamic problem in which we get a sequence of updates and queries, we only allow for a single batch update of size d after which we want to answer queries. Since we allow only a single update, the update and query times for such sensitivity problems are much faster than for the general fully dynamic problem.

* The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement no. 340506.

In this paper, we consider the subgraph connectivity problem with sensitivity d: We get a graph $G = (V, E)$ of which some vertices are *activated* and some are *deactivated* and we can preprocess it. There is a single update changing the states of up to d vertices. In the subsequent queries we need to answer if two given vertices are connected by a path which traverses only activated vertices. If the update can only active previously deactivated vertices, then an algorithm for this problem is called *incremental*; if it can only deactivate activated vertices, then it is *decremental*; if it can turn vertices on and off arbitrarily, then it is called *fully dynamic*.

The problem is of high practical interest as it models a scenario which is very relevant to infrastructure problems. For example, assume you are an internet service provider and you maintain many hubs which are connected to each other. In case of a defect, some of the hubs fail but there is a small number of *backup* hubs which can be used until the defect hubs are repaired in order to provide your services to your customers. Notice that in such a scenario it is likely that the number of backup hubs is much smaller than the number of regular hubs.

1.1 Our Contributions

We present the first incremental and fully dynamic algorithms for the subgraph connectivitiy problem with sensitivity d. The update and query times of our fully dynamic algorithm are only slightly slower than those of the best decremental algorithm for this problem. In addition, the incremental algorithm is essentially tight with respect to the best known conditional lower bound for this problem. Additionally, we contribute a characterization of the paths which are added to a graph when activating some nodes.

Our result for the fully dynamic problem with sensitivity d is given in the following theorem. We state the running time with respect to a blackbox algorithm for the decremental version of the problem as subprocedure. The number of initially deactivated vertices is denoted by n_{off}.

▶ **Theorem 1.** *Assume there exists an algorithm for the decremental subgraph connectivity problem with sensitivity d that has preprocessing time t_p, update time t_u, query time t_q and uses space S. Then there exists an algorithm for the fully dynamic subgraph connectivity problem with sensitivity d that uses space $O(n_{\mathit{off}}^2 \cdot S)$ and has preprocessing time $O(n_{\mathit{off}}^2 \cdot t_p)$. It can process an update of d vertices in time $O(d^2 \cdot \max\{t_u, t_q\})$ and queries in time $O(d \cdot t_q)$.*

For the decremental version of the subgraph connectivity problem with sensitivity d (which is also referred to as *d-failure connectivity*), the best known algorithm is by Duan and Pettie [11]. Their result is given in the following lemma.

▶ **Lemma 2** ([11]). *Let $G = (V, E)$ be a graph and let $n = |V|$, $m = |E|$, let $c \in \mathbb{N}$. Then there exists a data structure for the decremental subgraph connectivity problem with sensitivity d that has size $S = O(d^{1-2/c} m n^{1/c-1/(c \log(2d))} \log^2 n)$ and preprocessing time $\tilde{O}(S)$. An update deactivating d vertices takes time $O(d^{2c+4} \log^2 n \log\log n)$ and subsequent connectivity queries in the graph after the vertex deactivations take $O(d)$ time.*

As pointed out in [11], for moderate values of d the space S used by the data structure from Lemma 2 is $o(mn^{1/c})$; further, if $m < n^2$, then we always have $S = o(mn^{2/c})$. Using the algorithm of Lemma 2 as a subprocedure for our result from Theorem 1, we obtain the following corollary. The number of initially activated vertices is given by n_{on} and the number of initially activated edges in the graph is denoted by m_{on}.

▶ **Corollary 3.** *There exists an algorithm for the fully dynamic subgraph connectivity problem with sensitivity d with the following properties. For any $c \in \mathbb{N}$, it uses space $S' = O(n_{\mathit{off}}^2 \cdot S)$ and preprocessing time $\tilde{O}(S')$, where $S = O(d^{1-2/c} m_{on} n_{on}^{1/c-1/(c\log(2d))} \log^2 n_{on})$. It can process an update of d vertices in time $O(d^{2c+6} \log^2 n_{on} \log\log n_{on})$ and answer queries in the updated graph in time $O(d^2)$.*

In the case that we get an update of size $d' < d$, we can make the update and query times of the data structure depend only on d': We build the data structure for all values $d' = 2^1, \dots, 2^\ell$, where ℓ is the smallest integer such that $d \leq 2^\ell$. Asymptotically this will not use more space than building the data structure once for d; for an update of size d' we use the instance of the data structure for the smallest $2^i \geq d'$.

In the incremental algorithm we only allow for initially deactivated vertices to be activated. Our result for the incremental problem is given in the following theorem.

▶ **Theorem 4.** *There exists an algorithm for the incremental subgraph connectivity problem with sensitivity d which has preprocessing time $O(n_{\mathit{off}}^2 \cdot n_{on} + m)$, update time $O(d^2)$ and query time $O(d)$. It uses space $O(n_{\mathit{off}} \cdot n)$.*

The algorithm is simple and we believe it is implementable and efficient in practice.

For our incremental data structure the sensitivity parameter d does not have to be fixed beforehand, i.e., once initialized, the data structure can process updates of arbitrary sizes and the update and query times will only depend on the size of the given update.

We observe that the conditional lower bound given in Henzinger et al. [16] for the decremental version of the problem can easily be altered to work for the incremental problem as well. The conditional lower bound states that under the Online Matrix vector (OMv) conjecture any algorithm solving the incremental subgraph connectivity problem with sensitivity d which uses preprocessing time polynomial in n and and update time polynomial in d must have a query time of $\Omega(d^{1-\varepsilon})$ for all $\varepsilon > 0$. Examining the proof of the lower bound, we observed that the maximum of the query and update time even has to be in $\Omega(d^{2-\varepsilon})$ for all $\varepsilon > 0$. Hence, the update and query times of our incremental algorithm are essentially optimal under the OMv conjecture.

1.2 Related Work

In recent years there have been several results studying data structures for problems in an emergency planning or sensitivity setting when only a single update of small size is allowed. The field was introduced by Patrascu and Thorup [23] who considered connectivity queries after d edge failures. Demetrescu et al. [8] studied distance oracles avoiding a single failed node or edge. This setting was also considered by Bernstein and Karger [3, 4]. Later, Duan and Pettie [10] studied distance and connectivity oracles in case of two vertex failures. Khanna and Baswana [21] studied approximate shortest paths for a single vertex failure. As mentioned in Section 1.1, Duan and Pettie [11] studied the decremental subgraph connectivity problem with sensitivity d. Chechik et al. [7] considered distance oracles and routing schemes in case of d edge failures.

For the decremental subgraph connectivity problem with sensitivity d there also exist conditional lower bounds by Henzinger et al. [16] from the OMv conjecture and most recently by Kopelowitz, Pettie and Porat [22] from the 3SUM conjecture. The highest conditional lower bounds is the one in [16], which states that under the OMv conjecture any algorithm using preprocessing time polynomial in n and and update time polynomial in d must have a query time of $\Omega(d^{1-\varepsilon})$ for all $\varepsilon > 0$. Hence, the query time of the decremental algorithm by Duan and Pettie [11] is essentially optimal with respect to the lower bound.

The general subgraph connectivity problem, which allows for an arbitrary number of updates, has gained an increasing interest during the last years. The problem was introduced by Frigoni and Italiano [14], who studied it for planar graphs; they achieved amortized polylogarithmic update and query times. In general graphs, Duan [9] constructed a data structure which uses almost linear space, preprocessing time $\tilde{O}(m^{6/5})$, worst-case update time $\tilde{O}(m^{4/5})$ and worst-case query time $\tilde{O}(m^{1/5})$. In an amortized setting, the data structure given by Chan, Patrascu and Roditty [6] has an update time $\tilde{O}(m^{2/3})$ and query time $\tilde{O}(m^{1/3})$; its space usage and preprocessing time is $\tilde{O}(m^{4/3})$. This improved an earlier result by Chan [5] significantly. The data structure of [6] was later improved by Duan [9] to use only $\tilde{O}(m)$ space. Baswana et al. [2] gave a deterministic worst-case algorithm with update time $\tilde{O}(\sqrt{mn})$ and query time $O(1)$. Further, conditional lower bounds were derived for the subgraph connectivity problem from multiple conjectures [1, 16]. The highest such lower bound was given in [16]; it states that under the OMv conjecture, the subgraph connectivity problem cannot be solved faster than with update time $\Omega(m^{1-\delta})$ and query time $\Omega(m^{\delta})$ for any $\delta \in (0, 1)$ when we only allow polynomial preprocessing time of the input graph. Hence, the update and query times of the aforementioned algorithms are optimal up to polylogarithmic factors and tradeoffs between update and query times.

Compared to the subgraph connectivity problem, it has a much longer tradition to study the (edge) connectivity problem in which updates delete or add edges to the graph. Henzinger and King [17] were the first to give an algorithm with expected polylogarithmic update and query times; the best algorithm using Las Vegas randomization is by Thorup [24] with an amortized update time of $O(\log n(\log\log n)^3)$. Holm, de Lichtenberg and Thorup [18] gave the first deterministic algorithm with amortized polylogarithmic update times; currently the best such algorithm is given by Wulff-Nilsen [25] which has an update time of $O(\log^2 n/\log\log n)$. Recently, Kapron, King and Mountjoy [19] were able to provide the first data structure which has expected *worst case* polylogarithmic time per update and query. The result of [19] was lately improved by Gibb et al. [15] to have update time $O(\log^4 n)$. However, the best deterministic worst case data structures still have running times polynomial in the number of nodes of the graph. For a long time the results by Frederickson [13] and Eppstein et al. [12] running in time $O(\sqrt{n})$ were the best known. Only recently this was slightly improved by Kejlberg-Rasmussen et al. [20], who were able to obtain a worst case update time of $O\left(\sqrt{\frac{n(\log\log n)^2}{\log n}}\right)$.

The rest of the paper is outlined as follows: We start with notation and preliminaries in Section 2. In Section 3 we prove the results for the incremental algorithm which will already contain the main ideas for the more complicated fully dynamic algorithm. Section 4 provides the main result of this paper.

2 Preliminaries

In this section, we formally introduce the subgraph connectivity problem with sensitivity d. At the end of the section, we show a lemma that characterizes when disconnected vertices become connected after activating additional vertices; the lemma will be essential to prove the correctness of our algorithms.

The *subgraph connectivity problem* with sensitivity d is as follows: Let $G = (V, E)$ be a graph with n vertices and m edges and a partition of the vertices into sets V_{on} and V_{off}. The vertices in V_{on} are said to be *turned on* or *activated* and those in V_{off} are said to be *turned* *off* or *deactivated*. We get a single batch update in which the states of up to d vertices are be changed. In a query for two vertices u and v, the algorithm has to return if there exists a path from u to v only traversing activated vertices.

As we consider the subgraph connectivity problem in a sensitivity setting, after processing a single update and a sequence of queries, we roll back to the initial input graph. Hence, the data structure does not allow to alter the graph by an arbitrary amount. This allows us to offer much faster update and query times than the best algorithms which solve the general fully dynamic problem.

We introduce more notation. By G_{on} we denote the projection of G on the vertices which are initially on, i.e., $G_{\text{on}} = G[V_{\text{on}}] = (V_{\text{on}}, E_{\text{on}})$, where $E_{\text{on}} = \{(u,v) \in E : u, v \in V_{\text{on}}\}$. We set $E_{\text{off}} = E \setminus E_{\text{on}}$ to the set of edges which have at least one endpoint in V_{off}. To distinguish between the sizes of the activated and deactivated vertices and edges, we set $n_{\text{on}} = |V_{\text{on}}|$ to the number of activated vertices and $n_{\text{off}} = |V_{\text{off}}|$ to the number of deactivated vertices. Further, we set $m_{\text{on}} = |E_{\text{on}}|$ to the number of edges in G_{on} and $m_{\text{off}} = |E_{\text{off}}|$.

With this notation we can quickly describe the main difficulties of the subgraph connectivity problem: If G_{on} is connected, then already deactivating a single vertex of V_{on} can make it fall apart into $\Theta(n_{\text{on}})$ connected components; on the other hand, in G_{on} we can have $\Theta(n_{\text{on}})$ connected components initially and activating a single vertex of V_{off} with $\Theta(n_{\text{on}})$ edges can make the resulting graph connected. Hence, when deactivating or activating vertices, the number of connected components can change arbitrarily much. However, the update and query times of our algorithms are not supposed to polynomially depend on n, but only on the size of the udpate d which will usually be much smaller.

2.1 Characterisation of Paths After Activating Vertices

In this subsection, we introduce the terminology to characterize when vertices in a graph G become connected after we activated the vertices of a set I.

We say that a deactivated vertex $v \in V_{\text{off}}$ and a connected component C of G are *adjacent*, if there exists a vertex $u \in C$ such that $(u, v) \in E$. Two vertices $u, v \in V_{\text{off}}$ are *connected via a connected component*, if (1) there exists a connected component C to which both u and v are adjacent or (2) if $(u, v) \in E$. In other words, u and v are connected via a connected component if they can reach each other by a path that only traverses vertices from a single connected component of G or if u and v are connected by an edge. Two connected components $C_1 \neq C_2$ are *connected by the set* I if there exists a sequence of vertices $v_1, \dots, v_k \in I$ such that (1) v_1 is adjacent to C_u, (2) v_k is adjacent to C_v and (3) v_i and v_{i+1} are connected via a connected component for all $i = 1, \dots, k-1$.

We can characterize when two disconnected vertices become connected in G after the vertices of the set I are activated. This is done in the following lemma.

▶ **Lemma 5.** *Let $G = (V, E)$ be a graph with V_{on} and V_{off} as before. Further, let $I \subseteq V_{off}$ be a set of vertices which is activated. Let u, v be two disconnected vertices in G_{on} and let $C_u \neq C_v$ be their connected components. Then u and v are connected in G after activating the vertices in I if and only if C_u and C_v are connected by the set I.*

Proof. Assume u and v are connected in G after activating the vertices in I. Then there exists a path $u = w_0 \to w_1 \to \cdots \to w_\ell \to w_{\ell+1} = v$ in G; let $w_{j_1}, \dots, w_{j_r}$ be the vertices of the path which are from the set I with $j_i < j_{i+1}$ for all $i = 1, \dots, r$. Now observe that for all i, the vertices $w_{j_i+1}, \dots, w_{j_{i+1}-1}$ must be in the same connected component C_{j_i}. Clearly, w_{j_i} and $w_{j_{i+1}}$ are adjacent to C_{j_i} and they are connected by the connected component C_{j_i}.

The same arguments can be used to show that C_u and w_{j_1} are adjacent and to show that C_v and w_{j_r} are adjacent. This implies that C_u and C_v are connected by the set I.

The other direction of the proof is symmetric. ◀

We will use Lemma 5 to argue about the correctness of our algorithms. In particular, when we prove the correctness of our algorithms we show that the connected components of the query vertices become connected by the set I of newly activated vertices. This is useful as we can preprocess which vertices of V_{off} are connected via connected components and which deactivated vertices are reachable from the connected components of G. With these properties, we are able to avoid having to keep track of all connected components of G after an update.

3 Incremental Algorithm

In this section, we describe an algorithm for the incremental subgraph connectivity problem that has preprocessing time $O(n_{\text{off}}^2 \cdot n_{\text{on}} + m)$, update time $O(d^2)$, query time $O(d)$ and uses space $O(n_{\text{off}} \cdot n)$. This will prove Theorem 4 stated in the introduction.

The main idea of the algorithm is to exploit Lemma 5 by preprocessing which deactivated vertices are connected by connected components of G_{on} and preprocessing the adjacency of deactivated vertices and connected components of G_{on}.

3.1 Preprocessing

We first compute the connected components $C_1, \dots, C_k$ of G_{on} and label each vertex in V_{on} with its connected component. For each connected component C_i, we use a binary array A_{C_i} of size n_{off} to store which vertices in V_{off} are adjacent to C_i. We further equip each vertex $u \in V_{\text{off}}$ with a binary array A_u of size $k + n_{\text{off}} = O(n)$: In the first k entries of A_u, we store to which connected components u is connected; in the final n_{off} components of A_u, we store to which $v \in V_{\text{off}}$ the vertex u is connected by a connected component.

When the algorithm performs updates, it uses the arrays A_u to determine in constant time if u is connected to other deactivated vertices from V_{off} via connected components. This avoids having to check all connected components C_i of the vertex u which could take time $\Theta(n_{\text{on}})$.

The preprocessing takes time $O(m_{\text{on}})$ to compute the connected components C_i and labeling the vertices in V_{on}. Using one pass over all edges we can compute the arrays containing the connectivity information between the C_i and V_{off}, i.e., we can fill the arrays A_{C_i} and the first k components of the A_u. This takes $O(m)$ time.

Notice that if u is adjacent to C_i, then A_u must have a 1 wherever A_{C_i} has a 1. Then to finish building the arrays A_u, we can compute the last n_{off} entries of A_u as the bitwise OR of the arrays A_{C_i} for all C_i which u is adjacent to. This can be done in time $O(k \cdot n_{\text{off}}) = O(n_{\text{on}} \cdot n_{\text{off}})$ for a single vertex u. Since we have n_{off} vertices in V_{off}, computing all A_u takes time $O(n_{\text{off}}^2 \cdot n_{\text{on}})$. The computation of the A_u therefore dominates the running time of the preprocessing.

The space we require during the preprocessing is $O(n_{\text{off}})$ for each connected component of G_{on} and $O(n)$ for each vertex in V_{off}. Hence, in total we require $O(n_{\text{off}} \cdot n)$ space and preprocessing time $O(n_{\text{off}}^2 \cdot n_{\text{on}} + m)$.

3.2 Updates

During an update which activates d vertices from a set I, we build the *increment graph S* with the vertices of I as its nodes. We add an edge between a pair of vertices $u, v \in I$ if they are connected by a connected component C of G_{on}. Notice that the increment graph encodes the connectivity of the vertices in I via the connected components of G_{on}.

Computationally, this can be done in time $O(d^2)$: For each pair of vertices $u, v \in I$, we check in A_u if u is connected to v via a connected component in time $O(1)$. As we have to consider $O(d^2)$ pairs of vertices, the total time to construct the increment graph is $O(d^2)$.

Finally, we compute the connected components $S_1, \dots, S_\ell$ of S and label each vertex in S with its connected component. This can be done in time $O(|S|) = O(d^2)$. Hence, the total update time is $O(d^2)$.

3.3 Queries

Consider a query if two activated vertices u and v are connected.

We find the connected components C_u and C_v of u and v, respectively. If $C_u = C_v$, then we return that u and v are connected and we are done.

Otherwise, let S_i be a connected component of S. We consider each vertex w of S_i and check if it is connected to C_u or C_v using A_{C_u} and A_{C_v}. After considering all vertices of S_i, we check if both C_u and C_v are connected to S_i. If this is the case, we return that u and v are connected, otherwise, we proceed to the next connected component of S.

During the query we considered each vertex in S exactly once and spent time $O(1)$ processing it. Hence, the total query time is $O(d)$.

It is left to prove the correctness of the result of the queries. This is done in the following lemma.

▶ **Lemma 6.** *Consider an update which activates the vertices from a set $I \subseteq V_{\text{off}}$. Then a query if two vertices u and v are connected in G after the update delivers the correct result.*

Proof. If in the query procedure we encountered that $C_u = C_v$, then the result of the algorithm is clearly correct.

If $C_u \neq C_v$, then observe that the algorithm returns true if and only if C_u and C_v are connected by the set I: Let S_i be the connected component of S for which the query returns true. Then there must exist vertices $w_1, \dots, w_t$ in the increment graph such that (1) w_1 is adjacent to C_u, (2) w_t is adjacent to C_v and (3) (w_i, w_{i+1}) is an edge in S for all $i = 1, \dots, t-1$. The first two claims are true because the query procedure checks this in the arrays A_{C_u} and A_{C_v}. By construction of the increment graph, the increment graph has an edge (w_i, w_{i+1}) if and only if those vertices are connected by a connected component (this follows from what we preprocessed in the arrays A_{w_i}). This implies that a query returns true iff C_u and C_v are connected by the set I.

By Lemma 5 the algorithm returns the correct answer. ◀

4 Fully Dynamic Algorithm

In this section, we present the main result of the paper. We provide a data structure for the fully dynamic subgraph connectivity problem with sensitivity d, i.e., we process a batch update which changes the states of at most d vertices. Our algorithm uses a data structure for the decremental problem as a subprocedure. Assume the decremental algorithm uses space S, preprocessing time t_p, update time t_u and query time t_q. Then the fully dynamic algorithm uses space $O(n_{\text{off}}^2 \cdot S)$, preprocessing time $O(n_{\text{off}}^2 \cdot t_p)$, update time $O(d^2 \cdot \max\{t_u, t_q\})$ and query time $O(d \cdot t_q)$.

We reuse the increment graphs which we used in the incremental algorithm. For the construction of the increment graphs we replace the vectors A_u and A_{C_i} of the previous section by slightly augmented versions of G_{on} which are equipped with a decremental subgraph connectivity data structure, e.g., the one of Lemma 2 by Duan and Pettie [11]. The purpose of the augmented graphs is to check if a pair of initially deactivated vertices is connected via a connected component after deactivating some vertices of V_{on}.

We sketch the main steps of our algorithm. In the preprocessing we build an augmented graph for each pair of vertices of V_{off}; each augmented graph is equipped with a decremental subgraph connectivity data structure. In an update, we first process the vertex deactivations in the augmented graphs. Then we build the increment graph of vertices that were activated. Queries are handled similarly to the incremental algorithm by using the increment graph, but we have to check if the vertices of the increment graph can still reach the query vertices (this connectivity may have been destroyed by the vertex deactivations).

4.1 Preprocessing

For each pair of nodes $u, v \in V_{\text{off}}$, we build the augmented graph $G_{u,v} = G[V_{\text{on}} \cup \{u, v\}]$, i.e., $G_{u,v}$ consists of G_{on} after adding u and v. Observe that u and v cannot introduce more than $O(n_{\text{on}})$ edges and hence $G_{u,v}$ still has $O(n_{\text{on}})$ vertices and $O(m_{\text{on}})$ edges. We equip $G_{u,v}$ with a decremental subgraph connectivity data structure with sensitivity d. Later, we use the graph $G_{u,v}$ to check if u and v are connected via a connected component after deleting vertices from G_{on}; intuitively, the graphs $G_{u,v}$ replace the vectors A_u and A_v of the incremental algorithm. We need space $O(n_{\text{off}}^2 \cdot S)$ to store the $G_{u,v}$ where S is the space to store G_{on} with the decremental data structure.

For each $u \in V_{\text{off}}$, we build the graph $G_u = G[V_{\text{on}} \cup \{u\}]$ and equip it with the decremental data structure; we further equip G_{on} with the decremental data structure. We use the graphs G_u to replace the arrays A_{C_i} of the incremental algorithm; we cannot use the arrays anymore because the connected components of G_{on} can fall apart due to vertex deactivations. The space we need to store the graphs G_u and G is $O(n_{\text{off}} \cdot S)$.

In total, the preprocessing takes space $O(n_{\text{off}}^2 \cdot S)$ and time $O(n_{\text{off}}^2 \cdot t_p)$.

4.2 Updates

Assume that we get an update U which deactivates the vertices of a set $D \subseteq V_{\text{on}}$ and activates the vertices of a set $I \subseteq V_{\text{off}}$ with $|D| + |I| \leq d$. Our update procedure has two steps: We first remove the vertices in D from $G_{u,v}$ for all newly activated vertices $u, v \in I$. After that we build the increment graph consisting of the vertices of I as we did in the incremental algorithm.

We describe the sketched steps of the update procedure in more detail. Firstly, we process the deletions of the set D. For each pair $u, v \in I$, we delete the vertices of D in $G_{u,v}$ in time t_u. Since we have $O(d^2)$ pairs of vertices of I to consider, this takes time $O(d^2 \cdot t_u)$.

We update G_{on} and all G_u by deleting the vertices the vertices from D. This does not take longer than updating the graphs $G_{u,v}$.

Secondly, we build the increment graph consisting of the vertices in I. For each pair of vertices $u, v \in I$, we add an edge $e = (u, v)$ to the increment graph if a query in $G_{u,v}$ returns that u and v are connected. Such a query takes time t_q. The time we spend to build the increment graph is $O(d^2 \cdot t_u)$. Finally, we compute the connected components of the increment graph in time $O(d^2)$.

Altogether, the total update time of the update procedure is $O(d^2 \cdot \max\{t_u, t_q\})$.

4.3 Queries

We handle the query if two vertices u and v of G are connected similarly as in the incremental algorithm by using the increment graph.

Before we use the increment graph, we query if u and v are connected in the instance of G_{on} in which we deactivated the vertices of the set D. If the query returns true, then u and v are connected, otherwise, we proceed by using the increment graph.

For each connected component B of the increment graph, we consider each vertex $w \in B$ and we query in G_w if w is connected to u or v. If B had vertices w, w' which are connected to u and v, respectively, then we return that u and v are connected. Otherwise, we proceed to the next connected component of the increment graph.

The total query time of our algorithm is $O(d \cdot t_q)$ as in the worst case we have to perform a query in G_w for each of the $O(d)$ vertices $w \in I$.

Notice that due to the vertex deactivations we cannot precompute the connected components C_i of G_{on} and their connectivity with vertices in V_{off} as we did in the incremental algorithm: Each C_i may consist of $\Theta(n_{\text{on}})$ vertices and might as well fall apart into $\Theta(n)$ connected components after the vertex deactivations. Hence, in the update procedure we cannot keep the information about the connectivity of the vertices C_i and the added vertices up to date, as this may take time $\Theta(n)$. For our construction this also rules out obtaining a better query time.

We conclude the section by proving that the query returns the correct results in the following lemma.

▶ **Lemma 7.** *Consider an update U deactivating the vertices from a set D and activating the ones from a set I. Then a query if two vertices u and v are connected in G after the update delivers the correct result.*

Proof. In the query procedure, we first check if u and v are connected in G_{on} after deleting the vertices from D. Clearly, if the algorithm returns true, then u and v are connected.

We move on to argue about the correctness in the case that u and v are not connected in the graph $H = G_{\text{on}} \setminus D$. Let $C_1, \dots, C_k$ be the connected components of H (not those of G_{on}) and let C_u and C_v be the connected components of u and v. We show that a query returns that u and v are connected if and only if C_u and C_v are connected by the set I. Then Lemma 5 implies the correctness of the algorithm.

Observe that a query returns that u and v are connected if and only if there exists a connected component B in the increment graph which contains vertices $w_1, \dots, w_\ell \in B \subseteq I$, such that (1) w_1 is connected to u, (2) w_ℓ is connected to v and (3) there is an edge between w_i and w_{i+1} in the increment graph for all $i = 1, \dots, \ell - 1$: We obtain the first two properties from the queries in G_{w_1} and G_{w_ℓ}; the third property is true due to the queries in the augmented graphs $G_{w_i, w_{i+1}}$ and implies that the w_i are connected via connected components.

Hence, we conclude that a query returns that u and v are connected if and only if C_u and C_v are connected by the set I. Lemma 5 implies that the algorithm is correct. ◀

Acknowledgements. We would like to thank the reviewers for their helpful comments.

References

1 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *IEEE 55th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 434–443, Philadelphia, PA, USA, 2014.

2 Surender Baswana, Shreejit Ray Chaudhury, Keerti Choudhary, and Shahbaz Khan. Dynamic DFS tree in undirected graphs: breaking the O(m) barrier. *CoRR*, abs/1502.02481, 2015.

3 Aaron Bernstein and David Karger. Improved distance sensitivity oracles via random sampling. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 34–43, Philadelphia, PA, USA, 2008.

4 Aaron Bernstein and David Karger. A nearly optimal oracle for avoiding failed vertices and edges. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing (STOC)*, pages 101–110, New York, NY, USA, 2009.

5 Timothy M. Chan. Dynamic subgraph connectivity with geometric applications. In *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing (STOC)*, pages 7–13, New York, NY, USA, 2002.

6 Timothy M Chan, Mihai Patrascu, and Liam Roditty. Dynamic connectivity: Connecting to networks and geometry. *SIAM Journal on Computing*, 40(2):333–349, 2011.

7 Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. f-sensitivity distance oracles and routing schemes. *Algorithmica*, 63(4):861–882, 2011.

8 Camil Demetrescu, Mikkel Thorup, Rezaul Alam Chowdhury, and Vijaya Ramachandran. Oracles for distances avoiding a failed node or link. *SIAM Journal on Computing*, 37(5):1299–1318, 2008.

9 Ran Duan. New data structures for subgraph connectivity. In *37th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 201–212, Bordeaux, France, 2010.

10 Ran Duan and Seth Pettie. Dual-failure distance and connectivity oracles. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 506–515, Philadelphia, PA, USA, 2009.

11 Ran Duan and Seth Pettie. Connectivity oracles for failure prone graphs. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing (STOC)*, pages 465–474, New York, NY, USA, 2010.

12 David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Amnon Nissenzweig. Sparsification – a technique for speeding up dynamic graph algorithms. *Journal of the ACM*, 44(5):669–696, 1997.

13 Greg N. Frederickson. Data structures for on-line updating of minimum spanning trees. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing (STOC)*, pages 252–257, New York, NY, USA, 1983.

14 D. Frigioni and F. G. Italiano. Dynamically switching vertices in planar graphs. *Algorithmica*, 28(1):76–103, 2000.

15 David Gibb, Bruce M. Kapron, Valerie King, and Nolan Thorn. Dynamic graph connectivity with improved worst case update time and sublinear space. *CoRR*, abs/1509.06464, 2015.

16 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing (STOC)*, pages 21–30, New York, NY, USA, 2015.

17 Monika R. Henzinger and Valerie King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *Journal of the ACM*, 46(4):502–516, 1999.

18 Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *Journal of the ACM*, 48(4):723–760, 2001.

19 Bruce M. Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1131–1142, 2013.

20 Casper Kejlberg-Rasmussen, Tsvi Kopelowitz, Seth Pettie, and Mikkel Thorup. Faster worst case deterministic dynamic connectivity. *CoRR*, abs/1507.05944, 2015.

21 Neelesh Khanna and Surender Baswana. Approximate Shortest Paths Avoiding a Failed Vertex: Optimal Size Data Structures for Unweighted Graphs. In *27th International Symposium on Theoretical Aspects of Computer Science (STACS)*, Leibniz International Proceedings in Informatics (LIPIcs), pages 513–524, 2010.

22 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3sum conjecture. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1272–1287, 2016.

23 Mihai Patrascu and Mikkel Thorup. Planning for fast connectivity updates. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 263–271, 2007.

24 Mikkel Thorup. Near-optimal fully-dynamic graph connectivity. In *Proceedings of the Thirty-second Annual ACM Symposium on Theory of Computing (STOC)*, pages 343–350, 2000.

25 Christian Wulff-Nilsen. Faster deterministic fully-dynamic graph connectivity. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1757–1769, 2013.

A Combinatorial Approximation Algorithm for Graph Balancing with Light Hyper Edges

Chien-Chung Huang[1] **and Sebastian Ott**[2]

1 **Chalmers University of Technology, Göteborg, Sweden**
villars@gmail.com

2 **Max-Planck-Institut für Informatik, Saarbrücken, Germany**
ott@mpi-inf.mpg.de

Abstract

Makespan minimization in restricted assignment $(R|p_{ij} \in \{p_j, \infty\}|C_{\max})$ is a classical problem in the field of machine scheduling. In a landmark paper in 1990 [9], Lenstra, Shmoys, and Tardos gave a 2-approximation algorithm and proved that the problem cannot be approximated within 1.5 unless P=NP. The upper and lower bounds of the problem have been essentially unimproved in the intervening 25 years, despite several remarkable successful attempts in some special cases of the problem [2, 3, 13] recently.

In this paper, we consider a special case called *graph-balancing with light hyper edges*, where heavy jobs can be assigned to at most two machines while light jobs can be assigned to any number of machines. For this case, we present algorithms with approximation ratios strictly better than 2. Specifically,

- **Two job sizes**: Suppose that light jobs have weight w and heavy jobs have weight W, and $w < W$. We give a 1.5-approximation algorithm (note that the current 1.5 lower bound is established in an even more restrictive setting [1, 4]). Indeed, depending on the specific values of w and W, sometimes our algorithm guarantees sub-1.5 approximation ratios.
- **Arbitrary job sizes**: Suppose that W is the largest given weight, heavy jobs have weights in the range of $(\beta W, W]$, where $4/7 \leq \beta < 1$, and light jobs have weights in the range of $(0, \beta W]$. We present a $(5/3 + \beta/3)$-approximation algorithm.

Our algorithms are purely combinatorial, without the need of solving a linear program as required in most other known approaches.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Approximation Algorithms, Machine Scheduling, Graph Balancing, Combinatorial Algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.49

1 Introduction

Let $\mathcal{J}$ be a set of n jobs and $\mathcal{M}$ a set of m machines. Each job $j \in \mathcal{J}$ has a *weight* w_j and can be assigned to a specific subset of the machines. An assignment $\sigma : \mathcal{J} \to \mathcal{M}$ is a mapping where each job is mapped to a machine to which it can be assigned. The objective is to minimize the *makespan*, defined as $\max_{i \in \mathcal{M}} \sum_{j:\sigma(j)=i} w_j$. This is the classical MAKESPAN MINIMIZATION IN RESTRICTED ASSIGNMENT $(R|p_{ij} \in \{p_j, \infty\}|C_{\max})$, itself a special case of the MAKESPAN MINIMIZATION IN UNRELATED MACHINES $(R||C_{\max})$, where a job j has possibly different weight w_{ij} on different machines $i \in \mathcal{M}$. In the following, we just call them RESTRICTED ASSIGNMENT and UNRELATED MACHINE PROBLEM for short.

The first constant approximation algorithm for both problems is given by Lenstra, Shmoys, and Tardos [9] in 1990, where the ratio is 2. They also show that RESTRICTED ASSIGNMENT (hence also the UNRELATED MACHINE PROBLEM) cannot be approximated within 1.5 unless P=NP, even if there are only two job weights. The upper bound of 2 and the lower bound of 1.5 have been essentially unimproved in the intervening 25 years. How to close the gap continues to be one of the central topics in approximation algorithms. The recent book of Williamson and Shmoys [15] lists this as one of the ten open problems.

Our Result

We consider a special case of RESTRICTED ASSIGNMENT, called GRAPH BALANCING WITH LIGHT HYPER EDGES, which is a generalization of the GRAPH BALANCING problem introduced by Ebenlendr, Krčál and Sgall [4]. There the restriction is that every job can be assigned to only two machines, and hence the problem can be interpreted in a graph-theoretic way: each machine is represented by a node, and each job is represented by an edge. The goal is to find an orientation of the edges so that the maximum weight sum of the edges oriented towards a node is minimized. In our problem, jobs are partitioned into *heavy* and *light*, and we assume that heavy jobs can go to only two machines while light jobs can go to any number of machines[1]. In the graph-theoretic interpretation, light jobs are represented by hyper edges, while heavy jobs are represented by regular edges.

We present approximation algorithms with performance guarantee *strictly better than* 2 in the following settings. For simplicity of presentation, we assume that all job weights w_j are integral (this assumption is just for ease of exposition and can be easily removed).

Two job sizes: Suppose that heavy jobs are of weight W and light jobs are of weight w, and $w < W$. We give a 1.5-approximation algorithm, matching the general lower bound of RESTRICTED ASSIGNMENT (it should be noted that this lower bound is established in an even more restrictive setting [1, 4], where all jobs can only go to two machines and job weights are 1 and 2). This is the first time the lower bound is matched in a nontrivial case of RESTRICTED ASSIGNMENT (without specific restrictions on the job weight values). In fact, sometimes our algorithm achieves an approximation ratio strictly better than 1.5. Supposing that $w \leq \frac{W}{2}$, the ratio we get is $1 + \frac{\lfloor W/2 \rfloor}{W}$.

Arbitrary job sizes: Suppose that $\beta \in [4/7, 1)$ and W is the largest given weight. A heavy job has weight in $(\beta W, W]$ while a light job has weight in $(0, \beta W]$. We give a $(5/3 + \beta/3)$-approximation algorithm.

Both algorithms have the running time of $\mathcal{O}\big(n^2 m^3 \log\big(\sum_{j\in\mathcal{J}} w_j\big)\big)$.[2]

The general message of our result is clear: as long as the heaviest jobs have only two choices, it is relatively easy to break the barrier of 2 in the upper bound of RESTRICTED ASSIGNMENT. This should coincide with our intuition. The heavy jobs are in a sense the "trouble-makers". A mistake on them causes bigger damage than a mistake on lighter jobs. Restricting the choices of the heavy jobs thus simplifies the task.

[1] If some jobs can be assigned to just one machine, then it is the same as saying a machine has some *dedicated load*. All our algorithms can handle arbitrary dedicated loads on the machines.

[2] For simplicity, here we upper bound $\sum_{j\in\mathcal{J}} a_j$, where a_j is the number of the machines j can be assigned to, by nm.

The original GRAPH BALANCING problem assumes that all jobs can be assigned to only two machines and the algorithm of Ebenlendr et al. [4] gives a 1.75-approximation. According to [11], their algorithm can be extended to our setting: given any $\beta \in [0.5, 1)$, they can obtain a $(3/2 + \beta/2)$-approximation. Although this ratio is superior to ours, let us emphasize two interesting aspects of our approach.

(1) The algorithm of Ebenlendr et al. requires solving a linear program (in fact, almost all known algorithms for the problem are LP-based), while our algorithms are purely combinatorial. In addition to the advantage of faster running time, our approach introduces new proof techniques (which do not involve linear programming duality).

(2) In GRAPH BALANCING, Ebenlendr et al. showed that with only two job weights and dedicated loads on the machines, their strongest LP has the integrality gap of 1.75, while we can break the gap. Our approach thus offers a possible angle to circumvent the barrier posed by the integrality gap, and has the potential of seeing further improvement.

Before explaining our technique in more detail, we should point out another interesting connection with a result of Svensson [13] for general RESTRICTED ASSIGNMENT. He gave two local search algorithms, which terminate (but it is unknown whether in polynomial time) and (1) with two job weights $\{\epsilon, 1\}$, $0 < \epsilon < 1$, the returned solution has an approximation ratio of $5/3 + \epsilon$, and (2) with arbitrary job weights, the returned solution has an approximation ratio of ≈ 1.94. It is worth noting that his analysis is done via the primal-duality of the configuration-LP (thus integrality gaps smaller than two for the configuration-LP are implied). With two job weights, our algorithm has some striking similarity to his algorithm – but it should be emphasized that the two algorithms behave differently. We are able to prove our algorithm terminates in polynomial time – but our setting is more restrictive. A very interesting direction for future work is to investigate how the ideas in the two algorithms can be related and combined.

Our Technique

Our approach is inspired by that of Gairing et al. [5] for general RESTRICTED ASSIGNMENT. So let us first review their ideas. Suppose that a certain optimal makespan t is guessed. Their core algorithm either (1) correctly reports that t is an underestimate of OPT, or (2) returns an assignment with makespan at most $t + W - 1$. By a binary search on the smallest t for which an assignment with makespan $t + W - 1$ is returned, and the simple fact that $\text{OPT} \geq W$, they guarantee the approximation ratio of $\frac{t+W-1}{\text{OPT}} \leq 1 + \frac{W-1}{\text{OPT}} \leq 2 - \frac{1}{W}$ (the first inequality holds because t is the smallest number an assignment is returned by the core algorithm). Their core algorithm is a preflow-push algorithm. Initially all jobs are arbitrarily assigned. Their algorithm tries to redistribute the jobs from overloaded machines, i.e., those with load more than $t + W - 1$, to those that are not. The redistribution is done by pushing the jobs around while updating the height labels (as commonly done in preflow-push algorithms). The critical thing is that after a polynomial number of steps, if there are still some overloaded machines, they use the height labels to argue that t is a wrong guess, i.e., $\text{OPT} \geq t + 1$. Our contribution is a refined core algorithm in the same framework. With a guess t of the optimal makespan, our core algorithm either (1) correctly reports that $\text{OPT} \geq t + 1$, or (2) returns an assignment with makespan at most $(5/3 + \beta/3)t$.

We divide all jobs into two categories, the *rock jobs* $\mathbb{R}$, and the *pebble jobs* $\mathbb{P}$ (not to be confused with heavy and light jobs). The former consists of those with weights in $(\beta t, t]$ while the latter includes all the rest. We use the rock jobs to form a graph $G_{\mathbb{R}} = (V, \mathbb{R})$, and assign the pebbles arbitrarily to the nodes. Our core algorithm will push around the pebbles so as to redistribute them. Observe that as $t \geq W$, all rocks are heavy jobs. So the formed graph $G_{\mathbb{R}}$ has only simple edges (no hyper edges). As $\beta \geq 4/7$, if OPT $\leq t$, then every node can receive at most one rock job in the optimal solution. In fact, it is easy to see that we can simply assume that the formed graph $G_{\mathbb{R}}$ is a disjoint set of trees and cycles. Our entire task boils down to the following:

> Redistribute the pebbles so that there exists an orientation of the edges in $G_{\mathbb{R}}$ in which each node has total load (from both rocks and pebbles) at most $(5/3 + \beta/3)t$; and if not possible, gather evidence that t is an underestimate.

Intuitively speaking, our algorithm maintains a certain *activated set* $\mathbb{A}$ of nodes. Initially, this set includes those nodes whose total loads of pebbles cause conflicts in the orientation of the edges in $G_{\mathbb{R}}$. A node "reachable" from a node in the activated set is also included into the set. (Node u is reachable from node v if a pebble in v can be assigned to u.) Our goal is to push the pebbles among nodes in $\mathbb{A}$, so as to remove all conflicts in the edge orientation. Either we are successful in doing so, or we argue that the total load of all pebbles currently owned by the activated set, together with the total load of the rock jobs assigned to $\mathbb{A}$ in any *feasible orientation* of the edges in $G_{\mathbb{R}}$ (an orientation in $G_{\mathbb{R}}$ is *feasible* if every node receives at most one rock), is strictly larger than $t \cdot |\mathbb{A}|$. The progress of our algorithm (hence its running time) is monitored by a potential function, which we show to be monotonically decreasing.

The most sophisticated part of our algorithm is the "activation strategy". We initially add nodes into $\mathbb{A}$ if they cause conflicts in the orientation or can be (transitively) reached from such. However, sometimes we also include nodes that do not fall into the two categories. This is purposely done for two reasons: pushing pebbles from these nodes may help alleviate the conflict in edge orientation indirectly; and their presence in $\mathbb{A}$ strengthens the contradiction proof.

Due to the intricacy of our main algorithm, we first present the algorithm for the two job weights case in Section 3 and then present the main algorithm for the arbitrary weights in Section 4. The former algorithm is significantly simpler (with a straightforward activation strategy) and contains many ingredients of the ideas behind the main algorithm.

Due to the space limit, some of the proofs are omitted. Please refer to the full version [7] for details.

Related Work

For RESTRICTED ASSIGNMENT, besides the several recent advances mentioned earlier, see the survey of Leung and Li for other special cases [10]. For two job weights, Chakrabarti, Khanna and Li [2] showed that using the configuration-LP, they can obtain a $(2 - \delta)$-approximation for a fixed $\delta > 0$ (and note that there is no restriction on the number of machines a job can go to). Kolliopoulos and Moysoglou [8] also considered the two job weights case. In the GRAPH BALANCING setting (with two job weights), they gave a 1.652-approximation algorithm using a flow technique (thus they also break the integrality gap in [3]). They also show that the optimal makespan for RESTRICTED ASSIGNMENT with two job weights can be estimated in polynomial time within a factor of at most 1.883 (and this is further improved to 1.833 in [2]).

For UNRELATED MACHINES, Shchepin and Vakhania [12] improved the approximation ratio to $2 - 1/m$. A combinatorial 2-approximation algorithm was given by Gairing, Monien, and Woclaw [6]. Verschae and Wiese [14] showed that the configuration-LP has integrality gap of 2, even if every job can be assigned to only two machines. They also showed that it is possible to achieve approximation ratios strictly better than 2 if the job weights w_{ij} respect some constraints.

2 Preliminary

Let t be a guess of OPT. Given t, our two core algorithms either report that OPT $\geq t+1$, or return an assignment with makespan at most $1.5t$ or $(5/3+\beta/3)t$, respectively. We conduct a binary search on the smallest $t \in [W, \sum_{j\in\mathcal{J}} w_j]$ for which an assignment is returned by the core algorithms. This particular assignment is then the desired solution.

We now explain the initial setup of the core algorithms. In our discussion, we will not distinguish a machine and a node. Let $dl(v)$ be the dedicated load of v, i.e., the sum of the weights of jobs that can only be assigned to v. We can assume that $dl(v) \leq t$ for all nodes v. Let $\mathcal{J}' \subseteq \mathcal{J}$ be the jobs that can be assigned to at least two machines. We divide $\mathcal{J}'$ into rocks $\mathbb{R}$ and pebbles $\mathbb{P}$. A job $j \in \mathcal{J}'$ is a rock,

- in the 2 job weights case (Section 3), if $w_j > t/2$ and $w_j = W$;
- in the general job weights case (Section 4), if $w_j > \beta t$.

A job $j \in \mathcal{J}'$ that is not a rock is a pebble. Define the graph $G_{\mathbb{R}} = (V, \mathbb{R})$ as a graph with machines $\mathcal{M}$ as node set and rocks $\mathbb{R}$ as edge set. By our definition, a rock can be assigned to exactly two machines. So $G_{\mathbb{R}}$ has only simple edges (no hyper edges). For the sake of convenience, we call the rocks just "edges", avoiding ambiguity by exclusively using the term "pebble" for the pebbles.

Suppose that OPT $\leq t$. Then a machine can receive at most one rock in the optimal solution. If any connected component in $G_{\mathbb{R}}$ has more than one cycle, we can immediately declare that OPT $\geq t+1$. If a connected component in $G_{\mathbb{R}}$ has exactly one cycle, we can direct all edges away from the cycle and remove these edges, i.e., assign the rock to the node v to which it is directed. W.L.O.G, we can assume that this rock is part of v's dedicated load. (Also observe that then node v must become an isolated node). Finally, we can eliminate cycles of length 2 in $G_{\mathbb{R}}$ with the following simple reduction. If a pair of nodes u and v is connected by two distinct rocks $r1$ and $r2$, remove the two rocks, add $\min(w_{r1}, w_{r2})$ to both u's and v's dedicated load, and introduce a new pebble of weight $|w_{r1} - w_{r2}|$ between u and v. Let Ψ denote the set of orientations in $G_{\mathbb{R}}$ where each node has at most one incoming edge. We use a proposition to summarize the above discussion.

▶ **Proposition 1.** *We can assume that*

- *the rocks in $\mathbb{R}$ correspond to the edge set of the graph $G_{\mathbb{R}}$, and all pebbles can be assigned to at least two machines;*
- *the graph $G_{\mathbb{R}}$ consists of disjoint trees, cycles (of length more than 2), and isolated nodes;*
- *for each node $v \in V$, $dl(v) \leq t$;*
- *if OPT $\leq t$, then the orientation of the edges in $G_{\mathbb{R}}$ in the optimal assignment must be one of those in Ψ.*

3 The 2-Valued Case

In this section, we describe the core algorithm for the two job weights case, with the guessed makespan $t \geq W$. Observe that when $t \in [W, 2w)$, if OPT $\leq t$, then every node can receive at most one job (pebble or rock) in the optimal assignment. Hence, we can solve the problem exactly using the standard max-flow technique. So in the following, assume that $t \geq 2w$.

```
EXPLORE1
Initialize $\mathbb{A} := \{v | v$ is hypercritical, or $v$ is critical in a bad system$\}$.
    Set LEVEL$(v) := 0$ for all nodes in $\mathbb{A}$; $i := 0$.
While $\exists v \notin \mathbb{A}$ reachable from $\mathbb{A}$ do:
    $i := i + 1$.
    $\mathbb{A}_i := \{v \notin \mathbb{A} | v$ reachable from $\mathbb{A}\}$.
    $\mathbb{A}'_i := \{v \notin \mathbb{A} |\ v$ is critical in a good system and $\exists u \in \mathbb{A}_i$ in the same system$\}$.
    Set LEVEL$(v) := i$ for all nodes in $\mathbb{A}_i$ and $\mathbb{A}'_i$.
    $\mathbb{A} := \mathbb{A} \cup \mathbb{A}_i \cup \mathbb{A}'_i$.
For each node $v \notin \mathbb{A}$, set LEVEL$(v) = \infty$.
```

Figure 1 The procedure EXPLORE1.

Furthermore, let us first assume that $t < 2W$ (the case of $t \geq 2W$ will be discussed at the end of the section). Then the rocks have weight W and the pebbles have weight w. Initially, the pebbles are arbitrarily assigned to the nodes. Let $pl(v)$ be the total weight of the pebbles assigned to node v.

▶ **Definition 2.** A node v is

- *uncritical*, if $dl(v) + pl(v) \leq 1.5t - W - w$;
- *critical*, if $dl(v) + pl(v) > 1.5t - W$;
- *hypercritical*, if $dl(v) + pl(v) > 1.5t$.

(Notice that it is possible that a node is neither uncritical nor critical.)

▶ **Definition 3.** Each tree, cycle, or isolated node in $G_{\mathbb{R}}$ is a *system*. A system is *bad* if any of the following conditions holds.

- It is a tree and has at least two critical nodes, or
- It is a cycle and has at least one critical node, or
- It contains a hypercritical node.

A system that is not bad is *good*.

If all systems are good, then orienting the edges in each system such that every node has at most one incoming edge gives us a solution with makespan at most $1.5t$. So let assume that there is at least one bad system.

We next define the *activated set* $\mathbb{A}$ of nodes constructively. Roughly speaking, we will move pebbles around the nodes in $\mathbb{A}$ so that either there is no more bad system left, or we argue that, in every feasible assignment, *some* nodes in $\mathbb{A}$ cannot handle their total loads, thereby arriving at a contradiction.

In the following, if a pebble in u can be assigned to node v, we say v is reachable from u. Node v is reachable from $\mathbb{A}$ if v is reachable from any node $u \in \mathbb{A}$. A node added into $\mathbb{A}$ is *activated*.

Informally, all nodes that cause a system to be bad are activated. A node reachable from $\mathbb{A}$ is also activated. Furthermore, suppose that a system is good and it has a critical node v (thus the system cannot be a cycle). If any other node u in the same system is activated, then so is v. We now give the formal procedure EXPLORE1 in Figure 1. Notice that in the process of activating the nodes, we also define their *levels*, which will be used later for the algorithm and the potential function.

The next proposition follows straightforwardly from EXPLORE1.

▶ **Proposition 4.** *The following holds.*

1. *All nodes reachable from $\mathbb{A}$ are in $\mathbb{A}$.*
2. *Suppose that v is reachable from $u \in \mathbb{A}$. Then $\textsc{Level}(v) \leq \textsc{Level}(u) + 1$.*
3. *If a node v is critical and there exists another node $v' \in \mathbb{A}$ in the same system, then $\textsc{Level}(v) \leq \textsc{Level}(v')$.*
4. *Suppose that node $v \in \mathbb{A}$ has $\textsc{Level}(v) = i > 0$. Then there exists another node $u \in \mathbb{A}$ with $\textsc{Level}(u) = i - 1$ so that either v is reachable from u, or there exists another node $v' \in \mathbb{A}$ reachable from u with $\textsc{Level}(v') = i$ in the same system as v and v is critical.*

After Explore1, we apply the Push operation (if possible), defined as follows.

▶ **Definition 5.** Push operation: push a pebble from u^* to v^* if the following conditions hold.

1. The pebble is at u^* and it can be assigned to v^*.
2. $\textsc{Level}(v^*) = \textsc{Level}(u^*) + 1$.
3. v^* is uncritical, or v^* is in a good system that remains good with an additional weight of w at v^*.
4. Subject to the above three conditions, choose a node u^* so that $\textsc{Level}(u^*)$ is minimized (if there are multiple candidates, pick any).

Our algorithm can be simply described as follows.

Algorithm 1: As long as there is a bad system, apply Explore1 and Push operation repeatedly. When there is no bad system left, return a solution with makespan at most $1.5t$. If at some point, push is no longer possible, declare that OPT $\geq t + 1$.

▶ **Lemma 6.** *When there is at least one bad system and the* Push *operation is no longer possible,* OPT $\geq t + 1$.

Proof. Let $\mathbb{A}(S)$ denote the set of activated nodes in system S. Recall that Ψ denotes the set of all orientations in $G_{\mathbb{R}}$ in which each node has at most one incoming edge. We prove the lemma via the following claim.

▶ **Claim 7.** *Let S be a system.*

- *Suppose that S is bad. Then*

$$W \cdot (\min_{\psi \in \Psi} \textit{number of rocks to } \mathbb{A}(S) \textit{ according to } \psi) + \sum_{v \in \mathbb{A}(S)} pl(v) + dl(v) > |\mathbb{A}(S)|t. \quad (1)$$

- *Suppose that S is good. Then*

$$W \cdot (\min_{\psi \in \Psi} \textit{number of rocks to } \mathbb{A}(S) \textit{ according to } \psi) + \sum_{v \in \mathbb{A}(S)} pl(v) + dl(v) > |\mathbb{A}(S)|t - w. \quad (2)$$

Observe that the term $|\mathbb{A}(S)|t$ is the maximum total weight that all nodes in $\mathbb{A}(S)$ can handle if OPT $\leq t$. As pebbles owned by nodes in $\mathbb{A}$ can only be assigned to the nodes in $\mathbb{A}$, by the pigeonhole principle, in all orientations $\psi \in \Psi$, and all possible assignments of the pebbles, at least one bad system S has at least the same number of pebbles in $\mathbb{A}(S)$ as the current assignment, or a good system S has at least one more pebble than it currently has in $\mathbb{A}(S)$. In both cases, we reach a contradiction.

Proof of Claim 7. First observe that in all orientations in Ψ, the nodes in $\mathbb{A}(S)$ have to receive at least $|\mathbb{A}(S)| - 1$ rocks. If S is a cycle, then the nodes in $\mathbb{A}(S)$ have to receive exactly $|\mathbb{A}(S)|$ rocks.

Next observe that none of the nodes in $\mathbb{A}(S)$ is uncritical, since otherwise, by Proposition 4(4) and Definition 5(3), the Push operation would still be possible. By the same reasoning, if S is a tree and $\mathbb{A}(S) \neq \emptyset$, at least one node $v \in \mathbb{A}(S)$ is critical; furthermore, if $|\mathbb{A}(S)| = 1$, this node v satisfies $dl(v) + pl(v) > 1.5t - w$, as an additional weight of w would make v hypercritical. Similarly, if S is an isolated node $v \in \mathbb{A}$, then $dl(v) + pl(v) > 1.5t - w$.

We now prove the claim by the following case analysis.

1. Suppose that S is a good system and $\mathbb{A}(S) \neq \emptyset$. Then either S is a tree and $\mathbb{A}(S)$ contains exactly one critical (but not hypercritical) node, or S is an isolated node, or S is a cycle and has no critical node. In the first case, if $|\mathbb{A}(S)| \geq 2$, the LHS of (2) is at least
$$(1.5t - W + 1) + (|\mathbb{A}(S)| - 1)(1.5t - W - w + 1) + (|\mathbb{A}(S)| - 1)W =$$
$$|\mathbb{A}(S)|t + (|\mathbb{A}(S)| - 2)(0.5t - w + 1) + t - W - w + 2 > |\mathbb{A}(S)|t - w,$$
using the fact that $0.5t \geq w$, $t \geq W$, and $|\mathbb{A}(S)| \geq 2$. If, on the other hand, $|\mathbb{A}(S)| = 1$, then the LHS of (2) is strictly more than
$$1.5t - w \geq t = |\mathbb{A}(S)|t,$$
and the same also holds for the case when S is an isolated node. Finally, in the third case, the LHS of (2) is at least
$$|\mathbb{A}(S)|(1.5t - W - w + 1) + |\mathbb{A}(S)|W > |\mathbb{A}(S)|t.$$
2. Suppose that $\mathbb{A}(S)$ contains at least two critical nodes, or that S is a cycle and $\mathbb{A}(S)$ has at least one critical node. In both cases, S is a bad system. Furthermore, the LHS of (1) can be lower-bounded by the same calculation as in the previous case with an extra term of w.
3. Suppose that $\mathbb{A}(S)$ contains a hypercritical node. Then the system S is bad, and the LHS of (1) is at least
$$(1.5t + 1) + (|\mathbb{A}(S)| - 1)(1.5t - W - w + 1) + (|\mathbb{A}(S)| - 1)W =$$
$$|\mathbb{A}(S)|t + (|\mathbb{A}(S)| - 1)(0.5t - w + 1) + 0.5t + 1 > |\mathbb{A}(S)|t,$$
where the inequality holds because $0.5t \geq w$. ◀

◀

We argue that Algorithm 1 terminates in polynomial time by the aid of a potential function, defined as
$$\Phi = \sum_{v \in \mathbb{A}} (|V| - \textsc{Level}(v)) \cdot (\text{number of pebbles at } v).$$

Trivially, $0 \leq \Phi \leq |V| \cdot |\mathbb{P}|$. The next lemma implies that Φ is monotonically decreasing after each Push operation. The proof can be found in the full version.

▶ **Lemma 8.** *For each node $v \in V$, let $\textsc{Level}(v)$ and $\textsc{Level}'(v)$ denote the levels before and after a* Push *operation, respectively. Then $\textsc{Level}'(v) \geq \textsc{Level}(v)$.*

Proof. We prove by contradiction. Suppose that there exist nodes x with $\textsc{Level}'(x) < \textsc{Level}(x)$. Choose v to be one among them with minimum $\textsc{Level}'(v)$. By the choice of v, and Definition 5(3), $\textsc{Level}'(v) > 0$ and $v \in \mathbb{A}$ after the Push operation. Thus, by Proposition 4(4), there exists a node u with $\textsc{Level}'(u) = \textsc{Level}'(v) - 1$, so that after Push,

- Case 1: v is reachable from $u \in \mathbb{A}$, or
- Case 2: there exists another node $v' \in \mathbb{A}$ reachable from $u \in \mathbb{A}$ with $\textsc{Level}'(v') = \textsc{Level}'(v)$ in the same system as v, and v is critical.

Notice that by the choice of v, in both cases, $\textsc{Level}'(u) \geq \textsc{Level}(u)$, and $u \in \mathbb{A}$ also before the Push operation. Let p be the pebble by which u reaches v (Case 1), or v' (Case 2), after Push. Before the Push operation, p was at some node $u' \in \mathbb{A}$ (u' may be u, or p is the pebble pushed: from u' to u).

By Proposition 4(2), in Case 1, $\textsc{Level}(v) \leq \textsc{Level}(u') + 1$ (as v is reachable from u' via p before Push), and $\textsc{Level}(v') \leq \textsc{Level}(u') + 1$ in Case 2. Furthermore, if in Case 2 v was already critical before push, then $\textsc{Level}(v) \leq \textsc{Level}(v')$ by Proposition 4(3) (note that $v' \in \mathbb{A}$ as it is reachable from $u' \in \mathbb{A}$). Hence, in both cases we would have

$$\textsc{Level}(v) \leq \textsc{Level}(u') + 1 \leq \textsc{Level}(u) + 1 \leq \textsc{Level}'(u) + 1 = \textsc{Level}'(v),$$

a contradiction. Note that the second inequality holds no matter $u = u'$ or not.

Finally consider Case 2 where v was not critical before the Push operation. Then a pebble $p' \neq p$ is pushed into v in the operation. Note that in this situation, v's system is a tree and contains no critical nodes before Push (by Definition 5(3)); in particular v' is not critical. Furthermore, the presence of p in u implies that $\textsc{Level}(v') \leq \textsc{Level}(u) + 1$ by Proposition 4(2), and that $v' \in \mathbb{A}$ by Proposition 4(1). As v' is not critical, $\textsc{Level}(v') > 0$, and by Proposition 4(4) there exists a node u'' with $\textsc{Level}(u'') = \textsc{Level}(v') - 1$ so that u'' can reach v' by a pebble p'' (u'' may be u and p'' may be p). As

$$\textsc{Level}(v') \leq \textsc{Level}(u) + 1 \leq \textsc{Level}'(u) + 1 = \textsc{Level}'(v) < \textsc{Level}(v),$$

the Push operation should have pushed p'' into v' instead of p' into v (see Definition 5(4)), since u'' and v' satisfy all the first three conditions of Definition 5. ◀

By Lemma 8 and the fact that a pebble is pushed to a node with higher level, the potential Φ strictly decreases after each Push operation, implying that Algorithm 1 finishes in polynomial time.

Approximation Ratio: When $t < 2W$, we apply Algorithm 1. In the case of $t \geq 2W$, we apply the algorithm of Gairing et al. [5], which either correctly reports that $\text{OPT} \geq t + 1$, or returns an assignment with makespan at most $t + W - 1 < 1.5t$.

Suppose that t is the smallest number for which an assignment is returned. Then $\text{OPT} \geq t$, and our approximation ratio is bounded by $\frac{1.5t}{\text{OPT}} \leq 1.5$. We use a theorem to conclude this section.

▶ **Theorem 9.** *With arbitrary dedicated loads on the machines, jobs of weight W that can be assigned to two machines, and jobs of weight w that can be assigned to any number of machines, we can find a 1.5 approximate solution in polynomial time.*

In the full version, we show that a slight modification of our algorithm yields an improved approximation ratio of $1 + \frac{\lfloor \frac{W}{2} \rfloor}{W}$ if $W \geq 2w$.

4 The General Case

In this section, we describe the core algorithm for the case of arbitrary job weights. This algorithm inherits some basic ideas from the previous section, but has several significantly new ingredients – mainly due to the fact that the rocks now have different weights. Before formally presenting the algorithm, let us build up intuition by looking at some examples.

For simplicity, we rescale the numbers and assume that $t = W = 1$ and $\beta = 0.7$. We aim for an assignment with makespan of at most $5/3 + 0.7/3 = 1.9$ or decide that $\text{OPT} > 1$.

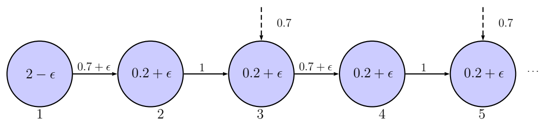


Figure 2 There are $2k+1$ nodes (the rest is repeating the same pattern). Numbers inside the shaded circles (nodes) are their pebble load.

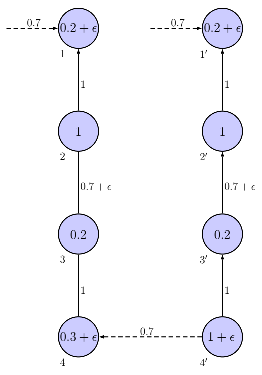


Figure 3 A naive PUSH will oscillate the pebble between nodes 4 and $4'$.

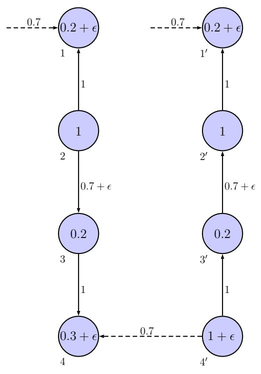


Figure 4 A fake orientation from node 2 to 3 causes node 4 to have an incoming edge, thus informing node $4'$ not to push the pebble.

Consider the example in Figure 2. Note that there are $2k+1$ (for some large k) nodes (the pattern of the last two nodes repeats). Due to node 1 (which can be regarded as the analog of a critical node in the previous section), all edges are to be directed toward the right if we shoot for the makespan of 1.9. Suppose that there is an isolated node with the pebble load of $2+\epsilon$ (this node can be regarded as a bad system by itself) and it has a pebble of weight 0.7 that can be assigned to node 3, 5, 7 and so on up to $2k+1$. Clearly, we do not want to push the pebble into any of them, as it would cause the makespan to be larger than 1.9 by whatever orientation. Rather, we should activate node 1 and send its pebbles away with the aim of relieving the "congestion" in the current system (later we will see that this is activation rule 1). In this example, all odd-numbered nodes are activated, and the entire set of nodes (including even-numbered nodes) form a *conflict set* (which will be defined formally later). Roughly speaking, the conflict sets contain activated nodes and the nodes that can be reached by "backtracking" the directed edges from them. These conflict sets embody the "congestion" in the systems.

Recall that in the previous section, if the PUSH operation was no longer possible, we argued that the total load is too much for the activated nodes *system by system*. Analogously, in this example, we need to argue that in all feasible orientations, the activated set of nodes (totally $k+1$ of them) in this conflict set cannot handle the total load. However, if all edges are directed toward the left, their total load is only $(0.2+\epsilon)k+(2-\epsilon)+(0.7+\epsilon)k = 2+0.9k+\epsilon(2k-1)$, which is less than what they can handle (which is $k+1$) when k is large. As a result, we are unable to arrive at a contradiction.

To overcome this issue, we introduce another activation rule to strengthen our contradiction argument. If all edges are directed to the left, *on the average*, each activated node has a total load of about $0.2+0.7$. However, each inactivated node has, *on the average*, a total load of about $0.2+1$. This motivates our activation rule 2 : if an activated node is connected by a "relatively light" edge to some other node in the conflict set, the latter should be activated as well. The intuition behind is that the two nodes *together* will receive a relatively heavy load. We remark that it is easy to modify this example to show that if we do not apply activation rule 2, then we cannot hope for a $2 - \delta$ approximation for any small $\delta > 0$. [3]

Next consider the example in Figure 3. Here nodes 2, $2'$, and $4'$ can be regarded as the critical nodes, and $\{1, 2\}$, $\{1', 2', 3', 4'\}$ are the two conflict sets. Both nodes 1 and $1'$ can be reached by an isolated node with heavy load (the bad system) with a pebble of weight 0.7. Suppose further that node $4'$ can reach node 4 by another pebble of weight 0.7. It is easy to see that a naive PUSH definition will simply "oscillate" the pebble between nodes 4 and $4'$, causing the algorithm to cycle.

Intuitively, it is not right to push the pebble from $4'$ into 4, as it causes the conflict set in the left system to become bigger. Our principle of pushing a pebble should be to relieve the congestion in one system, while not worsening the congestion in another. To cope with this problematic case, we use *fake orientations*, i.e., we direct edges away from a conflict set, as shown in Figure 4. Node 2 directs the edge toward node 3, which in turn causes the next edge to be directed toward node 4. With the new incoming edge, node 4 now has a total load of $1 + 0.3 + \epsilon$ to handle, and the pebble thus will not be pushed from node $4'$ to node 4.

4.1 Formal description of the algorithm

We inherit some terminology from the previous section. We say that v is *reachable* from u if a pebble in u can be assigned to v, and that v is reachable from $\mathbb{A}$ if v is reachable from any node $u \in \mathbb{A}$. Each tree, cycle, isolated node in $G_{\mathbb{R}}$ is a system. Note that there is exactly one edge between two adjacent nodes in $G_{\mathbb{R}}$ (see Proposition 1). For ease of presentation, we use the short hand vu to refer to the edge $\{v, u\}$ in $G_{\mathbb{R}}$ and w_{vu} is its weight.

The orientation of the edges in $G_{\mathbb{R}}$ will be decided dynamically. If uv is directed toward v, we call v a *father* of u, and u a *child* of v (notice that a node can have several fathers and children). We write $rl(v)$ to denote total weight of the rocks that are (currently) oriented towards v, and $pl(v)$ still denotes the total weight of the pebbles at v. An edge that is currently un-oriented is *neutral*. In the beginning, all edges in $G_{\mathbb{R}}$ are neutral.

A set $\mathbb{C}$ of nodes, called the *conflict set*, will be collected in the course of the algorithm. Let $\mathcal{D}(v) := \{u \in \mathbb{C} : u \text{ is child of } v\}$ and $\mathcal{F}(v) := \{u \in \mathbb{C} : u \text{ is father of } v\}$ for any $v \in \mathbb{C}$. A node $v \in \mathbb{C}$ is a *leaf* if $\mathcal{D}(v) = \emptyset$, and a *root* if $\mathcal{F}(v) = \emptyset$. Furthermore, a node v is *overloaded* if $dl(v) + pl(v) + rl(v) > (5/3 + \beta/3)t$, and a node $v \in \mathbb{C}$ is *critical* if there exists $u \in \mathcal{F}(v)$ such that $dl(v) + pl(v) + w_{vu} > (5/3 + \beta/3)t$. In other words, a node in the conflict set is critical if it has enough load by itself (without considering incoming rocks) to "force" an incident edge to be directed toward a father in the conflict set.

Initially, the pebbles are arbitrarily assigned to the nodes. The orientation of a subset of the edges in $G_{\mathbb{R}}$ is determined by the procedure FORCED ORIENTATIONS in Figure 5.

Intuitively, the procedure first finds a "source node" v, whose dedicated, pebble, and rock load is so high that it "forces" an incident edge vu to be oriented away from v. The orientation of this edge then propagates through the graph, i.e. edge-orientations induced by

[3] Looking at this particular example, one is tempted to use the idea of activating all nodes in the conflict set. However, such an activation rule will not work. Consider the following example: There are $k + 2$ nodes forming a path, and the $k + 1$ edges connecting them all have weight $0.95 + \epsilon$. The first node has a pebble load of 1 and thus "forces" an orientation of the entire path (for a makespan of at most 1.9). The next k nodes have a pebble load of 0, and the last node has a pebble load of 0.25 and is reachable from a bad system via a pebble of weight 0.7. The conflict set is the entire path, and activating all nodes leads to a total load of $(k + 1) \cdot (0.95 + \epsilon) + 1 + 0.25$, which is less than $k + 2$ for large k.

```
FORCED ORIENTATIONS
While ∃ neutral edge $vu$ in $G_{\mathbb{R}}$, s.t. $dl(v) + pl(v) + rl(v) + w_{vu} > (5/3 + \beta/3)t$:
    Direct $vu$ towards $u$; MARKED := $\{u\}$.
    While ∃ neutral edge $v'u'$ in $G_{\mathbb{R}}$, s.t. $dl(v') + pl(v') + rl(v') + w_{v'u'} > (5/3 + \beta/3)t$ and $v' \in$ MARKED:
        Direct $v'u'$ towards $u'$; MARKED := MARKED $\cup \{u'\}$.
```

Figure 5 The procedure FORCED ORIENTATIONS.

the direction of vu are established. Then the next "source" is found, and so on. To simplify our proofs, we assume that ties are broken according to a fixed total order if several pairs (v, u) satisfy the conditions of the while-loops.

The following lemma describes a basic property of the procedure FORCED ORIENTATIONS, that will be used in the subsequent discussion. Its proof can be found in the full version.

▶ **Lemma 10.** *Suppose that a node v becomes overloaded during* FORCED ORIENTATIONS. *Then there exists a path $u_0u_1 \dots u_kv$ of neutral edges, such that $dl(u_0) + pl(u_0) + rl(u_0) + w_{u_0u_1} > (5/3 + \beta/3)t$ before the procedure, that becomes directed from u_0 towards v during the procedure (note that u_0 could be v). Furthermore, other than u_kv, no edge becomes directed toward v in the procedure.*

Proof. We start with a simple observation. Let ab be the first edge directed in some iteration of the procedure's outer while-loop; suppose from a to b. It is easy to see that up to this moment, no edge has been directed toward a in course of the procedure. Furthermore, if another edge $a'b'$ is directed in the same iteration of the outer while-loop, then there exists a path of neutral edges, starting with ab and ending with $a'b'$, that becomes directed during this iteration. This proves the first part of the lemma.

Now suppose that some node v becomes overloaded and has more than one edge directed towards it during the procedure. Let vx and vy be the last two edges directed toward v, and note that both, vx and vy, become directed in the same iteration of the outer while-loop (because as soon as one of the two is directed toward v, the other edge satisfies the conditions of the inner while-loop). Hence, there are two different paths directed towards v (with final edges vx and vy, respectively), both of which start with the first edge that becomes directed in this iteration of the outer while-loop. This is not possible, since every system is a tree or a cycle, a contradiction. ◀

Clearly, if after the procedure FORCED ORIENTATIONS a node v still has a neutral incident edge vu, then $dl(v)+pl(v)+rl(v)+w_{vu} \le (5/3+\beta/3)t$. Now suppose that after the procedure, none of the nodes is overloaded. Then orienting the neutral edges in each system in such a way that every node has at most one more incoming edge gives us a solution with makespan at most $(5/3 + \beta/3)t$. So assume the procedure ends with a non-empty set of overloaded nodes. We then apply the procedure EXPLORE2 in Figure 6.

Let us elaborate the procedure. In each round, we perform the following three tasks.

1. Add those nodes reachable from the nodes in $\mathbb{A}_{i-1}$ into $\mathbb{A}_i$ in case of $i > 1$; or the overloaded nodes into $\mathbb{A}_i$ in case of $i = 0$. These nodes will be referred to as Type A nodes.
2. In the sub-procedure *Conflict set construction*, nodes not in the conflict set and having a directed path to those Type A nodes in $\mathbb{A}_i$ are continuously added into the conflict set

```
EXPLORE2

Initialize $\mathbb{A} := \emptyset$; $\mathbb{C} := \emptyset$; $i := 0$. Call FORCED ORIENTATIONS.
Repeat:
    If $i = 0$: $\mathbb{A}_i := \{v | v$ is overloaded$\}$.
    Else $\mathbb{A}_i := \{v | v \notin \mathbb{A}, v$ is reachable from $\mathbb{A}_{i-1}\}$.
    If $\mathbb{A}_i = \emptyset$: stop.
    $\mathbb{C}_i := \mathbb{A}_i$; $\mathbb{A} := \mathbb{A} \cup \mathbb{A}_i$; $\mathbb{C} := \mathbb{C} \cup \mathbb{C}_i$.

    (Conflict set construction)
    While $\exists v \notin \mathbb{C}$ with a father $u \in \mathbb{C}$ or $\exists$ neutral $vu$ with $v \in \mathbb{C}$ do:
        While $\exists v \notin \mathbb{C}$ with a father $u \in \mathbb{C}$:
            $\mathbb{C}_i := \mathbb{C}_i \cup \{v\}$; $\mathbb{C} := \mathbb{C} \cup \mathbb{C}_i$.
        If $\exists$ neutral $vu$ with $v \in \mathbb{C}$:
            Direct $vu$ towards $u$; Call FORCED ORIENTATIONS.

    (Activation of nodes)
    While $\exists v \in \mathbb{C} \setminus \mathbb{A}$ satisfying one of the following conditions:
        Rule 1: $\exists u \in \mathcal{F}(v)$, such that $dl(v) + pl(v) + w_{vu} > (5/3 + \beta/3)t$
        Rule 2: $\exists u \in \mathbb{A} \cap (\mathcal{D}(v) \cup \mathcal{F}(v))$, such that $w_{vu} < (2/3 + \beta/3)t$
    Do: $\mathbb{A}_i := \mathbb{A}_i \cup \{v\}$; $\mathbb{A} := \mathbb{A} \cup \mathbb{A}_i$.

    $i := i + 1$.
```

Figure 6 The procedure EXPLORE2.

$\mathbb{C}_i$. Furthermore, the earlier mentioned *fake orientations* are applied: each node $v \in \mathbb{C}_i$, if having an incident neutral edge vu, direct it toward u and call the procedure FORCED ORIENTATIONS. It may happen that in this process, two disjoint nodes in $\mathbb{C}_i$ are now connected by a directed path P, then all nodes in P along with all nodes having a path leading to P are added into $\mathbb{C}_i$ (observe that all these nodes have a directed path to some Type A node in $\mathbb{A}_i$). We note that the order of fake orientations does not materially affect the outcome of the algorithm.

3. In the next sub-procedure *Activation of nodes*, we use two rules to activate extra nodes in $\mathbb{C} \setminus \mathbb{A}$. Rule 1 activates the critical nodes; Rule 2 activates those nodes whose father or child are already activated and they are connected by an edge of weight less than $(2/3 + \beta/3)t$. We will refer to the former as Type B nodes and the latter as Type C nodes.

Observe that except in the initial call of FORCED ORIENTATIONS, no node ever becomes overloaded in EXPLORE2 (by Lemma 10 and the fact that every system is a tree or a cycle). Let us define LEVEL$(v) = i$ if $v \in \mathbb{A}_i$. In case $v \notin \mathbb{A}$, let LEVEL$(v) = \infty$. The next proposition summarizes some important properties of the procedure EXPLORE2.

▶ **Proposition 11.** *After the procedure* EXPLORE2, *the following holds.*

1. *All nodes reachable from $\mathbb{A}$ are in $\mathbb{A}$.*
2. *Suppose that $v \in \mathbb{A}$ is reachable from $u \in \mathbb{A}$. Then* LEVEL$(v) \leq$ LEVEL$(u) + 1$.

Furthermore, at the end of each round i, the following holds.

3. *Every node v that can follow a directed path to a node in $\mathbb{C} := \cup_{\tau=0}^{i} \mathbb{C}_\tau$ is in $\mathbb{C}$. Furthermore, if a node $v \in \mathbb{C}$ has an incident edge vu with $u \notin \mathbb{C}$, then vu is directed toward u.*
4. *Each node $v \in \mathbb{A}_i$ is one of the following three types.*

(a) **Type A**: *there exists another node $u \in \mathbb{A}_{i-1}$ so that v is reachable from u, or v is overloaded and is part of $\mathbb{A}_0$.*
(b) **Type B**: *v is activated via Rule 1 (hence v is critical)*[4]*, and there exists a directed path from v to $u \in A_i$ of Type A.*
(c) **Type C**: *v is activated via Rule 2, and there exists an adjacent node $u \in \cup_{\tau=0}^{i} \mathbb{A}_\tau$ so that $w_{vu} < (2/3 + \beta/3)t$ and $u \in \mathcal{D}(v) \cup \mathcal{F}(v)$.*

After the procedure EXPLORE2, we apply the PUSH operation (if possible), defined as follows.

▶ **Definition 12.** PUSH operation: push a pebble from u^* to v^* if the following conditions hold (if there are multiple candidates, pick any).
1. The pebble is at u^* and it can be assigned to v^*.
2. $\text{LEVEL}(v^*) = \text{LEVEL}(u^*) + 1$.
3. $dl(v^*) + pl(v^*) + rl(v^*) \leq (5/3 - 2/3 \cdot \beta)t.$
4. $\mathcal{D}(v^*) = \emptyset$, or $dl(v^*) + pl(v^*) + w_{v^*u} \leq (5/3 - 2/3 \cdot \beta)t$ for all $u \in \mathcal{F}(v)$.

Definition 12(3) is meant to make sure that v^* does not become overloaded after receiving a new pebble (whose weight can be as heavy as βt). Definition 12(4) says either v^* is a leaf, or adding a pebble with weight as heavy as βt does not cause v^* to become critical.

Algorithm 2: Apply EXPLORE2. If it ends with $\mathbb{A}_0 = \emptyset$, return a solution with makespan at most $(5/3 + \beta/3)t$. Otherwise, apply PUSH. If PUSH is impossible, declare that OPT $\geq t + 1$. Un-orient all edges in $G_{\mathbb{R}}$ and repeat this process.

▶ **Lemma 13.** *When there is at least one overloaded node and the* PUSH *operation is no longer possible,* OPT $\geq t + 1$.

▶ **Lemma 14.** *For each node $v \in V$, let* $\text{LEVEL}(v)$ *and* $\text{LEVEL}'(v)$ *denote the levels before and after a* PUSH *operation, respectively. Then* $\text{LEVEL}'(v) \geq \text{LEVEL}(v)$.

The proofs of the preceding two lemmas can be found in the full version. We again use the potential function

$$\Phi = \sum_{v \in \mathbb{A}} (|V| - \text{LEVEL}(v)) \cdot (\text{number of pebbles at } v)$$

to argue the polynomial running time of Algorithm 2. Trivially, $0 \leq \Phi \leq |V| \cdot |\mathbb{P}|$. Furthermore, by Lemma 14 and the fact that a pebble is pushed to a node with higher level, the potential Φ strictly decreases after each PUSH operation. This implies that Algorithm 2 finishes in polynomial time.

We can therefore conclude:

▶ **Theorem 15.** *Let $\beta \in [4/7, 1)$. With arbitrary dedicated loads on the machines, if jobs of weight greater than βW can be assigned to only two machines, and jobs of weight at most βW can be assigned to any number of machines, we can find a $5/3 + \beta/3$ approximate solution in polynomial time.*

[4] For simplicity, if a node can be activated by both Rule 1 and Rule 2, we assume it is activated by Rule 1.

References

1 Yuichi Asahiro, Jesper Jansson, Eiji Miyano, Hirotaka Ono, and Kouhei Zenmyo. Approximation algorithms for the graph orientation minimizing the maximum weighted outdegree. *J. Comb. Optim.*, 22(1):78–96, 2011.

2 Deeparnab Chakrabarty, Sanjeev Khanna, and Shi Li. On $(1, \epsilon)$-restricted assignment makespan minimization. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'15)*, pages 1087–1101. SIAM, 2015.

3 T. Ebenlendr, M. Krčál, and J. Sgall. Graph balancing: A special case of scheduling unrelated parallel machines. *Algorithmica*, 68:62–80, 2014.

4 Tomáš Ebenlendr, Marek Krčál, and Jiří Sgall. Graph balancing: A special case of scheduling unrelated parallel machines. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'08)*, pages 483–490. SIAM, 2008.

5 M. Gairing, T. Lücking, M. Mavronicolas, and B. Monien. Computing nash equilibria for scheduling on restricted parallel links. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC'04)*, pages 613–622. ACM, 2004.

6 M. Gairing, B. Monien, and A. Wocalw. A faster combinatorial approximation algorithm for scheduling unrelated parallel machines. *Theor. Comput. Sci.*, 380(1-2):87–99, 2007.

7 Chien-Chung Huang and Sebastian Ott. A combinatorial approximation algorithm for graph balancing with light hyper edges. *CoRR*, abs/1507.07396, 2015.

8 S. G. Kolliopoulos and Y. Moysoglou. The 2-valued case of makespan minimization with assignment constraints. *Information Processing Letters*, 113(1-2):39–43, 2013.

9 J.K. Lenstra, D.B. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:256–271, 1990.

10 J.Y. Leung and C. Li. Scheduling with processing set restrictions: A survey. *International Journal of Production Economics*, 116:251–262, 2008.

11 J. Sgall. Private communication, 2015.

12 E. V. Shchepin and N. Vakhania. An optimal rounding gives a better approximation for scheduling unrelated machines. *Oper. Res. Lett.*, 33(2):127–133, 2005.

13 O. Svensson. Santa claus schedules jobs on unrelated machines. *SIAM J. Comput.*, 41(5):1318–1341, 2012.

14 J. Verschae and A. Wiese. On the configuration-lp for scheduling on unrelated machines. *J. Scheduling*, 17(4):371–383, 2014.

15 D. P. Willamson and D.B. Shmoys. *The design of approximation algorithms.* Cambridge University Press, 2010.

ε-Kernel Coresets for Stochastic Points

Lingxiao Huang*[1], Jian Li[2], Jeff M. Phillips†[3], and Haitao Wang‡[4]

1 **Tsinghua University, Beijing, China**
2 **Tsinghua University, Beijing, China**
`lijian83@mail.tsinghua.edu.cn`
3 **University of Utah, Salt Lake City, UT, USA**
`jeffp@cs.utah.edu`
4 **Utah State University, Logan, UT, USA**
`haitao.wang@usu.edu`

Abstract

With the dramatic growth in the number of application domains that generate probabilistic, noisy and uncertain data, there has been an increasing interest in designing algorithms for geometric or combinatorial optimization problems over such data. In this paper, we initiate the study of constructing ε-kernel coresets for uncertain points. We consider uncertainty in the existential model where each point's location is fixed but only occurs with a certain probability, and the locational model where each point has a probability distribution describing its location. An ε-kernel coreset approximates the width of a point set in any direction. We consider approximating the expected width (an ε-EXP-KERNEL), as well as the probability distribution on the width (an (ε, τ)-QUANT-KERNEL) for any direction. We show that there exists a set of $O(\varepsilon^{-(d-1)/2})$ deterministic points which approximate the expected width under the existential and locational models, and we provide efficient algorithms for constructing such coresets. We show, however, it is not always possible to find a subset of the original uncertain points which provides such an approximation. However, if the existential probability of each point is lower bounded by a constant, an ε-EXP-KERNEL is still possible. We also provide efficient algorithms for construct an (ε, τ)-QUANT-KERNEL coreset in nearly linear time. Our techniques utilize or connect to several important notions in probability and geometry, such as Kolmogorov distances, VC uniform convergence and Tukey depth, and may be useful in other geometric optimization problem in stochastic settings. Finally, combining with known techniques, we show a few applications to approximating the extent of uncertain functions, maintaining extent measures for stochastic moving points and some shape fitting problems under uncertainty.

1998 ACM Subject Classification B.2.4 Algorithms, F.2.2 Geometrical problems and computations

Keywords and phrases ε-kernel, coreset, stochastic point, shape fitting

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.50

1 Introduction

Uncertain Data Models: The wide deployment of sensor monitoring infrastructure and increasing prevalence of technologies such as data integration and cleaning [15, 18] have

* LXH and JL are supported in part by the National Basic Research Program of China grants 2015CB358700, 2011CBA00300, 2011CBA00301, and the National NSFC grants 61033001, 61361136003.
† JP thanks support from NSF CCF-1350888, IIS-1251019, ACI-1443046, and CNS-1514520.
‡ HW was supported in part by NSF under Grant CCF-1317143.

resulted in an abundance of uncertain, noisy and probabilistic data in many scientific and application domains. Managing, analyzing, and solving optimization problems over such data have become an increasingly important issue and have attracted significant attentions from several research communities including theoretical computer science, databases, machine learning and wireless networks.

In this paper, we focus on two stochastic models, the existential model and locational model. Both models have been studied extensively for a variety of geometric or combinatorial problems, such as closest pairs [34], nearest neighbors [5, 34], minimum spanning trees [32, 35], convex hulls [6, 24, 49, 52], maxima [3], perfect matchings [32], clustering [16, 26] , minimum enclosing balls [43] and range queries [1, 4, 38].

1. Existential uncertainty model: In this model, there is a set $\mathcal{P}$ of n points in $\mathbb{R}^d$. Throughout this paper, we assume that the dimension d is a constant. Each point $v \in \mathcal{P}$ is associated with a real number (called *existential probability*) $p_v \in [0, 1]$ which indicates that v is present independently with probability p_v.
2. Locational uncertainty model: There are a set $\mathcal{P}$ of n points and the existence of each point is certain. However, the location of each point $v \in \mathcal{P}$ is a random location in $\mathbb{R}^d$. We assume the probability distribution is discrete and independent of other points. For a point $v \in \mathcal{P}$ and a location $s \in \mathbb{R}^d$, we use $p_{v,s}$ to denote the probability that the location of point v is s. Let S be the set of all possible locations, and let $|S| = m$ be the number of all such locations.

In the locational uncertainty model, we distinguish the use of the terms "points" and "locations"; a point refers to the object with uncertain locations and a location refers to a point (in the usual sense) in $\mathbb{R}^d$. We will use capital letters (e.g., $P, S, \ldots$) to denote sets of deterministic points and calligraphy letters ($\mathcal{P}, \mathcal{S}, \ldots$) to denote sets of stochastic points.

Coresets: Given a large dataset P and a class C of queries, a coreset S is a dataset of much smaller size such that for every query $r \in C$, the answer $r(S)$ for the small dataset S is close to the answer $r(P)$ for the original large dataset P. Coresets [46] have become more relevant in the era of big data as they can drastically reduce the size of a dataset while guaranteeing that answers for certain queries are provably close. An early notion of a coreset concerns the directional width problem (in which a coreset is called an ε-kernel) and several other geometric shape-fitting problems in the seminal paper [7].

We introduce some notations and review the definition of ε-kernel. We assume the dimension d is a constant. For a set P of deterministic points, the *support function* $f(P, u)$ is defined to be $f(P, u) = \max_{p \in P}\langle u, p\rangle$ for $u \in \mathbb{R}^d$, where $\langle ., .\rangle$ is the inner product. The *directional width* of P in direction $u \in \mathbb{R}^d$, denoted by $\omega(P, u)$, is defined by $\omega(P, u) = f(P, u) + f(P, -u)$. It is not hard to see that the support function and the directional width only depend on the convex hull of P. A subset $Q \subseteq P$ is called an *ε-kernel of P* if for each direction $u \in \mathbb{R}^d$, $(1 - \varepsilon)\omega(P, u) \leq \omega(Q, u) \leq \omega(P, u)$. For any set of n points, there is an ε-kernel of size $O(\varepsilon^{-(d-1)/2})$ [7, 8], which can be constructed in $O(n+\varepsilon^{-(d-3/2)})$ time [13, 53].

1.1 Problem Formulations

We focus on constructing ε-kernel coresets when the input data is uncertain. These results not only provide an understanding of how to compactly represent an approximate convex hull under uncertainty, but can lead to solutions to a variety of other shape-fitting problems.

ε-Kernels for expectations. Suppose $\mathcal{P}$ is a set of stochastic points (in either the existential or locational uncertainty model). Define the *expected* directional width of $\mathcal{P}$ in direction u to be $\omega(\mathcal{P}, u) = \mathbb{E}_{P\sim\mathcal{P}}[\omega(P, u)]$, where $P \sim \mathcal{P}$ means that P is a (random) realization of $\mathcal{P}$.

▶ **Definition 1.** For a constant $\varepsilon > 0$, a set S of (deterministic or stochastic) points in $\mathbb{R}^d$ is called an ε-EXP-KERNEL of $\mathcal{P}$, if for all directions $u \in \mathbb{R}^d$,

$$(1-\varepsilon)\omega(\mathcal{P}, u) \le \omega(S, u) \le \omega(\mathcal{P}, u).$$

Recall in the deterministic setting, we require that the ε-kernel S be a subset of the original point set (we call this *the subset constraint*). It is important to consider the subset constraint since it can reveal how concisely arbitrary uncertain point sets can be represented with just a few uncertain points (size depending only on ε). For ε-kernels on deterministic points, $\Omega(\varepsilon^{-(d-1)/2})$ points may be required and can always be found under the subset constraint [7, 8]. However, in the stochastic setting, we will show this is no longer true. Yet, coresets without the subset constraint, in fact made of deterministic points, can sometimes be obtained when no coreset with the subset constraint is possible.

ε-Kernels for probability distributions. Sometimes it is useful to obtain more than just the expected value (say of the width) on a query; rather one may want (an approximation of) a representation of the full probability distribution that the query can take.

▶ **Definition 2.** For a constant $\varepsilon, \tau > 0$, a set $\mathcal{S}$ of stochastic points in $\mathbb{R}^d$ is called an (ε, τ)-QUANT-KERNEL of $\mathcal{P}$, if for all directions u and all $x \ge 0$,

$$\begin{aligned}\Pr_{P\sim\mathcal{P}}\Big[\omega(P,u) \le (1-\varepsilon)x\Big] - \tau \le \Pr_{S\sim\mathcal{S}}\Big[\omega(S,u) \le x\Big] \\ \le \Pr_{P\sim\mathcal{P}}\Big[\omega(P,u) \le (1+\varepsilon)x\Big] + \tau. \qquad (1)\end{aligned}$$

In the above definition, we do not require the points in $\mathcal{S}$ are independent. So when they are correlated, we will specify the distribution of $\mathcal{S}$. If all points in $\mathcal{P}$ are deterministic and $\tau < 0.5$, the above definition essentially boils down to requiring $(1-\varepsilon)\omega(\mathcal{P}, u) \le \omega(\mathcal{S}, u) \le (1+\varepsilon)\omega(\mathcal{P}, u)$. Assuming the coordinates of the input points are bounded, an (ε, τ)-QUANT-KERNEL ensures that for any choice of u, the cumulative distribution function of $\omega(\mathcal{S}, u)$ is within a distance ε under the Lévy metric, to that of $\omega(\mathcal{P}, u)$.

ε-Kernels for expected fractional powers. Sometimes, the notion ε-EXP-KERNEL is not powerful enough for certain shape fitting problems (e.g., the minimum enclosing cylinder problem and the minimum spherical shell problem) in the stochastic setting. The main reason is the appearance of the l_2-norm in the objective function. So we need to be able to handle the fractional powers in the objective function. For a set P of points in $\mathbb{R}^d$, the polar set of P is defined to be $P^\star = \{u \in \mathbb{R}^d \mid \langle u, v\rangle \ge 0, \forall v \in P\}$. Let r be a positive integer. Given a set P of points in $\mathbb{R}^d$ and $u \in P^\star$, we define a function

$$T_r(P, u) = \max_{v\in P}\langle u, v\rangle^{1/r} - \min_{v\in P}\langle u, v\rangle^{1/r}.$$

We only care about the directions in $\mathcal{P}^\star$ (i.e., the polar of the points in $\mathcal{P}$) for which $T_r(P, u), \forall P \sim \mathcal{P}$ is well defined.

▶ **Definition 3.** For a constant $\varepsilon > 0$, a positive integer r, a set $\mathcal{S}$ of stochastic points in $\mathbb{R}^d$ is called an (ε, r)-FPOW-KERNEL of $\mathcal{P}$, if for all directions $u \in \mathcal{P}^\star$,

$$(1-\varepsilon)\mathbb{E}_{P\sim\mathcal{P}}[T_r(P,u)] \le \mathbb{E}_{P\sim\mathcal{S}}[T_r(P,u)] \le (1+\varepsilon)\mathbb{E}_{P\sim\mathcal{P}}[T_r(P,u)].$$

1.2 Our Results

Now, we discuss the main technical results of the paper.

ε-Kernels for expectations. First, we consider ε-EXP-KERNELs under various constraints. Our first main result is that an ε-EXP-KERNEL of size $O(\varepsilon^{-(d-1)/2})$ exists for both existential and locational uncertainty models and can be constructed in nearly linear time.

▶ **Theorem 4.** *$\mathcal{P}$ is a set of n uncertain points in $\mathbb{R}^d$ (in either locational uncertainty model or existential uncertainty model). There exists an ε-EXP-KERNEL of size $O(\varepsilon^{-(d-1)/2})$ for $\mathcal{P}$. For existential uncertainty model (locational uncertainty model resp.), such an ε-EXP-KERNEL can be constructed in $O(\varepsilon^{-(d-1)}n\log n)$ time, ($O(\varepsilon^{-(d-1)}m\log m)$ time resp.), where n is the number of points and m is the total number of possible locations.*

The existential result is a simple Minkowski sum argument. We first show that there exists a convex polytope M such that for any direction, the directional width of M is exactly the same as the expected directional width of $\mathcal{P}$ (Lemma 10). This immediately implies the existence of a ε-EXP-KERNEL consisting $O(\varepsilon^{-(d-1)/2})$ deterministic points (using the result in [7]), but without the subset constraint. The Minkowski sum argument seems to suggest that the complexity of M is exponential. However, we show that the complexity of M is in fact polynomial $O(n^{2d-2})$ and we can construct it explicitly in $O(n^{2d-1}\log n)$ time (Theorem 14).

Although the complexity of M is polynomial, we cannot afford to construct it explicitly if we are to construct an ε-EXP-KERNEL in nearly linear time. Thus we construct the ε-EXP-KERNEL without explicitly constructing M. In particular, we show that it is possible to find the extreme vertex of M in a given direction in nearly linear time, by computing the gradient of the support function of M. We also provide quadratic-size data structures that can calculate the exact width $\omega(\mathcal{P},\cdot)$ in logarithmic time under both models in $\mathbb{R}^2$.

We also show that under subset constraint (i.e., the ε-EXP-KERNEL is required to be a subset of the original point set, with the same probability distribution for each chosen point), there is no ε-EXP-KERNEL of sublinear size (Lemma 15). However, if there is a constant lower bound $\beta > 0$ on the existential probabilities (called β-assumption), we can construct an ε-EXP-KERNEL of constant size (Theorem 16).

ε-Kernels for probability distributions. Now, we describe our main results for (ε,τ)-QUANT-KERNELs. We first propose a quite simple but general algorithm for constructing (ε,τ)-QUANT-KERNELs, which achieves the following guarantee.

▶ **Theorem 5.** *An (ε,τ)-QUANT-KERNEL of size $\widetilde{O}\left(\tau^{-2}\varepsilon^{-3(d-1)/2}\right)$ can be constructed in $\widetilde{O}\left(n\tau^{-2}\varepsilon^{-(d-1)}\right)$ time, under both existential and locational uncertainty models.*

The algorithm is surprisingly simple. Take a certain number N of i.i.d. realizations, compute an ε-kernel for each realization, and then associate each ε-kernel with probability $1/N$ (so the points are not independent). The analysis requires the VC uniform convergence bound for unions of halfspaces. The details can be found in Section 3.1.

For existential uncertainty model, we can improve the size bound as follows.

▶ **Theorem 6.** *$\mathcal{P}$ is a set of uncertain points in $\mathbb{R}^d$ with existential uncertainty. Let $\lambda = \sum_{v\in\mathcal{P}}(-\ln(1-p_v))$. There exists an (ε,τ)-QUANT-KERNEL for $\mathcal{P}$, which consists of a set of independent uncertain points of cardinality $\min\{\widetilde{O}(\tau^{-2}\max\{\lambda^2,\lambda^4\}),\widetilde{O}(\varepsilon^{-(d-1)}\tau^{-2})\}$. The algorithm for constructing such a coreset runs in $\widetilde{O}(n\log^{O(d)} n)$ time.*

We note that another advantage of the improved construction is that the (ε, τ)-QUANT-KERNEL is a set of independent stochastic points (rather than correlated points as in Theorem 5). We achieve the improvement by two algorithms. The first algorithm transforms the Bernoulli distributed variables into Poisson distributed random variables and creates a probability distribution using the parameters of the Poissons, from which we take a number of i.i.d. samples as the coreset. Our analysis leverages the additivity of Poisson distributions and the VC uniform convergence bound (for halfspaces). However, the number of samples required depends on $\lambda(\mathcal{P})$, so the first algorithm only works when $\lambda(\mathcal{P})$ is small. The second algorithm complements the first one by identifying a convex set K that lies in the convex hull of $\mathcal{P}$ with high probability (K exists when $\lambda(\mathcal{P})$ is large) and uses a small size deterministic ε-kernel to approximate K. The points in $\overline{K} = \mathcal{P} \setminus K$ can be approximated using the same sampling algorithm as in the first algorithm and we can show that $\lambda(\overline{K})$ is small, thus requiring only a small number of samples. In the appendix (Section 3.2.3), we show such an (ε, τ)-QUANT-KERNEL can be computed in $O(n \cdot \text{polylog} n)$ time using an iterative sampling algorithm. Our technique has some interesting connections to other important geometric problems (such as the Tukey depth problem) [42], may be interesting in its own right.

ε-Kernels for expected fractional powers. For (ε, r)-FPOW-KERNELs, we provide a linear time algorithm for constructing an (ε, r)-FPOW-KERNEL of size $\widetilde{O}(\varepsilon^{-(rd-r+2)})$ in the existential uncertainty model under the β-assumption. The algorithm is almost the same as the construction in Section 3.1 except that some parameters are different.

▶ **Theorem 7** (Section 4). *An (ε, r)-FPOW-KERNEL of size $\widetilde{O}(\varepsilon^{-(rd-r+2)})$ can be constructed in $\widetilde{O}\left(n\varepsilon^{-(rd-r+4)/2}\right)$ time in the existential uncertainty model under the β-assumption.*

Applications to Uncertain Function Approximation and Shape Fitting. Finally, we show that the above results, combined with the duality and linearization arguments [7], can be used to obtain constant size coresets for the function extent problem in the stochastic setting, and to maintain extent measures for stochastic moving points.

Using the above results, we also obtain efficient approximation schemes for various shape-fitting problems in the stochastic setting, such as minimum enclosing ball, minimum spherical shell, minimum enclosing cylinder and minimum cylindrical shell in different stochastic settings. We summarize our application results in the following theorems. The details can be found in Section 5.

▶ **Theorem 8.** *Suppose $\mathcal{P}$ is a set of n independent stochastic points in $\mathbb{R}^d$ under either existential or locational uncertainty model. There are linear time approximation schemes for the following problems: (1) finding a center point c to minimize $\mathbb{E}[\max_{v\in\mathcal{P}} \|v-c\|^2]$; (2) finding a center point c to minimize $\mathbb{E}[\mathsf{obj}(c)] = \mathbb{E}[\max_{v\in P} \|v-c\|^2 - \min_{v\in P} \|v-c\|^2]$. Note that when $d = 2$ the above two problems correspond to minimizing the expected areas of the enclosing ball and the enclosing annulus, respectively.*

Under β-assumption, we can obtain efficient approximation schemes for the following shape fitting problems.

▶ **Theorem 9.** *Suppose $\mathcal{P}$ is a set of n independent stochastic points in $\mathbb{R}^d$, each appearing with probability at least β, for some fixed constant $\beta > 0$. There are linear time approximation schemes for minimizing the expected radius (or width) for the minimum spherical shell, minimum enclosing cylinder, minimum cylindrical shell problems over $\mathcal{P}$.*

1.3 Other Related Work

Besides the stochastic models mentioned above, geometric uncertain data has also been studied in the *imprecise* model [11, 31, 36, 40, 44, 45, 50]. In this model, each point is provided with a region where it might be. This originated with the study of imprecision in data representation [27, 47], and can be used to provide upper and lower bounds on several geometric constructs such as the diameter, convex hull, and flow on terrains [19, 50].

Convex hulls have been studied for uncertain points: upper and lower bounds are provided under the imprecise model [20, 41, 44, 50], distributions of circumference and volume are calculated in the locational model [33, 39], the most likely convex hull is found in the existential model in $\mathbb{R}^2$ and shown NP-hard for $\mathbb{R}^d$ for $d > 2$ and in the locational model [49], and the probability a query point is inside the convex hull [6, 24, 52]. As far as we know, the expected complexity of the convex hull under uncertain points has not been studied, although it has been studied [28] under other random data models.

There is a large body of literature [46] on constructing coresets for various problems, such as shape fitting [7, 8], shape fitting with outliers [30], clustering [14, 22, 23, 29, 37], integrals [37], matrix approximation and regression [17, 22] and in different settings, such as geometric data streaming [8, 13] and privacy setting [21]. Coresets were constructed for imprecise points [41] to help derive results for approximating convex hulls and a variety of other shape-fitting problems, but because of the difference in models, these approaches do not translate to existential or locational models. In the locational model, coresets are created for range counting queries [1] under the subset constraint, but again these techniques do not translate because ε-kernel coresets in general cannot be constructed from a density-preserving subset of the data, as is preserved for the range counting coresets. Also in the locational model (and directly translating to the existential model) Löffler and Phillips [39] show how a large set of uncertain points can be approximated with a set of deterministic point sets, where each certain point set can be an ε-kernel. This can provide approximations similar to the (ε, τ)-QUANT-KERNEL with space $O(\varepsilon^{-(d+3)/2} \log(1/\delta))$ with probability at least $1 - \delta$. However it is not a coreset of the data, and answering width queries requires querying $O(\varepsilon^{-2} \log(1/\delta))$ deterministic point sets.

Recently, Munteanu et al. [43] studied the minimum enclosing ball problem over stochastic points, and obtained an efficient approximation scheme. Their algorithm and analysis utilize the results from the deterministic coreset literature [2]. However, they do not directly address the problem of constructing coresets for stochastic points and it is also unclear how to extend their technique to other shape fitting problems, such as minimum spherical shells.

Technically, our (ε, τ)-QUANT-KERNEL construction bears some similarity to the coreset by Har-Peled and Wang [30] for handling outliers. From the dual (function extent) perspective, they want to approximate the distance between two level sets in an arrangement of hyperplanes, and (the dual of) $\mathcal{H}$ in Section 3.2.2 also needs to be (approximately) sandwiched by two fractional level sets (our hyperplanes have weights). However, we have an important requirement that the total weight outside $(1+\varepsilon)\mathcal{H}$ must be small, which cannot be addressed by their technique.

2 ε-Kernels for Expectations of Width

We first state our results in this section for the existential uncertainty model. All results can be extended to the locational uncertainty model, with slightly different bounds (essentially replacing the number of points n with the number of locations m) or assumptions. We describe the difference for locational model in the full version.

For simplicity of exposition, we assume in this section that all points in $\mathcal{P}$ are in general positions and all p_vs are strictly between 0 and 1. For any $u, v \in \mathbb{R}^d$, we use $\langle u, v\rangle$ to denote the usual inner product $\sum_{i=1}^d u_i v_i$. For ease of notation, we write $v \succ_u w$ as a shorthand notation for $\langle u, v\rangle > \langle u, w\rangle$. For any $u \in \mathbb{R}^d$, the binary relation $\succ_u$ defines a total order of all vertices in $\mathcal{P}$. (Ties should be broken in an arbitrary but consistent manner.) We call this order the *canonical order of $\mathcal{P}$ with respect to u*. For any two points u and v, we use $\mathrm{d}(u, v)$ or $\|v - u\|$ to denote their Euclidean distance. For any two sets of points, A and B, the Minkowski sum of A and B is defined as $A \oplus B := \{a + b \mid a \in A, b \in B\}$. Recall the definitions for a set P of deterministic points and a direction $u \in \mathbb{R}^d$, the support function is $f(P, u) = \max_{p\in P}\langle u, p\rangle$ and the *directional width* is $\omega(P, u) = f(P, u) - f(P, -u)$. The support function and the directional width only depend on the convex hull of P.

▶ **Lemma 10.** *Consider a set $\mathcal{P}$ of uncertain points in $\mathbb{R}^d$ (in either locational uncertainty model or existential uncertainty model). There exists a set S of deterministic points in $\mathbb{R}^d$ (which may not be a subset of $\mathcal{P}$) such that $\omega(u, \mathcal{P}) = \omega(u, S)$ for all $u \in \mathbb{R}^d$.*

Proof. By the definition of the expected directional width of $\mathcal{P}$, we have that $\omega(\mathcal{P}, u) = \mathbb{E}_{P\sim\mathcal{P}}[\omega(P, u)] = \sum_{P\sim\mathcal{P}} \Pr[P]\Big(f(P, u) + f(P, -u)\Big)$. Consider the Minkowski sum $M = M(\mathcal{P}) := \sum_{P\sim\mathcal{P}} \Pr[P]\mathsf{ConvH}(P)$, where $\mathsf{ConvH}(P)$ is the convex hull of P (including the interior). It is well known that the Minkowski sum of a set of convex sets is also convex. Moreover, it also holds that for all $u \in \mathbb{R}^d$ (see e.g., [48]) $f(M, u) = \sum_{P\sim\mathcal{P}} \Pr[P]f(P, u)$. Hence, $\omega(\mathcal{P}, u) = \omega(M, u)$ for all $u \in \mathbb{R}^d$. ◀

By the result in [7], for any convex body in $\mathbb{R}^d$, there exists an ε-kernel of size $O(\varepsilon^{-(d-1)/2})$. Combining with Lemma 10, we can immediately obtain the following corollary.

▶ **Corollary 11.** *For any $\varepsilon > 0$, there exists an ε-*EXP-KERNEL *of size $O(\varepsilon^{-(d-1)/2})$.*

Recall that in Lemma 10, the Minkowski sum $M = \sum_{P\sim\mathcal{P}} \Pr[P]\mathsf{ConvH}(P)$. Since M is the Minkowski sum of exponential many convex polytopes, so M is also a convex polytope. At first sight, the complexity of M (i.e., number of vertices) could be exponential. However, as we will show shortly, the complexity of M is in fact polynomial.

We need some notations first. For each pair (r, w) of points in $\mathcal{P}$ consider the hyperplane $H_{r,w}$ that passes through the origin and is orthogonal to the line connecting r and w. We call these $\binom{n}{2}$ hyperplanes *the separating hyperplanes induced by $\mathcal{P}$* and use Γ to denote the set. Each such hyperplane divides $\mathbb{R}^d$ into 2 halfspaces. For all vectors $u \in \mathbb{R}^d$ in each halfspace, the order of $\langle r, u\rangle$ and $\langle w, u\rangle$ is the same (i.e., we have $r \succ_u w$ in one halfspace and $w \succ_u r$ in the other). Those hyperplanes in Γ pass through the origin and thus partition $\mathbb{R}^d$ into d-dimensional polyhedral cones. We denote this *arrangement* as $\mathbb{A}(\Gamma)$.

Consider an arbitrary cone $C \in \mathbb{A}(\Gamma)$. Let $\operatorname{int} C$ denote the interior of C. We can see that for all vectors $u \in \operatorname{int} C$, the canonical order of $\mathcal{P}$ with respect to u is the same (since all vector $u \in \operatorname{int} C$ lie in the same set of halfspaces). We use $|M|$ to denote the complexity of M, i.e., the number of vertices in $\mathsf{ConvH}(M)$.

▶ **Lemma 12.** *Assuming the existential model and $p_v \in (0, 1)$ for all $v \in \mathcal{P}$, the complexity of M is the same as the cardinality of $\mathbb{A}(\Gamma)$, i.e., $|M| = |\mathbb{A}(\Gamma)|$. Moreover, each cone $C \in \mathbb{A}(\Gamma)$ corresponds to exactly one vertex v of $\mathsf{ConvH}(M)$ in the following sense: the gradient $\nabla f(M, u) = v$ for all $u \in \operatorname{int} C$ (note that here v should be understood as a vector).*

Proof. (sketch) We have shown that M is a convex polytope. We first note that the support function uniquely defines a convex body (see e.g., [48]). We need the following well known

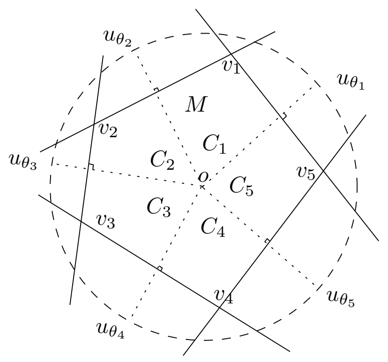


Figure 1 The figure depicts a pentagon M in $\mathbb{R}^2$ to illustrate some intuitive facts in convex geometry. (1) The plane can be divided into 5 cones $C_1, \ldots, C_5$, by 5 angles $\theta_1, \ldots, \theta_5$. u_{θ_i} is the unit vector corresponding to angle θ_i. Each cone C_i corresponds to a vertex v_i and for any direction $u \in C_i$, $f(M,u) = \langle u, v_i \rangle$ and the vector $\nabla f(M,u)$ is v_i. (2) Each direction θ_i is perpendicular to an edge of M. $M = \cap_{i=1}^5 H_i$ where H_i is the supporting halfplane with normal vector u_{θ_i}.

fact in convex geometry (see e.g., [25]): For any convex polytope M, $\mathbb{R}^d$ can be divided into exactly $|M|$ polyhedral cones (of dimension d, ignoring the boundaries), such that each such cone C_v corresponds to a vertex v of M, and for each vector $u \in C_v$, it holds $f(M,u) = \langle u, v \rangle$ (i.e., the maximum of $f(M,u) = \max_{v' \in M} \langle u, v' \rangle$ is achieved by v for all $u \in C_v$)[1]. See Figure 1 for an example in $\mathbb{R}^2$. Hence, for each $u \in \operatorname{int} C_v$ the gradient of of the support function (as a function of u) is exactly v:

$$\nabla f(M,u) = \left\{ \frac{\partial f(M,u)}{\partial u_j} \right\}_{j \in [d]} = \left\{ \frac{\partial \langle u, v \rangle}{\partial u_j} \right\}_{j \in [d]} = \left\{ \frac{\partial \sum_{j \in [d]} v_j u_j}{\partial u_j} \right\}_{j \in [d]} = v, \tag{2}$$

where u_j is the jth coordinate of u. With a bit abuse of notation, we denote the set of cones defined above by $\mathbb{A}(M)$.

Now, consider a cone $C \in \mathbb{A}(\Gamma)$. We show that for all $u \in \operatorname{int} C$, $\nabla f(M,u)$ is a distinct constant vector independent of u. In fact, we know that $f(M,u) = f(\mathcal{P},u) = \sum_{v \in \mathcal{P}} \Pr^R(v,u) \langle v, u \rangle$, where $\Pr^R(v,u) = \prod_{v' \succ_u v}(1 - p_{v'}) p_v$. For all $u \in \operatorname{int} C$, the $\Pr^R(v,u)$ value is the same since the value only depends on the canonical order with respect to u, which is the same for all $u \in C$. Hence, we can get that for all $u \in \operatorname{int} C$, $\nabla f(M,u) = \sum_{v \in \mathcal{P}} \Pr^R(v,u) v$, which is a constant independent of u. We can also show that the gradient $\nabla f(M,u)$ must be different for two adjacent cones C_1, C_2 (separated by some hyperplane in Γ) in $\mathbb{A}(\Gamma)$. So $\nabla f(M,u)$ is piecewise constant, with a distinct constant in each cone in $\mathbb{A}(M)$. The same also holds for $\mathbb{A}(\Gamma)$. This is only possible if $\mathbb{A}(\Gamma)$ (thinking as a partition of $\mathbb{R}^d$) partitions $\mathbb{R}^d$ exactly the same way as $\mathbb{A}(M)$ does. Hence, we have $\mathbb{A}(\Gamma) = \mathbb{A}(M)$ and the lemma follows immediately. ◀

Since $O(n^2)$ hyperplanes passing through the origin can divide $\mathbb{R}^d$ into at most $O(\binom{n^2}{d-1})$ d-dimensional polyhedral cones (see e.g., [9]), we immediately obtain the following corollary.

▶ **Corollary 13.** *It holds that* $|M| \le O(\binom{n^2}{d-1}) = O(n^{2d-2})$.

[1] The support function for a polytope is just the upper envelope of a finite set of linear functions, thus a piecewise linear function, and the domain of each piece is a polyhedral cone.

▶ **Theorem 14.** *In $\mathbb{R}^d$ for constant d, the polytope M which defines $f(\mathcal{P}, u)$ for any direction u can be described with $O(n^{2d-2})$ vertices in $\mathbb{R}^d$, and can be computed in $O(n^{2d-1}\log n)$ time. In $\mathbb{R}^2$, the runtime can be improved to $O(n^2 \log n)$.*

In fact, we can reduce the construction time to nearly linear time, which leads to Theorem 4. The details can be found in the full version.

2.1 ε-exp-kernel Under the Subset Constraint

First, we show that under the subset constraint (i.e., the ε-EXP-KERNEL is required to be a subset of the original point set, with the same probability distribution for each chosen point), there exists no ε-EXP-KERNEL with small size in general.

▶ **Lemma 15.** *For some constant $\varepsilon > 0$, there exist a set $\mathcal{P}$ of stochastic points such that no $o(n)$ size ε-EXP-KERNEL exists for $\mathcal{P}$ under the subset constraint (for both locational model and existential model).*

In light of the above negative result, we make the following β-*assumption*: we assume each possible location realizes a point with probability at least β, for a constant $\beta > 0$.

▶ **Theorem 16.** *Under the β-assumption, in the existential uncertainty model, there is an ε-EXP-KERNEL in $\mathbb{R}^d$ of size $O(\beta^{-(d-1)}\varepsilon^{-(d-1)/2}\log(1/\varepsilon))$ that satisfies the subset constraint.*

3 ε-Kernels for Probability Distributions of Width

Recall $\mathcal{S}$ is an (ε, τ)-QUANT-KERNEL if for all $x \geq 0$, $\Pr_{P\sim\mathcal{P}}\Big[\omega(P, u) \leq (1-\varepsilon)x\Big] - \tau \leq \Pr_{P\sim\mathcal{S}}\Big[\omega(S, u) \leq x\Big] \leq \Pr_{P\sim\mathcal{P}}\Big[\omega(P, u) \leq (1+\varepsilon)x\Big] + \tau$. For ease of notation, we sometimes write $\Pr\big[\omega(\mathcal{P}, u) \leq t\big]$ to denote $\Pr_{P\sim\mathcal{P}}\big[\omega(P, u) \leq t\big]$, and abbreviate the above as $\Pr\Big[\omega(S, u) \leq x\Big] \in \Pr\Big[\omega(\mathcal{P}, u) \leq (1 \pm \varepsilon)x\Big] \pm \tau$. We first provide a simple linear time algorithm for constructing an (ε, τ)-QUANT-KERNEL for both existential and locational models, in Section 3.1. The points in the constructed kernel are not independent. Then, for existential models, we provide a nearly linear time (ε, τ)-QUANT-KERNEL construction where all stochastic points in the kernel are independent in Section 3.2.

3.1 A Simple (ε, τ)-quant-kernel Construction

In this section, we show a linear time algorithm for constructing an (ε, τ)-QUANT-KERNEL for any stochastic model if we can sample a realization from the model in linear time (which is true for both locational and existential uncertainty models).

Algorithm: Let $N = O\big(\tau^{-2}\varepsilon^{-(d-1)}\log(1/\varepsilon)\big)$. We sample N independent realizations from the stochastic model. Let $\mathcal{H}_i$ be the convex hull of the present points in the ith realization. For $\mathcal{H}_i$, we use the algorithm in [7] to find a deterministic ε-kernel $\mathcal{E}_i$ of size $O(\varepsilon^{-(d-1)/2})$. Our (ε, τ)-QUANT-KERNEL $\mathcal{S}$ is the following simple stochastic model: with probability $1/N$, all points in $\mathcal{E}_i$ are present. Hence, $\mathcal{S}$ consists of $O\big(\tau^{-2}\varepsilon^{-3(d-1)/2}\log(1/\varepsilon)\big)$ points (two such points either co-exist or are mutually exclusive). Hence, for any direction u, $\Pr[\omega(\mathcal{S}, u) \leq t] = \frac{1}{N}\sum_{i=1}^{N}\mathbb{I}(\omega(\mathcal{E}_i, u) \leq t)$, where $\mathbb{I}(\cdot)$ is the indicator function.

For a realization $P \sim \mathcal{P}$, we use $\mathcal{E}(P)$ to denote the deterministic ε-kernel for P. So, $\mathcal{E}(P)$ is a random set of points, and we can think of $\mathcal{E}_1, \ldots, \mathcal{E}_N$ as samples from the random set. We first show that $\mathcal{S}$ is an (ε, τ)-QUANT-KERNEL for $\mathcal{E}(P)$.

▶ **Lemma 17.** *Let $N = O(\tau^{-2}\varepsilon^{-(d-1)}\log(1/\varepsilon))$. For any $t \geq 0$ and any direction u,*

$$\Pr[\omega(\mathcal{S}, u) \leq t] \in \Pr_{P\sim\mathcal{P}}[\omega(\mathcal{E}(P), u) \leq t] \pm \tau.$$

Proof (Sketch). Let $L = O(\varepsilon^{-(d-1)/2})$. We first build a mapping g that maps each realization $\mathcal{E}(P)$ to a point in $\mathbb{R}^{dL}$, as follows: Consider a realization P of $\mathcal{P}$. Suppose $\mathcal{E}(P) = \{(x_1^1, \dots, x_d^1), \dots, (x_1^L, \dots, x_d^L)\}$ (if $|\mathcal{E}(P)| < L$, we pad it with $(0, \dots, 0)$). We let $g(\mathcal{E}(P)) = (x_1^1, \dots, x_d^1, \dots, x_1^L, \dots, x_d^L) \in \mathbb{R}^{dL}$. For any $t \geq 0$ and any direction $u \in \mathbb{R}^d$, note that $\omega(\mathcal{E}(P), u) \geq t$ holds if and only if there exists some $1 \leq i, j \leq |\mathcal{E}(P)|, i \neq j$ satisfies that $\sum_{k=1}^{d}(x_k^i - x_k^j)u_k \geq t$, which is equivalent to saying that point $g(\mathcal{E}(P))$ is in the union of the those $O(|\mathcal{E}(P)|^2)$ halfspaces. We can show that the VC dimension of the union of $O(|\mathcal{E}(P)|^2)$ such halfspaces is bounded by $O(\varepsilon^{-(d-1)}\log(1/\varepsilon))$. Then, the lemma follows from the VC uniform convergence theorem [51, 10]. ◀

From the above lemma, it is not difficult to obtain Theorem 5.

3.2 Improved (ε, τ)-quant-kernel for Existential Models

In this section, we show an (ε, τ)-QUANT-KERNEL $\mathcal{S}$ can be constructed in nearly linear time for the existential model, and all points in $\mathcal{S}$ are independent of each other. The size bound $\widetilde{O}(\tau^{-2}\varepsilon^{-(d-1)})$ (see Theorem 6) is better than that in Theorem 5 for the general case, and the independence property may be useful in certain applications. Moreover, some of the insights developed in this section may be of independent interest (e.g., the connection to Tukey depth). Due to the independence requirement, the construction is somewhat more involved. For ease of the description, we assume the Euclidean plane first. All results can be easily extended to $\mathbb{R}^d$. We also assume that all probability values are strictly between 0 and 1 and $0 < \varepsilon, \tau \leq 1/2$ is a fixed constant.

Let $\lambda(\mathcal{P}) = \sum_{v\in\mathcal{P}}(-\ln(1-p_v))$. In the following, we present two algorithms. The first algorithm works for any $\lambda(\mathcal{P})$ and produces an (ε, τ)-QUANT-KERNEL $\mathcal{S}$ whose size depends on $\lambda(\mathcal{P})$. In Section 3.2.2, we present the second algorithm that only works for $\lambda(\mathcal{P}) \geq 3\ln(2/\tau)$ but produces an (ε, τ)-QUANT-KERNEL $\mathcal{S}$ with a constant size (the constant only depends on ε, τ and δ). Thus, we can get a constant size (ε, τ)-QUANT-KERNEL by running the first algorithm when $\lambda(\mathcal{P}) \leq 3\ln(2/\tau)$ and running the second algorithm otherwise.

3.2.1 Algorithm 1: For Any $\lambda(\mathcal{P})$

In this section, we present the first algorithm which works for any $\lambda(\mathcal{P})$. We can think of each point v associated with a Bernoulli random variable X_v that takes value 1 with probability p_v and 0 otherwise. Now, we replace the Bernoulli random variable X_v by a Poisson distributed random variable $\widetilde{X}_v$ with parameter $\lambda_v = -\ln(1-p_v)$ (denoted by Pois(λ_v)), i.e., $\Pr[\widetilde{X}_v = k] = \frac{1}{k!}\lambda_v^k e^{-\lambda_v}$, for $k = 0, 1, 2, \dots$. Here, $\widetilde{X}_v = k$ means that there are k realized points located at the position of v. We call the new instance *the Poissonized instance corresponding to* $\mathcal{P}$. We can check that $\Pr[\widetilde{X}_v = 0] = e^{-\lambda_v} = 1 - p_v = \Pr[X_v = 0]$. Also note that co-locating points does not affect any directional width, so the Poissonized instance is essentially equivalent to the original instance for our problem.

The construction of the (ε, τ)-QUANT-KERNEL $\mathcal{S}$ is as follows: Let $\mathfrak{A}$ be the probability measure over all points in $\mathcal{P}$ defined by $\mathfrak{A}(\{v\}) = \lambda_v/\lambda$ for every $v \in \mathcal{P}$, where $\lambda := \lambda(\mathcal{P}) = \sum_{v\in\mathcal{P}}\lambda_v$. Let τ_1 be a small positive constant to be fixed later. We take $N = \widetilde{O}(\tau_1^{-2})$ independent samples from $\mathfrak{A}$ (we allow more than one point to be co-located at the same position), and let $\mathfrak{B}$ be the empirical measure, i.e., each sample point having probability

$1/N$. The coreset $\mathcal{S}$ consists of the N sample points in $\mathfrak{B}$, each with the same existential probability $1 - \exp(-\lambda/N)$.

▶ **Theorem 18.** *Let $\tau_1 = O(\frac{\tau}{\max\{\lambda,\lambda^2\}})$ and $N = O(\frac{1}{\tau_1^2}\log\frac{1}{\delta}) = O(\frac{\max\{\lambda^2,\lambda^4\}}{\tau^2}\log\frac{1}{\delta})$. With probability at least $1-\delta$, for any $t \geq 0$ and any direction u, we have that $\Pr\Big[\omega(\mathcal{S},u) \leq t\Big] \in \Pr\Big[\omega(\mathcal{P},u) \leq t\Big] \pm \tau$.*

3.2.2 Algorithm 2: For $\lambda(\mathcal{P}) > 3\ln(2/\tau)$

In the second algorithm, we assume that $\lambda(\mathcal{P}) = \sum_{v\in\mathcal{P}} \lambda_v > 3\ln(2/\tau)$. When $\lambda(\mathcal{P})$ is large, we cannot directly use the sampling technique in the previous section since it requires a large number of samples. However, the condition $\lambda(\mathcal{P}) \geq 3\ln(2/\tau)$ implies there is a nonempty convex region $\mathcal{K}$ inside the convex hull of $\mathcal{P}$ with high probability. Moreover, we can show the sum of λ_v values in $\overline{\mathcal{K}} = \mathbb{R}^2 \setminus \mathcal{K}$ is small. Hence, we can use the sampling technique just for $\overline{\mathcal{K}}$ and use the deterministic ε-kernel construction for $\mathcal{K}$.

Again consider the Poissonized instance of $\mathcal{P}$. Imagine the following process. Fix a direction u. We move a sweep line ℓ_u orthogonal to u, along the direction u, to sweep through the points in $\mathcal{P}$. We use H_u to denote the halfplane defined by ℓ_u (with normal vector u) and $\overline{H}_u$ denote its complement. So $\mathcal{P}(\overline{H}_u) = \mathcal{P} \cap \overline{H}_u$ is the set of points that have been swept so far. We stop the movement of ℓ_u at the first point such that $\sum_{v\in\overline{H}_u} \lambda_v \geq \ln(2/\tau)$ (ties should be broken in an arbitrary but consistent manner). One important property about $\overline{H}_u$ is that $\Pr[\overline{H}_u \models 0] \leq \tau/2$. We repeat the above process for all directions u and let $\mathcal{H} = \cap_u H_u$. Since $\lambda(\mathcal{P}) > 3\ln(2/\tau)$, by Helly's theorem, $\mathcal{H}$ is nonempty. A careful examination of the above process reveals that $\mathcal{H}$ is in fact a convex polytope and each edge of the polytope is defined by two points in $\mathcal{P}$.

The construction of the (ε,τ)-QUANT-KERNEL $\mathcal{S}$ is as follows. First, we use the algorithm in [7] to find a deterministic ε-kernel $\mathcal{E}_{\mathcal{H}}$ of size $O(\varepsilon^{-1/2})$ for $\mathcal{H}$. One useful property of the algorithm in [7] is that $\mathcal{E}_{\mathcal{H}}$ is a subset of the vertices of $\mathcal{H}$. Hence the convex polytope $\mathsf{ConvH}(\mathcal{E}_{\mathcal{H}})$ is contained in $\mathcal{H}$. Since $\mathcal{E}_{\mathcal{H}}$ is an ε-kernel, $(1+\varepsilon)\mathsf{ConvH}(\mathcal{E}_{\mathcal{H}}))$ (properly shifted) contains $\mathcal{H}$. Let $\mathcal{K} = (1+\varepsilon)\mathsf{ConvH}(\mathcal{E}_{\mathcal{H}})$ and $\overline{\mathcal{K}} = \mathcal{P} \setminus \mathcal{K}$. See Figure 2.

Now, we apply the random sampling construction over $\overline{\mathcal{K}}$. More specifically, let $\lambda := \lambda(\overline{\mathcal{K}}) = \sum_{v\in\overline{\mathcal{K}}\cap\mathcal{P}} \lambda_v$. Let $\mathfrak{A}$ be the probability measure over $\mathcal{P}\cap\overline{\mathcal{K}}$ defined by $\mathfrak{A}(\{v\}) = \lambda_v/\lambda$ for every $v \in \mathcal{P}\cap\overline{\mathcal{K}}$. Let $\tau_1 = O(\tau/\lambda)$. We take $N = O(\tau_1^{-2}\log(1/\delta))$ independent samples from $\mathfrak{A}$ and let $\mathfrak{B}$ be the empirical distribution with each sample point having probability $1/N$. The (ε,τ)-QUANT-KERNEL $\mathcal{S}$ consists of the N points in $\mathfrak{B}$, each with the same existential probability $1-\exp(-\lambda/N)$, as well as all vertices of $\mathcal{K}$, each with probability 1.

▶ **Theorem 19.** *Let $\lambda = \lambda(\overline{\mathcal{K}})$ and $\tau_1 = O(\tau/\lambda)$, and $N = O\Big(\frac{1}{\tau_1^2}\log\frac{1}{\delta}\Big) = O\Big(\frac{\ln^2 1/\tau}{\varepsilon\tau^2}\log\frac{1}{\delta}\Big)$. With probability at least $1-\delta$, for any $t \geq 0$ and any direction u, we have that*

$$\Pr\Big[\omega(\mathcal{S},u) \leq t\Big] \in \Pr\Big[\omega(\mathcal{P},u) \leq (1\pm\varepsilon)t\Big] \pm \tau. \tag{3}$$

In summary, we obtain Theorem 6 combining Theorem 18 and Theorem 19.

4 (ε, r)-fpow-kernel Under the β-Assumption

We now show an (ε,r)-FPOW-KERNEL exists in the existential uncertainty model under the β-assumption. Recall that the function $T_r(P,u) = \max_{v\in P}\langle u,v\rangle^{1/r} - \min_{v\in P}\langle u,v\rangle^{1/r}$. For

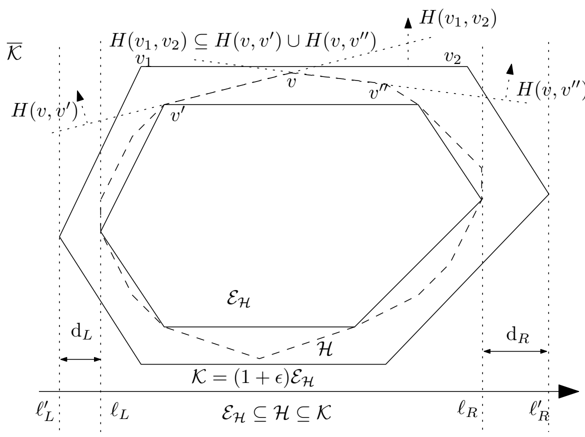

$\overline{\mathcal{K}}$
$H(v_1,v_2) \subseteq H(v,v') \cup H(v,v'')$
$H(v_1,v_2)$
v_1
v_2
v
v''
v'
$H(v,v')$
$H(v,v'')$
$\mathcal{E}_{\mathcal{H}}$
d_L
d_R
$\mathcal{H}$
$\mathcal{K} = (1+\epsilon)\mathcal{E}_{\mathcal{H}}$
ℓ'_L
ℓ_L
$\mathcal{E}_{\mathcal{H}} \subseteq \mathcal{H} \subseteq \mathcal{K}$
ℓ_R
ℓ'_R

Figure 2 Construction of (ε,τ)-QUANT-KERNEL $\mathcal{S}$. The dashed polygon is $\mathcal{H}$. The inner solid polygon is $\mathsf{ConvH}(\mathcal{E}_{\mathcal{H}})$ and the outer one is $K = (1+\varepsilon)\mathsf{ConvH}(\mathcal{E}_{\mathcal{H}})$. $\overline{\mathcal{K}}$ is the set of points outside $\mathcal{K}$.

ease of notation, we write $\mathbb{E}[T_r(\mathcal{P},u)]$ to denote $\mathbb{E}_{P\sim\mathcal{P}}[T_r(P,u)]$. Our goal is to find a set $\mathcal{S}$ of stochastic points so for all directions $u \in \mathcal{P}^\star$, that $\mathbb{E}[T_r(\mathcal{S},u)] \in (1\pm\varepsilon)\mathbb{E}[T_r(\mathcal{P},u)]$.

Our construction of $\mathcal{S}$ is almost the same as that in Section 3.1. Suppose we sample N (fixed later) independent realizations and take the ε_0-kernel for each of them. Suppose they are $\{\mathcal{E}_1,\ldots,\mathcal{E}_N\}$ and we associate each a probability $1/N$. We denote the resulting (ε,r)-FPOW-KERNEL by $\mathcal{S}$. Hence, for any direction $u \in \mathcal{P}^\star$, $\mathbb{E}[T_r(\mathcal{S},u)] = \frac{1}{N}\sum_{i=1}^{N} T_r(\mathcal{E}_i,u)$ and we use this value as the estimation of $\mathbb{E}[T_r(\mathcal{P},u)]$. Our result is summarized by Theorem 7.

5 Applications

In this section, we show that our coreset results for the directional width problem readily imply several coreset results for other stochastic problems, just as in the deterministic setting. We introduce these stochastic problems and briefly summarize our results below.

5.1 Approximating the Extent of Uncertain Functions

We first consider the problem of approximating the extent of a set $\mathcal{H}$ of uncertain functions. As before, we consider both the existential model and the locational model of uncertain functions.

1. In the existential model, each uncertain function h is a function in $\mathbb{R}^d$ associated with a existential probability p_f, which indicates the probability that h presents in a random realization.
2. In the locational model, each uncertain function h is associated with a finite set $\{h_1,h_2,\ldots\}$ of deterministic functions in $\mathbb{R}^d$. Each h_i is associated with a probability value $p(h_i)$, such that $\sum_i p(h_i) = 1$. In a random realization, h is independently realized to some h_i, with probability $p(h_i)$.

We use $\mathcal{H}$ to denote the random instance, that is a random set of functions. We use $h \in \mathcal{H}$ to denote the event that the deterministic function h is present in the instance. For each point $x \in \mathbb{R}^d$, we let the random variable $\mathfrak{E}_{\mathcal{H}}(x) = \max_{h\in\mathcal{H}} h(x) - \min_{h\in\mathcal{H}} h(x)$ be the extent of $\mathcal{H}$ at point x. Suppose $\mathcal{S}$ is another set of uncertain functions. We say $\mathcal{S}$ is the ε-EXP-KERNEL for $\mathcal{H}$ if $(1-\varepsilon)\mathfrak{E}_{\mathcal{H}}(x) \le \mathfrak{E}_{\mathcal{S}}(x) \le \mathfrak{E}_{\mathcal{H}}(x)$ for any $x \in \mathbb{R}^d$. We say $\mathcal{S}$ is the (ε,τ)-QUANT-KERNEL for $\mathcal{H}$ if $\Pr_{S\sim\mathcal{S}}\Big[\mathfrak{E}_S(x) \le t\Big] \in \Pr_{H\sim\mathcal{H}}\Big[\mathfrak{E}_H(x) \le (1\pm\varepsilon)t\Big] \pm \phi$. for any $t \ge 0$ and any $x \in \mathbb{R}^d$.

Let us first focus on linear functions in $\mathbb{R}^d$. Using the *duality transformation* that maps linear function $y = a_1x_1 + \ldots + a_dx_d + a_{d+1}$ to the point $(a_1, \ldots, a_{d+1}) \in \mathbb{R}^{d+1}$, we can reduce the extent problem to the directional width problem in $\mathbb{R}^{d+1}$. Let $\mathcal{H}$ be a set of uncertain linear functions (under either existential or locational model) in $\mathbb{R}^d$ for constant d. From Theorem 14 and Corollary 11, we can construct a set S of $O(n^{2d})$ deterministic linear functions in $\mathbb{R}^d$, such that $\mathfrak{E}_S(x) = \mathbb{E}[\mathfrak{E}_{\mathcal{H}}(x)]$ for any $x \in \mathbb{R}^d$. Moreover, for any $\varepsilon > 0$, there exists an ε-EXP-KERNEL of size $O(\varepsilon^{-d/2})$ and an (ε, τ)-QUANT-KERNEL of size $\widetilde{O}(\tau^{-2}\varepsilon^{-d})$. Using the standard linearization technique [7], we can obtain the following generalization for uncertain polynomials.

▶ **Theorem 20.** *Let $\mathcal{H}$ be a family of uncertain polynomials in $\mathbb{R}^d$ (under either existential or locational model) that admits linearization of dimension k. We can construct a set M of $O(n^{2k})$ deterministic polynomials, such that $\mathfrak{E}_M(x) = \mathbb{E}[\mathfrak{E}_{\mathcal{H}}(x)]$ for any $x \in \mathbb{R}^d$. Moreover, for any $\varepsilon > 0$, there exists an ε-EXP-KERNEL of size $O(\varepsilon^{-k/2})$ and an (ε, τ)-QUANT-KERNEL of size $\min\{\widetilde{O}(\tau^{-2}\max\{\lambda^2, \lambda^4\}), \widetilde{O}(\varepsilon^{-k}\tau^{-2})\}$. Here $\lambda = \sum_{h\in\mathcal{H}}(-\ln(1-p_h))$.*

Now, we consider functions of the form $u(x) = p(x)^{1/r}$ where $p(x)$ is a polynomial and r is a positive integer. We call such a function a *fractional polynomial*. We still use $\mathcal{H}$ to denote the random set of fractional polynomials. Let $\mathcal{H}^\star \subseteq \mathbb{R}^d$ be the set of points such that for any points $x \in \mathcal{H}^\star$ and any function $u \in \mathcal{H}$, we have $u(x) \geq 0$. For each point $x \in \mathcal{H}^\star$, we let the random variable $\mathfrak{E}_{r,\mathcal{H}}(x) = \max_{h\in\mathcal{H}} h(x)^{1/r} - \min_{h\in\mathcal{H}} h(x)^{1/r}$. We say another random set $\mathcal{S}$ of functions is the (ε, r)-FPOW-KERNEL for $\mathcal{H}$ if $(1-\varepsilon)\mathfrak{E}_{r,\mathcal{H}}(x) \leq \mathfrak{E}_{r,\mathcal{S}}(x) \leq \mathfrak{E}_{r,\mathcal{H}}(x)$ for any $x \in \mathcal{H}^\star$. By the duality transformation and Theorem 7, we can obtain the following result.

▶ **Theorem 21.** *Let $\mathcal{H}$ be a family of uncertain fractional polynomials in $\mathbb{R}^d$ in the existential uncertainty model under the β-assumption. Further assume that each polynomial admits a linearization of dimension k. For any $\varepsilon > 0$, there exists an (ε, r)-FPOW-KERNEL of size $\widetilde{O}(\varepsilon^{-(rk-r+2)})$. Furthermore, the (ε, r)-FPOW-KERNEL consists of $N = O\big(\varepsilon^{-(rk-r+4)/2}\big)$ sets, each occurring with probability $1/N$ and containing $O\big(\varepsilon^{-r(k-1)/2}\big)$ deterministic fractional polynomials.*

5.2 Stochastic Moving Points

We can extend our stochastic models to moving points. In the existential model, each point v is present with probability p_v and follows a trajectory $v(t)$ in $\mathbb{R}^d$ when present ($v(t)$ is the position of v at time t). In the locational model, each point v is associated with a distribution of trajectories (the support size is finite) and the actual trajectory of v is a random sample for the distribution. Such uncertain trajectory models have been used in several applications in spatial databases [54]. For ease of exposition, we assume the existential model in the following. Suppose each trajectory is a polynomial of t with degree at most r. For each point v, any direction u and time t, define the polynomial $f_v(u,t) = \langle v(t), u\rangle$ and let $\mathcal{H}$ include f_v with probability p_v. For a set $\mathcal{P}$ of points, the directional width at time t is $\mathfrak{E}_{\mathcal{H}}(u,t) = \max_{v\in\mathcal{P}} f_v(u,t) - \min_{v\in\mathcal{P}} f_v(u,t)$. Each polynomial f_v admits a linearization of dimension $k = (r+1)d - 1$. Using Theorem 20, we can see that there is a set M of $O(n^{2k})$ deterministic moving points, such that the directional width of M in any direction u is the same as the expected directional width of $\mathcal{P}$ in direction u. Moreover, for any $\varepsilon > 0$, there exists an ε-EXP-KERNEL (which consists of only deterministic moving points) of size $O(\varepsilon^{-(k-1)/2})$ and an (ε, τ)-QUANT-KERNEL (which consists of both deterministic and stochastic moving points) of size $\widetilde{O}(\varepsilon^{-k}\tau^{-2})$.

5.3 Shape Fitting Problems

Theorem 20 can be also applied to some stochastic variants of certain shape fitting problems. We first consider the following variant of the minimum enclosing ball problem over stochastic points. We are given a set $\mathcal{P}$ of stochastic points (under either existential or locational model), find the center point c such that $\mathbb{E}[\max_{v\in\mathcal{P}} \|v-c\|^2]$ is minimized. It is not hard to see that the problem is equivalent to minimizing the expected area of the enclosing ball in $\mathbb{R}^2$. For ease of exposition, we assume the existential model where v is present with probability p_v. For each point $v \in P$, define the polynomial $h_v(x) = \|x\|^2 - 2\langle x, v\rangle + \|v\|^2$, which admits a linearization of dimension $d+1$ [7]. Let $\mathcal{H}$ be the family of uncertain polynomials $\{h_v\}_{v\in\mathcal{P}}$ (h_v exists with probability p_v). We can see that for any $x \in \mathbb{R}^d$, $\max_{v\in\mathcal{P}} \|x-v\|^2 = \max_{h_v\in\mathcal{H}} h_v(x)$. Using Theorem 20, we can see that there is a set M of $O(n^{2d+2})$ deterministic polynomials such that $\max_{h\in M} h(x) = \mathbb{E}[\max_{v\in\mathcal{P}} \|x-v\|^2]$ for any $x \in \mathbb{R}^d$ and a set S of $O(\varepsilon^{-(d+1)/2})$ deterministic polynomials such that $(1-\varepsilon)\mathbb{E}[\max_{v\in\mathcal{P}} \|x-v\|^2] \leq \max_{h\in S} h(x) \leq \mathbb{E}[\max_{v\in\mathcal{P}} \|x-v\|^2]$ for any $x \in \mathbb{R}^d$. We can store the set S instead of the original point set in order to answer the following queries: given a point v, return the expected length of the furthest point from v. The problem of finding the optimal center c can be also carried out over S, which can be done in $O(\varepsilon^{-O(d^2)})$ time: We can decompose the arrangement of n semialgebraic surfaces in $\mathbb{R}^d$ into $O(n^{O(d+k)})$ cells of constant description complexity, where k is the linearization dimension (see e.g., [9]). By enumerating all those cells in the arrangement of S, we know which polynomials lie in the upper envelopes, and we can compute the minimum value in each such cell in constant time when d is constant.

The above argument can also be applied to the following variant of the spherical shell for stochastic points. We are given a set $\mathcal{P}$ of stochastic points (under either existential or locational model). Our objective is to find the center point c such that $\mathbb{E}[\mathsf{obj}(c)] = \mathbb{E}[\max_{v\in P} \|v-c\|^2 - \min_{v\in P} \|v-c\|^2]$ is minimized. The problem is equivalent to minimizing the expected area of the enclosing annulus in $\mathbb{R}^2$. The objective can be represented as a polynomial of linearization dimension $k = d+1$. Proceeding as for the enclosing balls, we can show there is a set S of $O(\varepsilon^{-(k-1)/2})$ deterministic polynomials such that $(1-\varepsilon)\mathbb{E}[\mathsf{obj}(c)] \leq \mathfrak{E}_S(x) \leq \mathbb{E}[\mathsf{obj}(c)]$ for any $x \in \mathbb{R}^d$. We summarize our results by Theorem 8. We would like to make a few remarks here.

1. We take the minimum enclosing ball for example. If we examine the construction of set S, each polynomial $h \in S$ may *not* be of the form $h(x) = \|x\|^2 - 2\langle x, v\rangle + \|v\|^2$, therefore does not translate back to a minimum enclosing ball problem over deterministic points.
2. Another natural objective function for the minimum enclosing ball and the spherical shell problem would be the expected radius $\mathbb{E}[\max_{v\in P} \mathrm{d}(v,c)]$ and the expected shell width $\mathbb{E}[\max_{v\in P} \mathrm{d}(v,c) - \min_{v\in P} \mathrm{d}(v,c)]$. However, due to the fractional powers (square roots) in the objectives, simply using an ε-EXP-KERNEL does not work. This is unlike the deterministic setting. [2] We leave the problem of finding small coresets for the spherical shell problem as an interesting open problem. However, under the β-assumption, we can use (ε, r)-FPOW-KERNELs to handle such fractional powers, as in the next subsection.

5.4 Shape Fitting Problems (Under the β-assumption)

In this subsection, we consider several shape fitting problems in the existential model *under the β-assumption.* We show how to use Theorem 21 to obtain linear time approximation schemes for those problems.

[2] In particular, there is no stochastic analogue of Lemma 4.6 in [7].

1. (Minimum spherical shell) We first consider the minimum spherical shell problem. Given a set $\mathcal{P}$ of stochastic points (under the β-assumption), our goal is to find the center point c such that $\mathbb{E}[\max_{v\in P}\|v-c\|-\min_{v\in P}\|v-c\|]$ is minimized. For each point $v\in P$, let $h_v(x)=\|x\|^2-2\langle x,v\rangle+\|v\|^2$, which admits a linearization of dimension $d+1$. It is not hard to see that $\mathbb{E}[\max_{v\in P}\|v-c\|]=\mathbb{E}[\max_{v\in P}\sqrt{h_v(c)}]$ and $\mathbb{E}[\min_{v\in P}\|v-c\|]=\mathbb{E}[\min_{v\in P}\sqrt{h_v(c)}]$. Using Theorem 21, we can see that there are $N=\widetilde{O}\big(\varepsilon^{-(d+3)}\big)$ sets S_i, each containing $O\big(\varepsilon^{-(d+1)}\big)$ fractional polynomial $\sqrt{h_v}$s such that for all $x\in\mathbb{R}^d$,

$$\frac{1}{N}\sum_{i\in[N]}(\max_{S_i}\sqrt{h_v(x)}-\min_{S_i}\sqrt{h_v(x)})\in(1\pm\varepsilon)(\mathbb{E}[\max_{v\in P}\|v-x\|]-\mathbb{E}[\min_{v\in P}\|v-x\|]). \tag{4}$$

Note that our (ε,r)-FPOW-KERNEL satisfies the subset constraint. Hence, each function $\sqrt{h_v}$ corresponds to an original point in $\mathcal{P}$. So, we can store N point sets $P_i\subseteq\mathcal{P}$, with $|P_i|=O\big(\varepsilon^{-d}\big)$ as the coreset for the original point set. By (4), an optimal solution for the coreset is an $(1+\varepsilon)$-approximation for the original problem.
Now, we briefly sketch how to compute the optimal solution for the coreset. Consider all points in $\cup_i P_i$. Consider the arrangement of $O\big(\varepsilon^{-O(d)}\big)$ hyperplanes, each bisecting a pair of points in $\cup_i P_i$. For each cell C of the arrangement, for any point $v\in C$, the ordering of all points in $\cup_i P_i$ is fixed. We then enumerate all those cells in the arrangement and try to find the optimal center in each cell. Fix a cell C. For any point set P_i, we know which point is the furthest one and which point is the closest one from points in C_0. Say they are $v_i=\arg\max_{v\in P_i}\|v-x\|$ and $v_i'=\arg\min_{v\in P_i}\|v-x\|$. Hence, our problem can be formulated as the following optimization problem:

$$\min_x\frac{1}{N}\sum_i(d_i-d_i'),\quad \text{s.t.}\quad d_i^2=\|v_i-x\|^2, d_i'^2=\|v_i'-x\|^2, d_i,d_i'\ge 0,\forall i\in[N]; x\in C_0.$$

The polynomial system has a constant number of variables and constraints, hence can be solved in constant time. More specifically, we can introduce a new variable t and let $t=\frac{1}{N}\sum_i(d_i-d_i')$. All polynomial constraints define a semi-algebraic set. By using constructive version of Tarski-Seidenberg theorem, we can project out all variables except t and the resulting set is still a semi-algebraic set (which would be a finite collection of points and intervals in $\mathbb{R}^1$) (See e.g.,[12]).
2. (Minimum enclosing cylinder, Minimum cylindrical shell) Let $\mathcal{P}$ be a set of stochastic points in the existential uncertainty model under the β-assumption. Let $\mathrm{d}(\ell,v)$ denote the distance between a point $v\in\mathbb{R}^d$ and a line $\ell\subset\mathbb{R}^d$. The goal for the minimum enclosing cylinder problem is to find a line ℓ such that $\mathbb{E}[\max_{v\in\mathcal{P}}\mathrm{d}(\ell,v)]$ is minimized, while that for the minimum cylindrical shell problem is to minimize $\mathbb{E}[\max_{v\in\mathcal{P}}\mathrm{d}(\ell,v)-\min_{v\in\mathcal{P}}\mathrm{d}(\ell,v)]$. The algorithms for both problems are almost the same and we only sketch the one for the minimum enclosing cylinder problem.
We follow the approach in [7]. We represent a line $\ell\in\mathbb{R}^d$ by a $(2d-1)$-tuple $(x_1,\ldots,x_{2d-1})\in\mathbb{R}^{2d-1}$: $\ell=\{p+tq\mid t\in\mathbb{R}\}$, where $p=(x_1,\cdots,x_{d-1},0)$ is the intersection point of ℓ with the hyperplane $x_d=0$ and $q=(x_d,\ldots,x_{2d-1})$, $\|q\|^2=1$ is the orientation of ℓ. Then for any point $v\in\mathbb{R}^d$, we have that

$$\mathrm{d}(\ell,v)=\|(p-v)-\langle p-v,q\rangle q\|,$$

where the polynomial $\mathrm{d}^2(\ell,v)$ admits a linearization of dimension $O(d^2)$. Now, proceeding as for the minimum enclosing ball problem and using Theorem 21, we can obtain a coreset $\mathcal{S}$ consisting $N=O\big(\varepsilon^{-O(d^2)}\big)$ deterministic point sets $P_i\subseteq\mathcal{P}$.

We briefly sketch how to obtain the optimal solution for the coreset. We can also decompose $\mathbb{R}^{2d-1}$ (a point x in the space with $\|(x_d, \ldots, x_{2d-1})\| = 1$ represents a line in $\mathbb{R}^d$) into $O\big(\varepsilon^{-O(d^2)}\big)$ semi-algebraic cells such that for each cell, the ordering of the points in $\mathcal{S}$ (by their distances to a line in the cell) is fixed. Note that such a cell is a semi-algebraic cell. For a cell C, assume that $v_i = \arg\max_{v\in P_i} \mathrm{d}(\ell, v_i)$ for all $i \in [N]$, where ℓ is an arbitrary line in C. We can formulate the problem as the following polynomial system:

$$\min_{l} \frac{1}{N}\sum_i d_i, \quad \text{s.t.} \quad d_i^2 = \mathrm{d}^2(\ell, v_i), d_i \geq 0, \forall i \in [N]; \ell = (p,q) \in C_0, \|q\|^2 = 1.$$

Again the polynomial system has a constant number of variables and constraints. Thus, we can compute the optimum in constant time. We summarize our results by Theorem 9.

References

1 A. Abdullah, S. Daruki, and J.M. Phillips. Range counting coresets for uncertain data. In *Proceedings 29th ACM Syposium on Computational Geometry*, pages 223–232, 2013.

2 Marcel R Ackermann, Johannes Blömer, and Christian Sohler. Clustering for metric and nonmetric distance measures. *ACM Transactions on Algorithms (TALG)*, 6(4):59, 2010.

3 P. Afshani, P.K. Agarwal, L. Arge, K.G. Larsen, and J.M. Phillips. (Approximate) uncertain skylines. In *Proceedings of the 14th International Conference on Database Theory*, pages 186–196, 2011.

4 P.K. Agarwal, S.-W. Cheng, and K. Yi. Range searching on uncertain data. *ACM Transactions on Algorithms (TALG)*, 8(4):43, 2012.

5 P.K. Agarwal, A. Efrat, S. Sankararaman, and W. Zhang. Nearest-neighbor searching under uncertainty. In *Proceedings of the 31st Symposium on Principles of Database Systems*, pages 225–236, 2012.

6 P.K. Agarwal, S. Har-Peled, S. Suri, H. Yıldız, and W. Zhang. Convex hulls under uncertainty. In *Proceedings of the 22nd Annual European Symposium on Algorithms*, pages 37–48, 2014.

7 P.K. Agarwal, S. Har-Peled, and K.R. Varadarajan. Approximating extent measures of points. *Journal of the ACM*, 51(4):606–635, 2004.

8 P.K. Agarwal, S. Har-Peled, and K.R. Varadarajan. Geometric approximation via coresets. *Combinatorial and Computational Geometry*, 52:1–30, 2005.

9 P.K. Agarwal and M. Sharir. Arrangements and their applications. *Handbook of Computational Geometry,* J. Sack and J. Urrutia (eds.), pages 49–119. Elsevier, Amsterdam, The Netherlands, 2000.

10 Martin Anthony and Peter L Bartlett. *Neural network learning: Theoretical foundations.* cambridge university press, 2009.

11 D. Bandyopadhyay and J. Snoeyink. Almost-Delaunay simplices: Nearest neighbor relations for imprecise points. In *Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms*, pages 410–419, 2004.

12 Saugata Basu, Richard Pollack, and M Roy. Algorithms in real algebraic geometry. *AMC*, 10:12, 2011.

13 T.M. Chan. Faster core-set constructions and data-stream algorithms in fixed dimensions. *Computational Geometry: Theory and Applications*, 35:20–35, 2006.

14 K. Chen. On coresets for k-median and k-means clustering in metric and euclidean spaces and their applications. *SIAM Journal on Computing*, 39(3):923–947, 2009.

15 R. Cheng, J. Chen, and X. Xie. Cleaning uncertain data with quality guarantees. *Proceedings of the VLDB Endowment*, 1(1):722–735, 2008.

16 G. Cormode and A. McGregor. Approximation algorithms for clustering uncertain data. In *Proceedings of the 27th Symposium on Principles of Database Systems*, pages 191–200, 2008.

17 A. Deshpande, L. Rademacher, S. Vempala, and G. Wang. Matrix approximation and projective clustering via volume sampling. In *Proceedings of the 17th ACM-SIAM symposium on Discrete algorithm*, pages 1117–1126, 2006.

18 X. Dong, A.Y. Halevy, and C. Yu. Data integration with uncertainty. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, pages 687–698, 2007.

19 A. Driemel, H. HAverkort, M. Löffler, and R.I. Silveira. Flow computations on imprecise terrains. *Journal of Computational Geometry*, 4:38–78, 2013.

20 W. Evans and J. Sember. The possible hull of imprecise points. In *Proceedings of the 23rd Canadian Conference on Computational Geometry*, 2011.

21 D. Feldman, A. Fiat, H. Kaplan, and K. Nissim. Private coresets. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, pages 361–370, 2009.

22 D. Feldman and M. Langberg. A unified framework for approximating and clustering data. In *Proceedings of the 43rd ACM Symposium on Theory of Computing*, pages 569–578, 2011.

23 Dan Feldman and Leonard J Schulman. Data reduction for weighted and outlier-resistant clustering. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 1343–1354. SIAM, 2012.

24 Martin Fink, John Hershberger, Nirman Kumar, and Subhash Suri. Hyperplane seperability and convexity of probabilistic point sets. In *Proceedings Symposium on Computational Geometry*, 2016.

25 P.K. Ghosh and K.V. Kumar. Support function representation of convex bodies, its application in geometric computing, and some related representations. *Computer Vision and Image Understanding*, 72(3):379–403, 1998.

26 S. Guha and K. Munagala. Exceeding expectations and clustering uncertain data. In *Proceedings of the 28th Symposium on Principles of Database Systems*, pages 269–278, 2009.

27 L.J. Guibas, D. Salesin, and J. Stolfi. Constructing strongly convex approximate hulls with inaccurate primitives. *Algorithmica*, 9:534–560, 1993.

28 S. Har-Peled. On the expected complexity of random convex hulls. *arXiv:1111.5340*, 2011.

29 S. Har-Peled and S. Mazumdar. On coresets for k-means and k-median clustering. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, pages 291–300, 2004.

30 Sariel Har-Peled and Yusu Wang. Shape fitting with outliers. *SIAM Journal on Computing*, 33(2):269–285, 2004.

31 M. Held and J.S.B. Mitchell. Triangulating input-constrained planar point sets. *Information Processing Letters*, 109(1):54–56, 2008.

32 Lingxiao Huang and Jian Li. Approximating the expected values for combinatorial optimization problems over stochastic points. In *The 42nd International Colloquium on Automata, Languages, and Programming*, pages 910–921. Springer, 2015.

33 A.G. Jørgensen, M. Löffler, and J.M. Phillips. Geometric computation on indecisive points. In *Proceedings of the 12th Algorithms and Data Structure Symposium*, pages 536–547, 2011.

34 P. Kamousi, T.M. Chan, and S. Suri. The stochastic closest pair problem and nearest neighbor search. In *Proceedings of the 12th Algorithms and Data Structure Symposium*, pages 548–559, 2011.

35 P. Kamousi, T.M. Chan, and S. Suri. Stochastic minimum spanning trees in euclidean spaces. In *Proceedings of the 27th Symposium on Computational Geometry*, pages 65–74, 2011.

36 H. Kruger. Basic measures for imprecise point sets in $\mathbb{R}^d$. Master's thesis, Utrecht University, 2008.

37 M. Langberg and L.J. Schulman. Universal ε-approximators for integrals. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms*, 2010.

38 J. Li and H. Wang. Range queries on uncertain data. *Theoretical Computer Science*, 609(1):32–48, 2016.

39 M. Löffler and J. Phillips. Shape fitting on point sets with probability distributions. In *Proceedings of the 17th European Symposium on Algorithms*, pages 313–324, 2009.

40 M. Löffler and J. Snoeyink. Delaunay triangulations of imprecise points in linear time after preprocessing. In *Proceedings of the 24th Sympoium on Computational Geometry*, pages 298–304, 2008. doi:10.1145/1377676.1377727.

41 M. Löffler and M. van Kreveld. Approximating largest convex hulls for imprecise points. *Journal of Discrete Algorithms*, 6:583–594, 2008.

42 J. Matoušek. Computing the center of planar point sets. *Discrete and Computational Geometry*, 6:221, 1991.

43 A. Munteanu, C. Sohler, and D. Feldman. Smallest enclosing ball for probabilistic data. In *Proceedings of the 30th Annual Symposium on Computational Geometry*, 2014.

44 T. Nagai and N. Tokura. Tight error bounds of geometric problems on convex objects with imprecise coordinates. In *Jap. Conf. on Discrete and Comput. Geom.*, LNCS 2098, pages 252–263, 2000.

45 Y. Ostrovsky-Berman and L. Joskowicz. Uncertainty envelopes. In *Abstracts of the 21st European Workshop on Comput. Geom.*, pages 175–178, 2005.

46 Jeff M. Phillips. Coresets and sketches. In *Handbook of Discrete and Computational Geometry*. CRC Press, 3rd edition, 2016. Chapter 49.

47 D. Salesin, J. Stolfi, and L.J. Guibas. Epsilon geometry: building robust algorithms from imprecise computations. In *Proceedings of the 5th Symposium on Computational Geometry*, pages 208–217, 1989.

48 R. Schneider. *Convex bodies: the Brunn-Minkowski theory*, volume 44. Cambridge University Press, 1993.

49 S. Suri, K. Verbeek, and H. Yıldız. On the most likely convex hull of uncertain points. In *Proceedings of the 21st European Symposium on Algorithms*, pages 791–802, 2013.

50 M. van Kreveld and M. Löffler. Largest bounding box, smallest diameter, and related problems on imprecise points. *Computational Geometry: Theory and Applications*, 43:419–433, 2010. doi:10.1016/j.comgeo.2009.03.007.

51 V.N. Vapnik and A.Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability & Its Applications*, 16(2):264–280, 1971.

52 Jie Xue, Yuan Li, and Ravi Janardan. On the separability of stochasitic geometric objects, with applications. In *Proceedings Symposium on Computational Geometry*, 2016.

53 H. Yu, P.K. Agarwal, R. Poreddy, and K. Varadarajan. Practical methods for shape fitting and kinetic data structures using coresets. *Algorithmica*, 52(378-402), 2008.

54 K. Zheng, G. Trajcevski, X. Zhou, and P. Scheuermann. Probabilistic range queries for uncertain trajectories on road networks. In *Proceedings of the 14th International Conference on Extending Database Technology*, pages 283–294, 2011.

Every Property Is Testable on a Natural Class of Scale-Free Multigraphs*

Hiro Ito

School of Informatics and Engineering, The University of Electro-Communications (UEC)/Tokyo, Japan; and
CREST, JST/Tokyo, Japan
itohiro@uec.ac.jp

Abstract

In this paper, we introduce a natural class of multigraphs called hierarchical-scale-free (HSF) multigraphs, and consider constant-time testability on the class. We show that a very wide subclass of HSF is hyperfinite. Based on this result, an algorithm for a deterministic partitioning oracle can be constructed. We conclude by showing that every property is constant-time testable on the above subclass of HSF. This algorithm utilizes findings by Newman and Sohler of STOC'11. However, their algorithm is based on a bounded-degree model, while it is known that actual scale-free networks usually include hubs, which have a very large degree. HSF is based on scale-free properties and includes such hubs. This is the first universal result of constant-time testability on a class of graphs made by a model of scale-free networks, and it has the potential to be applicable on a very wide range of scale-free networks.

1998 ACM Subject Classification G.2.2 Graph Theory: Graph algorithms, G.3 Probability and Statistics: Probabilistic algorithms

Keywords and phrases constant-time algorithms, scale-free networks, complex networks, isolated cliques, hyperfinite

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.51

1 Introduction

How to handle big data is a very important issue in computer science. In the theoretical area, developing efficient algorithms for handling big data is an urgent task. For this purpose, constant-time algorithms look like they could be powerful tools, as they are able to read very small parts (constant size) of inputs.

Property testing is the most well-studied area in constant-time algorithms. A testing algorithm (or a tester) for a property accepts an input if it has the stipulated property and rejects it if it is far away from having the stipulated property with a high probability (e.g., at least 2/3) by reading a constant part of the input. A property is said to be testable if there is a tester [10].

Property testing of graph properties has been well studied and many fruitful results have been obtained [2, 3, 7, 10, 11, 12, 13, 18, 20, 22, 23]. Testers on the graphs are separated into three groups according to model: the dense-graph model (the adjacent-matrix model), the bounded-degree model, and the general model. The dense-graph model is the best clarified: In this model, the characteristics of testable properties have been obtained [2]. However,

* This work was partially supported by the Algorithms on Big Data project (ABD14) of CREST, JST, the ELC project (MEXT KAKENHI Grant Number 24106003), and JSPS KAKENHI Grant Numbers 24650006 and 15K11985.

graphs based on actual networks are usually sparse and thus unfortunately the dense-graph model does not fit. Studies on the bounded-degree model have been proceeding recently. One of the most important findings for this model is that every minor-closed property is testable [3]. This result can be extended to the surprising result that every property of a hyperfinite graph is testable [23]. However, graphs based on actual models have no degree bounds, i.e., it is known that web-graphs have hubs [1, 17], which have a large degree, and, unfortunately once again, these algorithms do not work for them.

Typical big-data graph models are scale-free networks, which are characterized by the power-law degree distribution. Many models have been proposed for scale-free networks [1, 4, 5, 6, 9, 17, 21, 24, 25, 26, 27]. Recently, a promising model based on another property of a hierarchical isomorphic structure has been presented: If we look at a graph in a broad perspective, we find a similar structure to local structures. Shigezumi, Uno, and Watanabe [25] presented a model that is based on the idea of the hierarchical isomorphic structure of power-law distribution of isolated cliques. An idea of isolated cliques was given by Ito and Iwama [15, 16], and the definition is as follows. For a nonnegative integer $c \geq 0$, a *c-isolated clique* is a clique such that the number of outgoing edges (edges between the clique and the other vertices) is less than ck, where k is the number of vertices of the clique. A 1-isolated clique is sometimes simply called an *isolated clique*.

Based on the model of [25], we introduce a class of multigraphs, hierarchical scale-free multigraphs (HSF, Definitions 1.8)[1], which represents natural scale-free networks. We show the following result (Theorem 1.10):

Every property is testable on HSF if the power-law exponent[2] *is greater than two.*

Given this result, many problems on actual scale-free big networks will prove to be solvable in constant time. Although this result is an application of the algorithms of [23], which is a result on bounded-degree graphs, HSF is not a class of bounded-degree graphs. This is the first universal result of constant-time testability on a class of graphs made by a model of scale-free networks.

1.1 Definitions

In this paper, we consider undirected multigraphs without self-loops. We simply call this type of multigraph a "graph" in this paper and use $G = (V, E)$ to denote it, where V is the vertex set and E is the edge (multi)set. Sometimes V and E are denoted by $V[G]$ and $E[G]$, respectively. Henceforth, we use "set" to refer to a multiset for notational simplicity. Throughout this paper, n is used to denote the number of vertices of a graph, i.e., $|V| = n$.

For a graph $G = (V, E)$ and vertex subsets $X, Y \subseteq V$, $E_G(X, Y)$ denotes the edge set between X and Y, i.e., $E_G(X, Y) = \{(x, y) \in E \mid x \in X, y \in Y\}$. $E_G(X, V \backslash X)$ is also simply written as $E_G(X)$. $|E_G(X)|$ is denoted by $d_G(X)$. For a vertex $v \in V$, the number of edges incident to v is called the *degree* of v. A singleton set $\{x\}$ is often written as x for notational simplicity. E.g., the degree of v is represented by $d_G(v)$. The subscript G in the above $E_G(*)$, $d_G(*)$, etc., may be omitted if it is clear.

For a vertex $v \in V$, $\Gamma_G(v)$ denotes the set of vertices adjacent to v, i.e., $\Gamma_G(v) := \{u \in V \mid (v, u) \in E\}$. Note that $|\Gamma_G(v)|$ may not be equal to $d_G(v)$ as parallel edges may exist.

[1] In a preliminary version of this paper, [14], the definition of HSF is different. The definition in this paper is far more general (wider) than in the preliminary version.

[2] This is a parameter of HSF. For the definition, see the sentence just after Definitions 1.7.

For a graph $G = (V, E)$ and a vertex subset $X \subseteq V$, the *subgraph induced by* X is defined as $G(X) = (X, \{(u, v) \in E \mid u, v \in X\})$.

For a vertex subset $X \subseteq V$, a *contraction* of X is defined as an operation to (i) replace X with a new vertex v_X, (ii) replace each edge (v, u) in $E(X)$ $(v \in X, u \in V \backslash X)$ with a new edge (v_X, u), and (iii) remove all edges between vertices in X. That is, by contracting $X \subseteq V$, a graph $G = (V, E)$ is changed to $G' = (V', E')$ such that

$$V' = V \backslash X \cup \{v_X\}, \text{ and}$$
$$E' = E \backslash \{(v, u) \mid v \in X, u \in V\} \cup \{(v_X, u) \mid (v, u) \in E, v \in X, u \in V - X\}.$$

We identify the above $(v_X, u) \in E'$ with $(v, u) \in E$. In other words, we say that (v, u) remains in G' (as (v_X, u)). Note that the graphs are multigraphs, and thus if there are two edges $(v, u), (v', u) \in E$ for $v, v' \in X$, $v \neq v'$ and $u \in V \backslash X$, then two parallel edges, both represented by (v_X, u), one of which corresponds to (v, u) and the other of which corresponds to (v', u), are added to E'. Also note that none of the graphs considered in this paper contain self-loops, and hence an edge $(v, v') \in E$ with $v, v' \in X$ is removed by contracting X.

Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are *isomorphic* if there is a one-to-one correspondence $\Phi : V_1 \to V_2$ such that $E_{G_1}(u, v) = E_{G_2}(\Phi(u), \Phi(v))$ for all $u, v \in V_1$. A graph property (or property, for short) is a (possibly infinite) family of graphs, which is closed under isomorphism.

▶ **Definition 1.1** (ϵ-far and ϵ-close)**.** Let $G = (V, E)$ and $G' = (V', E')$ be two graphs with $|V| = |V'| = n$ vertices. Let $m(G, G')$ be the number of edges that need to be deleted and/or inserted from G in order to make it isomorphic to G'. The distance between G and G' is defined as[3] $\text{dist}(G, G') = m(G, G')/n$. We say that G and G' are *ϵ-far* if $\text{dist}(G, G) > \epsilon$; otherwise *$\epsilon$-close*. Let P be a non-empty property. The distance between G and P is $\text{dist}(G, P) = \min_{G'' \in P} \text{dist}(G, G'')$. We say that G is *ϵ-far* from P if $\text{dist}(G, P) > \epsilon$; otherwise *$\epsilon$-close*.

▶ **Definition 1.2** (testers)**.** A *testing algorithm* for a property P is an algorithm that, given query access to a graph G, accepts every graph from P with a probability of at least 2/3, and rejects every graph that is ϵ-far from P with probability at least 2/3. Oracles in the general graph model are: for any vertex v, the algorithm may ask for the degree $d(v)$, and may ask for the ith neighbor of the vertex (for $1 \le i \le d(v)$).[4] The number of queries made by an algorithm to the given oracle is called the *query complexity* of the algorithm. If the query complexity of a testing algorithm is a constant, independent of n (but it may depend on ϵ), then the algorithm is called a *tester*[5]. A (graph) property is *testable* if there is a tester for the property.

▶ **Definition 1.3** (isolated cliques [15])**.** For a graph $G = (V, E)$ and a real number $c \ge 0$, a vertex subset $Q \subseteq V$ is called a *c-isolated clique* if Q is a clique (i.e., $(u, v) \in E$, for all $u, v \in Q$ and $u \neq v$) and $d_G(Q) < c|Q|$. A 1-isolated clique is sometimes called an *isolated clique*. In this paper, we don't use $c > 1$ except section **4** (summary and future work).

[3] The distance defined here may be larger than 1 as $m(G, G') > n$ may occur. (In the bounded-degree model it is defined as $\text{dist}(G, G') = m(G, G')/dn$.) However, here we consider sparse graphs and they have an implicit upper bound of the average (not possibly maximum) degree, say d, and thus $\text{dist}(G, G')$ is bounded by d.

[4] Although asking whether there is an edge between any two vertices is also allowed in the general graph model, the algorithms we use in this paper do not need to use this query.

[5] In this paper, a tester may be nonuniform, i.e., it may depend on n and ϵ.

▶ **Definition 1.4.** Let $\mathcal{E}(G)$ be the graph obtained from G by contracting all isolated cliques. Two distinct isolated cliques never overlap, except in the special case of *double-isolated-cliques*, which consists of two isolated cliques with size k sharing $k-1$ vertices. A double-isolated-clique Q has no edge between Q and the other part of the graph (i.e., $d_G(Q) = 0$), and thus we specially define that a double-isolated-clique in G is contracted into a vertex in $\mathcal{E}(G)$. Under this assumption, $\mathcal{E}(G)$ is uniquely defined.

▶ **Definition 1.5** (hyperfinite [8]). For real numbers $t > 0$ and $\epsilon > 0$, a graph $G = (V, E)$ consisting of n vertices is (t, ϵ)*-hyperfinite* if one can remove at most ϵn edges from G and obtain a graph whose connected components have size at most t. For a function $\rho : \boldsymbol{R}^+ \to \boldsymbol{R}^+$, G is ρ*-hyperfinite* if it is $(\rho(\epsilon), \epsilon)$-hyperfinite for all $\epsilon > 0$. A family $\mathcal{G}$ of graphs is ρ*-hyperfinite* if all $G \in \mathcal{G}$ are ρ-hyperfinite. A family $\mathcal{G}$ of graphs is *hyperfinite* if there exists a function ρ such that $\mathcal{G}$ is ρ-hyperfinite.

Hyperfinite is a large class, as it is known that any minor-closed property is hyperfinite in a bounded-degree model. From the viewpoint of testing, the importance of hyperfiniteness stems from the following result.

▶ **Theorem 1.6** ([23]). *For the bounded-degree model, any property is testable for any class of hyperfinite graphs.*

This result is very strong, but there is a problem in that the result works on bounded-degree graphs and it is natural to consider that actual scale-free networks do not have a degree bound.

1.2 Our contribution and related work

In this paper, we apply the universal algorithm of [23] to scale-free networks. We formalize two natural classes, $\mathcal{SF}$ and $\mathcal{HSF}$ that represent scale-free networks[6]. The latter is a subclass of the former.

▶ **Definition 1.7.** For positive real numbers $c > 1$ and $\gamma > 1$, a class of *scale-free graphs (SF)* $\mathcal{SF}(c, \gamma)$ consists of (multi)graphs $G = (V, E)$ for which the following condition holds: Let ν_i be the number of vertices v with $d(v) = i$. Then:

$$\nu_i \le cni^{-\gamma}, \quad \forall i \in \{2, 3, \dots, \}. \tag{1}$$

The above property (1) is generally called a power-law and we call γ a *power-law exponent.* In many actual scale-free networks, it is said that $2 < \gamma < 3$ [1]. That is, $\mathcal{SF}$ is a class of multigraphs that obey the power-law degree distribution.

We show that this class is ϵ-close to a bounded-degree class if $\gamma > 2$ (Lemma 2.1).

After showing this property, we show the hyperfiniteness of the class. Hyperfiniteness seems to be closely related to a high clustering coefficient, where the cluster coefficient $\mathrm{cl}(G)$ of a graph $G = (V, E)$ is defined as[7]:

$$\mathrm{cl}(G) := \frac{1}{n}\sum_{v \in V} \mathrm{cl}_G(v), \qquad \mathrm{cl}_G(v) := \frac{|\{(u, w) \in E \mid u, w \in \Gamma_G(v), u \ne w\}|}{\binom{|\Gamma_G(v)|}{2}}.$$

[6] $\mathcal{HSF}$ was introduced in the preliminary version of this paper [14]. However, the definition in this paper is more general (wider) than in the preliminary version.

[7] There is another way to define the cluster coefficient: $3 \times$ (# of cycles of length three)/(# of paths of length two). Although these two values are different generally, they are close under the assumption of the power-law degree distribution.

Sometimes $\mathrm{cl}_G(v)$ is called the *local cluster coefficient* of v. It is said that $\mathrm{cl}(G)$ is $\Theta(1)$ for many classes that model actual social networks, while $\lim_{n\to\infty} \mathrm{cl}(G) = 0$ for random graphs.

These three characterizations, "high clustering coefficient," "existence of isolated cliques," and "hyperfiniteness" appear to be closely related to each other. In fact, it is readily observed that if $\mathrm{cl}_G(v) = 1$ for a bounded-degree graph G (the degree bound is d), then G consists of only (completely) isolated cliques with size at most $d + 1$, and G is $(d + 1, 0)$-hyperfinite!

Unfortunately, however, it is also observed that for any $0 < c < 1$, there is a class of bounded-degree graphs G such that $\lim_{n\to\infty} \mathrm{cl}(G) = c$ and it is not (t, ϵ)-hyperfinite for any pair of constants t and $\epsilon < 1/2$, e.g., $G = (V, E)$ consists of n/d cliques of size d, and random $n/2$ edges between vertices in different cliques (each vertex has $d - 1$ adjacent vertices in its clique and one adjacent vertex outside the clique). To separate this graph into constant-sized connected components, almost all of the edges between cliques (their number is $n/2$) must be removed.

However, we do not need to give up here, as the above model is very special, e.g., by contracting each isolated clique, it becomes a mere random graph with n/d vertices[8]. From this fact, the hierarchical structure of a high cluster coefficient looks important. The model presented by [25] has such a structure. Based on this model, we present the following class of multigraphs:

▶ **Definition 1.8** (Hierarchical Scale-Free Graphs)**.** For positive real numbers $c, \gamma > 1$ and a positive integer $n_0 \geq 1$, a class of *hierarchical scale-free graphs (HSF)* $\mathcal{HSF} = \mathcal{HSF}(c, \gamma, n_0)$ consists of (multi)graphs $G = (V, E)$ for which the following conditions hold:

(i) $G \in \mathcal{SF}(c, \gamma)$

(ii) Consider the infinite sequence of graphs $G_0 = G$, $G_1 = \mathcal{E}(G_0)$, $G_2 = \mathcal{E}(G_1)$, $\ldots$. If $|V[G_i]| \geq n_0$, then G_i includes at least one isolated clique $Q \subseteq V$ with $|Q| \geq 2$. (Note that if G_k has no such isolated clique, then $G_k = G_{k+1} = G_{k+2} = \cdots$.)

We show the following results.

▶ **Theorem 1.9.** *For any $\mathcal{HSF} = \mathcal{HSF}(c, \gamma, n_0)$ with $\gamma > 2$ and any real number $\epsilon > 0$, there is a real number $t_{1.9} = t_{1.9}(\mathcal{HSF}, \epsilon)$ such that $\mathcal{HSF}$ is $(t_{1.9}, \epsilon)$-hyperfinite.*

We give a global algorithm for obtaining the partition realizing the hyperfiniteness of Theorem 1.9. The algorithm is deterministic, i.e., if a graph and the parameter ϵ are fixed, then the partition is also fixed. The algorithm can be easily revised to a local algorithm and we obtain a deterministic partitioning oracle to get the partition (Lamma 3.2). Note that almost all algorithms for partitioning oracles presented to date have been randomized algorithms[9]. By using this partitioning oracle and an argument similar to one used in [23], we get the following main theorem.

▶ **Theorem 1.10.** *Any property is testable for $\mathcal{HSF}(c, \gamma, n_0)$ with $\gamma > 2$.*

As stated earlier, for the bounded-degree model, Newman and Sohler [23] presented a universal tester (which can test any property) for hyperfinite graphs. In the general graph model, although some works have tried to found universal tester [7, 18, 22], these results are weaker than for the bounded-degree graph model and the dense graph model.

[8] However, note that this model is not useless, since it is investigated in some works [19].

[9] The algorithm for testing forests presented by Kusumi and Yoshida [18] may be only deterministic one so far. That is, our partitioning oracle looks the first deterministic one for a graph class that includes cyclic graphs.

This paper gives a universal tester that can test every property on a natural class of scale-free multigraphs in constant time. This is the first result for universal constant-time algorithms which cover a class of graphs made by a model of scale-free networks.

2 Hyperfiniteness and a Global Partitioning Algorithm

2.1 Degree bounding

For a graph G and a nonnegative integer $d \geq 0$, $G|d$ is a graph made by deleting all edges incident to each vertex v with $d(v) > d$ from G. Note that $G|d$ is a bounded-degree graph with degree bound d.

▶ **Lemma 2.1.** *For any $\mathcal{SF} = \mathcal{SF}(c, \gamma)$ with $\gamma > 2$, and any positive real number $\epsilon > 0$, there is a constant $\delta_{2.1} = \delta_{2.1}(\epsilon, c, \gamma)$ such that for any graph $G \in \mathcal{SF}$, $G|\delta_{2.1}$ is ϵ-close to G.*

Before showing a proof of this lemma, we introduce some definitions. Riemann zeta function is defined by $\zeta(\gamma) = \sum_{i=1}^{\infty} i^{-\gamma}$. This function is known to converge to a constant ($\zeta(\gamma) < 1 + (\gamma - 1)^{-1}$) for any $\gamma > 1$. We introduce a generalization of this function by using a positive integer $k \geq 1$ as $\zeta(k, \gamma) = \sum_{i=k}^{\infty} i^{-\gamma}$. Note that $\zeta(\gamma) = \zeta(1, \gamma)$.

▶ **Lemma 2.2.** *For any $\epsilon > 0$ and $\gamma > 1$, there is an integer $k_{2.2} = k_{2.2}(\epsilon, \gamma) \geq 1$ such that $\zeta(k_{2.2}, \gamma) < \epsilon$.*

Proof. It is clear from the above fact that $\zeta(\gamma)$ converges for every $\gamma > 1$. ◀

Proof of Lemma 2.1. Let d be an arbitrary positive integer. Let m_d be the number of removed edges to make $G|d$ from G. From (1),

$$m_d = \sum_{i=d+1}^{\infty} i\nu_i \leq \sum_{i=d+1}^{\infty} cni^{-(\gamma-1)} = cn\zeta(d+1, \gamma - 1).$$

From the assumption of $\gamma > 2$ and Lemma 2.2, $\zeta(d+1, \gamma-1) < \epsilon/c$ if $d+1 \geq k_{2.2}(\epsilon/c, \gamma - 1)$. Thus by letting $\delta_{2.1}(\epsilon, c, \gamma) = k_{2.2}(\epsilon/c, \gamma - 1) - 1$, we have $m_{\delta_{2.1}} < \epsilon n$. ◀

From here, we denote the above $\delta_{2.1}(\epsilon, c, \gamma)$ by δ for notational simplicity.

2.2 Hierarchical contraction, structure tree, and coloring

Let $W_1, \dots, W_k$ ($W_i \subseteq V$, $\forall i \in \{1, \dots, k\}$) be a family of subsets of vertices satisfying that $W_i \cap W_j = \emptyset$ for every $i, j \in \{1, \dots, k\}$ and $i \neq j$, and $W_1 \cup \cdots \cup W_k = V$. Then $\{W_1, \dots, W_k\}$ is called a *partition* of V. Below, we explain a global algorithm for obtaining a partition of V realizing the hyperfiniteness of a graph in $\mathcal{HSF}$ with $\gamma > 2$, i.e., $|W_i|$ is bounded by a constant and the number of edges between different W_i and W_j is, at most, ϵn. First, we give a base algorithm.

```
procedure HierarchicalContraction($G$)
begin
1    $i := 0$, $G_0 := G$
2    while there exists an isolated clique in $G_i = (V_i, E_i)$ do
3       $i := i + 1$, $G_i := \mathcal{E}(G_{i-1})$
4    enddo
end.
```

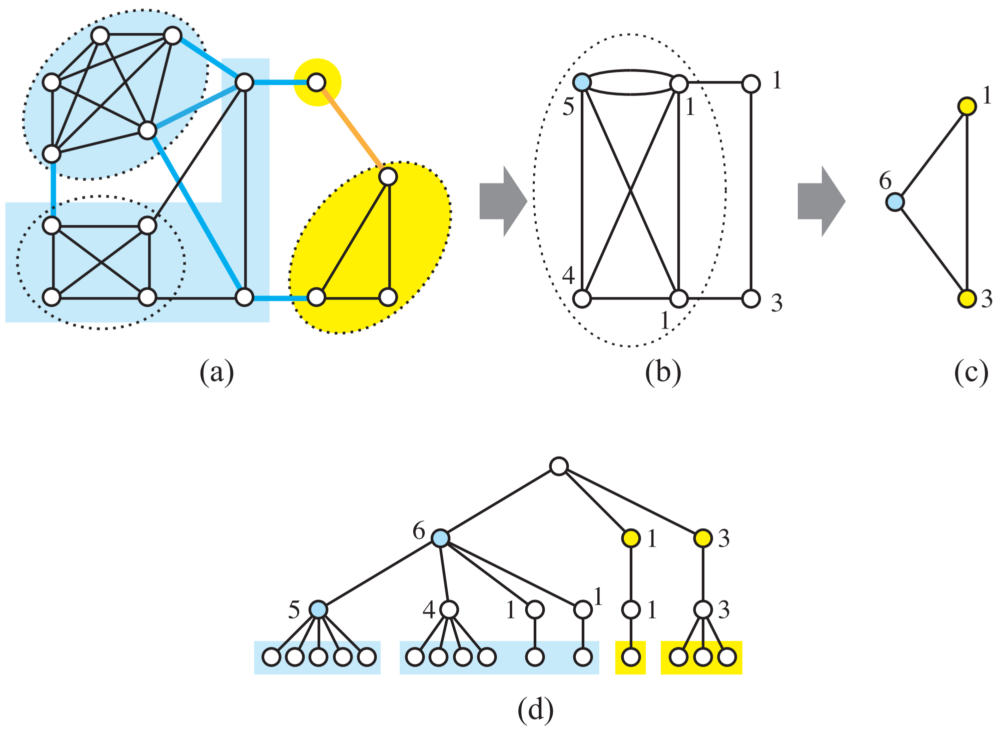


Figure 1 An example of HierarchicalContraction, the structure tree T, and the coloring: Here, we assume $\delta/\epsilon = 4.5$; the number beside a vertex is $w(*)$; the dotted circles are isolated cliques; colored areas are blue or yellow components.

We denote $G_i = (V_i, E_i)$ for $i \in \{0, 1, \ldots\}$. Let $G_k = (V_k, E_k)$ be the final graph of HierarchicalContraction(G). From the definitions of HSF, $|V_k| < n_0$. See Fig. 1 (a)–(c) for an example of applying this procedure.

The trail of the contraction can be represented by a rooted tree $T = (V[T], E[T])$, which is called the *structure tree* of G, defined as follows. (Fig. 1 (d) shows an example of the structure tree[10].)

$V[T] := V_0 \cup V_1 \cup \cdots \cup V_k \cup \{r\}$, where r is the (artificial) root of T. Each $v \in V_0$ is a leaf of T, and a vertex $v \in V_i$ $(i \in \{0, \ldots, k\})$ is on the level i of T, i.e., $v \in V_i$ $(i \geq 1)$ is the parent of $u \in V_{i-1}$ if "v is made by contracting a subset (an isolated-clique or a double-isolated-clique) $Q \subseteq V_{i-1}$ such that $u \in Q$" or "$v = u$ (i.e., u is not included in an isolated clique in G_{i-1})." The root r is the parent of every vertex in V_k. (The reason r is added is only to make T a tree.)

We introduce a function $W : V[T]\backslash\{r\} \rightarrow 2^V$ and coloring on the vertices in $V[T]$ as follows:

- For $v \in V_0$:
  - $W(v) = \{v\}$, and
  - if $d(v) > \delta$, then v is colored *red*, otherwise uncolored.
- For $v \in V_i$ $(i = 1, \ldots, k)$:
  - let $S(v)$ be the set of uncolored children of v,

[10] In this example, we ignore to color vertices red. Since $\delta < 4.5$ follows $\delta/\epsilon = 4.5$, some vertices in this figure might have to be red.

- $W(v) = \bigcup_{u \in S(v)} W(u)$, and
- if $|W(v)| > \delta/\epsilon$, then v is colored *blue*,
- else if $v \in V_k$ and $W(v) \neq \emptyset$, then v is colored *yellow*,
- otherwise, v is uncolored.

Note that for any two distinct colored vertices $u, v \in V[T]$, $W(u) \cap W(v) = \emptyset$. For every $v \in V[T]$, we also define a weight function as $w(v) = |W(v)|$. For a blue (resp. yellow) colored vertex $v \in V[T]$, $W(v) \subseteq V$ is called a *blue (resp. yellow) component*.

By using these colors, we also color the edges in E $(= E_0)$ in the following manner:

- For every red vertex $v \in V_0$ $(= V)$, all edges in $E_G(v)$ are colored *red*.
- For every blue component $W \subseteq V$, for every edge $e \in E_G(W)$, if e is not colored red, then e is colored *blue*.
- For every yellow component $W \subseteq V$, for every edge $e \in E_G(W)$, if e is not colored either red or blue, then e is colored *yellow*.

The other edges in E are uncolored. The set of red, blue, and yellow edges in E are represented by R, B, and Y, respectively. These colors are preserved in $G_1 = \mathcal{E}(G_0)$, $G_2 = \mathcal{E}(G_1)$, …, $G_k = \mathcal{E}(G_{k-1})$, e.g., if an edge $e \in E_i$ is red, then the corresponding edge in E_{i+1} is also red.

2.3 Proof of Theorem 1.9

Before showing the proof of Theorem 1.9, we prepare some lemmas.

▶ **Lemma 2.3.** *For any G_i $(i \in \{0, \dots, k\})$, all edges incident to a vertex with a degree higher than δ are red.*

Proof. For $G_0 = G$, the statement clearly holds from the coloring rule. Assume that the statement holds in G_{i-1}, and does not hold in some G_i. Let v be a vertex in V_i such that $d_{G_i}(v) \geq \delta + 1$ and a non-red edge is incident to v. Then v must be made by contracting an isolated clique in G_{i-1}, say $Q \subseteq V_{i-1}$, such that $d_{G_{i-1}}(Q) \geq \delta + 1$. From the definition of isolated cliques, $|Q| \geq d_{G_{i-1}}(Q) + 1 \geq \delta + 2$. Since Q is a clique, every vertex Q has degree at least $|Q| - 1 \geq \delta + 1$ in G_{i-1}. It follows that all edges incident to a vertex in Q must be red. This contradicts the assumption that a non-red edge is incident to v. ◀

▶ **Lemma 2.4.** $|R|, |B| < \epsilon n$*,* $|Y| < \delta n_0/2$*.*

Proof. $|R| < \epsilon n$ is directly obtained from Lemma 2.1. Let $v \in V_i$ be a blue vertex such that a non-red edge exists in $E(W(v))$. From Lemma 2.3, $d(W(v)) \leq \delta$. Thus $d(W(v))/w(v) < \delta/(\delta/\epsilon) = \epsilon$. This means that the average number of blue edges per a vertex is less than ϵ. Therefore $|B| < \epsilon n$. From Lemma 2.3, all edges incident to a vertex with degree higher than δ are red. From this it follows that the number of non-red edges in E_k is at most $\delta|V_k|/2$. Thus the number of yellow edges in E is also at most $\delta|V_k|/2$. By considering $|V_k| < n_0$, we have $|Y| < \delta n_0/2$. ◀

Let v_1^R, …, $v_{k_r}^R$ be the red vertices (k_r is the number of red vertices). Let W_1^B, …, $W_{k_b}^B$ be the blue components (k_b is the number of blue components). Let W_1^Y, …, $W_{k_y}^Y$ be the yellow components (k_y is the number of yellow components). We consider a family of vertex subsets as

$$\mathcal{P} := \{\{v_i^R\} \mid i = 1, \dots, k_r\} \cup \{W_i^B \mid i = 1, \dots, k_b\} \cup \{W_i^Y \mid i = 1, \dots, k_y\}.$$

From the definition of the function W and the coloring, $\mathcal{P}$ is clearly a partition of V.

Now we can prove Theorem 1.9.

Proof of Theorem 1.9. If $n \leq \delta n_0/(2\epsilon)$, then the statement is clear by setting $t \geq \delta n_0/(2\epsilon)$. Thus, we assume that $n > \delta n_0/(2\epsilon)$. Let G' be a graph obtained by deleting all red, blue, and yellow edges from G. From Lemma 2.4, the number of deleted edges is less than

$$2\epsilon n + \delta n_0/2 < 3\epsilon n. \tag{2}$$

Next, we will show that the maximum size of connected components in G' is at most $\delta(\delta+1)/\epsilon$. Assume that there exists a connected component $G'(X) = (X, E_X)$ consisting of more than $\delta(\delta+1)/\epsilon$ vertices in G'. X includes no vertex v with $d_G(v) > \delta$, since from Lemma 2.3 all edges in $E_G(v)$ are colored red. Moreover, there is no blue component $W \subseteq V$ such that $X \cap W \neq \emptyset$ and $X \backslash W \neq \emptyset$, as otherwise X would be disconnected in G' (by deleting blue edges).

From this it follows that there is a blue or yellow component $W = W(x)$ such that $X \subseteq W(x)$. If x is a yellow vertex, then $w(x) \leq \delta/\epsilon$ (as otherwise x would be colored blue), and $|X| \leq w(v) \leq \delta/\epsilon < \delta(\delta+1)/\epsilon$, which is a contradiction. Thus x must be a blue vertex. Assume that $x \in V_h$. Let $Z \subseteq V_{h-1}$ be the set of children of x (in T). Z consists of an isolated clique or a double-isolated-clique in G_{h-1}. Let $S(x)(\subseteq Z)$ be the set of uncolored vertices in Z. For every vertex $v \in S(x)$, $d_{G_{h-1}}(v) \leq \delta$ (from Lemma 2.3). From this and the fact that Z consists of an isolated clique or a double-isolated-clique, it follows that $|Z| \leq \delta + 1$.

For $v \in S(x)$, $w(v) \leq \delta/\epsilon$. Hence,

$$w(x) = \sum_{v \in S(x)} w(v) \leq |S(x)| \cdot \delta/\epsilon \leq |Z| \cdot \delta/\epsilon \leq (\delta+1)\delta/\epsilon,$$

which is a contradiction. Therefore, the maximum size of connected components in G' is $\delta(\delta+1)/\epsilon$.

Thus, we have proved that G is $(\max\{\delta n_0/(2\epsilon), \delta(\delta+1)/\epsilon\}, 3\epsilon)$-hyperfinite. Here, ϵ is an arbitrary real number in $(0,1]$, then by defining $t_{1.9} = \max\{3\delta n_0/(2\epsilon), 3\delta(\delta+1)/\epsilon\}$, G is $(t_{1.9}, \epsilon)$-hyperfinite for any $\epsilon > 0$. ◀

3 Testing Algorithm

3.1 Deterministic partitioning oracle

The global partitioning algorithm of Theorem 1.9 can be easily revised to run locally, i.e., a "partitioning oracle" based on this algorithm can be obtained. A partitioning oracle, which calculates a partition realizing hyperfiniteness locally, was introduced by Benjamini, et al. [3] implicitly and by Hassidim, et al. [13] explicitly. It is a powerful tool for constructing constant-time algorithms for sparse graphs. It has been revised by some researchers and Levi and Ron's algorithm [20] is the fastest to date. As mentioned before almost all algorithms for partitioning oracles presented to date have been randomized algorithms. Our algorithm, however, does not use any random valuable and it runs deterministically. That is, we call it a *deterministic partitioning oracle*, which is rigorously defined as follows[11]:

▶ **Definition 3.1.** $\mathcal{O}$ is a deterministic (t, ϵ)-partitioning oracle for a class of graphs $\mathcal{C}$, if, given query access to a graph $G = (V, E)$, it provides query access to a partition $\mathcal{P}$ of G. For a query about $v \in V$, $\mathcal{O}$ returns $\mathcal{P}(v)$. The partition has the following properties: (i) $\mathcal{P}$ is a function of G, t, and ϵ. (It does not depend on the order of queries to $\mathcal{O}$.) (ii) For

[11] However, since Levi and Ron's algorithm [20] looks fast, using it may be better in practice.

every $v \in V$, $|\mathcal{P}(v)| \leq t$ and $\mathcal{P}(v)$ induces a connected subgraph of G. (iii) If $G \in \mathcal{C}$, then $|\{(u,v) \in E \mid \mathcal{P}(u) \neq \mathcal{P}(v)\}| \leq \epsilon|V|$.

▶ **Lemma 3.2.** *There is a deterministic $(t_{1.9}, \epsilon)$-partitioning oracle $\mathcal{O}_{HSF}$ for HSF with $\gamma > 2$ with query complexity $\delta^{O(\delta^2/\epsilon + n_0)}$ for one query.*

Before giving a proof of this lemma, we introduce some notation as follows. A connected graph $G = (V, E)$ with a specified marked vertex v is called a *rooted graph*, and we sometimes say that G is rooted at v. A rooted graph $G = (V, E)$ has a radius t, if every vertex in V has a distance at most t from the root v. Two rooted graphs are isomorphic if there is a graph isomorphism between these graphs that identifies the roots with each other. We denote by $N(d, t)$ the number of all non-isomorphic rooted graphs with a maximum degree of d and a maximum radius of t. For a graph $G = (V, E)$, integers d and t, and a vertex $v \in V$, let $B_G(v, d, t)$ be the subgraph rooted at v that is induced by all vertices of $G|d$ that are at distance t or less from v. $B_G(v, d, t)$ is called a (d, t)-*disk* around v. From these definitions, the number of possible non-isomorphic (d, t)-disks is at most $N(d, t)$.

Proof of Lemma 3.2. The global algorithm of Theorem 1.9 can be easily simulated locally. To find $\mathcal{P}(v)$, if $d(v) > \delta$, then the algorithm outputs $\mathcal{P}(v) := \{v\}$. Otherwise, if the algorithm finds a vertex u with $d(u) > \delta$ in the process of the local search, u is ignored (the algorithm does not check the neighbors of u). Thus, the algorithm behaves as on the bounded-degree model. For any vertex v, $|\mathcal{P}(v)| \leq t_{1.9} = O(\delta^2/\epsilon)$. Each $u \in B_G(v, \delta, t_{1.9})$ may be included in $\mathcal{P}(w)$ of $w \in B_G(u, \delta, t_{1.9})$. Then, the algorithm checks most vertices in $B_G(v, \delta, 2t_{1.9}) = B_G(v, \delta, O(\delta^2/\epsilon + n_0))$, and thus the query complexity for one call of $\mathcal{P}(v)$ is at most $\delta^{O(\delta^2/\epsilon + n_0)}$. ◀

3.2 Abstract of the algorithm

The method of constructing a testing algorithm based on the partitioning oracle of Lemma 3.2 is almost the same as the method used in [23]. We use a distribution vector, which will be defined in Definition 3.3, of rooted subgraphs consisting of at most a constant number of vertices.

▶ **Definition 3.3.** For a graph $G = (V, E)$ and integers d and t, let $\text{disk}_G(d, t)$ be the distribution vector of all (d, t)-disks of G, i.e., $\text{disk}_G(d, t)$ is a vector of dimension $N(d, t)$. Each entry of $\text{disk}_G(d, t)$ corresponds to some fixed rooted graph H, and counts the number of (d, t)-disks of $G|d$ that are isomorphic to H. Note that $G|d$ has $n = |V|$ different disks, thus the sum of entries in $\text{disk}_G(d, t)$ is n. Let $\text{freq}_G(d, t)$ be the normalized distribution, namely $\text{freq}_G(d, t) = \text{disk}_G(d, t)/n$. For a vector $v = (v_1, \ldots, v_r)$, its l_1-norm is $||v||_1 = \sum_{i=1}^{r} |v_i|$. The l_1-norm is also the length of the vector. We say that the two unit-length vectors v and u are ϵ-close for $\epsilon > 0$ if $||v - u||_1 \leq \epsilon$.

By using the same discussion as in Theorem 3.1 in [23], the following lemma is proven.

▶ **Lemma 3.4.** *There exist functions $\lambda_{3.4} = \lambda_{3.4}(\mathcal{HFS}, \epsilon)$, $d_{3.4} = d_{3.4}(\mathcal{HFS}, \epsilon)$, $t_{3.4} = t_{3.4}(\mathcal{HFS}, \epsilon)$, $N_{3.4} = N_{3.4}(\mathcal{HFS}, \epsilon)$ such that for every $\epsilon > 0$ the following holds: For every $G_1, G_2 \in \mathcal{HFS}$ on $n \geq N_{3.4}$ vertices, if $|freq_{G_1}(d_{3.4}, t_{3.4}) - freq_{G_2}(d_{3.4}, t_{3.4})| \leq \lambda_{3.4}$, then G_1 and G_2 are ϵ-close.* ◀

A sketch of the algorithm is as follows. Let $G = (V, E)$ be a given graph and P be a property to test. First, we select some (constant) number $\ell = \ell(\epsilon)$ of vertices $v_i \in V$

$(i = 1, \dots, \ell)$ and find $\mathcal{P}(v_i)$ given by Theorem 1.9. This is done locally (shown by Lemma 3.2). Consider a graph $G' := \mathcal{P}(v_1) \cup \cdots \cup \mathcal{P}(v_\ell)$. Here, $\mathrm{freq}_G(d,t)$ and $\mathrm{freq}_{G'}(d,t)$ are very close with high probability. Next, we calculate $\min_{G \in P} |\mathrm{freq}_{G'}(d,t) - \mathrm{freq}_G(d,t)|$ approximately. There is a problem in that the number of graphs in P is generally infinite. However, to approximate it with a small error is adequate for our objective, and thus it is sufficient to compare G' with a constant number of vectors of $\mathrm{freq}(d,t)$. (Note that calculating such a set of frequency vectors requires much time. However, we can say that there exists such a set. This means that the existence of the algorithm is assured.) The algorithm accepts G if the approximate distance of $\min_{G \in P} |\mathrm{freq}_{G'}(d,t) - \mathrm{freq}_G(d,t)|$ is small enough, and otherwise it is rejected.

The above algorithm is the same as the algorithm presented in [23] except for two points – in our model: (1) G is not a bounded-degree graph, and (2) G is a multigraph. However, these differences are trivial. For the first difference, it is enough to add an ignoring-large-degree-vertex process, i.e., if the algorithm find a vertex v having a degree larger than $d_{3.4}$, all edges incident to v are ignored. By adding this process, G is regarded as $G|d_{3.4}$. This modification does not effect the result by Lemma 2.1. For the second difference, the algorithm treats bounded-degree graphs as mentioned above, and the number of non-isomorphic multigraphs with n vertices and degree upper bound $d_{3.4}$ is finite (bounded by $O({d_{3.4}}^{n^2})$).

Proof of Theorem 1.10. Obtained from the above discussion. ◀

4 Summary and future work

We presented a natural class of multigraphs $\mathcal{HSF}$ representing scale-free networks, and we showed that a very wide subclass of it is hyperfinite (Theorem 1.9). By using this result, the useful result that every property is testable on the class (Theorem 1.10) is obtained.

$\mathcal{HSF}$ is a class of multigraphs based on the hierarchical structure of isolated cliques. We may relax “isolated cliques” to “c-isolated cliques” or “isolated dense subgraphs [15]” and we may introduce a wider class. We consider such classes also to be hyperfinite. Finding such classes and proving their hyperfiniteness is important future work.

Acknowledgements. We are grateful to Associate Professor Yuichi Yoshida of the National Institute of Informatics for his valuable suggestions. We also appreciate the fruitful discussions with Professor Ilan Newman of University of Haifa, Professor Osamu Watanabe of the Tokyo Institute of Technology, and Associate Professor Yushi Uno of Osaka Prefecture University. We also would like to thank the anonymous referees for their variable comments. Finally, we thank the Algorithms on Big Data project (ABD14) of CREST, JST, the ELC project (MEXT KAKENHI Grant Number 24106003), and JSPS KAKENHI Grant Numbers 24650006 and 15K11985 through which this work was partially supported.

References

1 R. Albert and A.-L. Barabási: Statistical mechanics of complex networks, Review of Modern Physics, Vol. 74, 2002, pp. 47–97.
2 N. Alon, E. Fischer, I. Newman, and A. Shapira: A combinatorial characterization of the testable graph properties: it’s all about regularity, SIAM J. Comput., 39(1):143–167, 2009.
3 I. Benjamini, O. Schramm, and A. Shapira: Every minor-closed property of sparse graphs is testable, Proc. STOC 2008, ACM, 2008, pp. 393–402.
4 A. Bottreau, Y. Métivier: Some remarks on the Kronecker product of graphs, Information Processing Letters, Vol. 68, Elsevier, 1998, pp. 55–61.

5 A. Z. Broder, S. R. Kumar, F. Maghoul, R. Raghavan, S. Rajagoplalan, R. Stata, A. Tomkins and J.L. Wiener: Graph structure in the Web, Computer Networks, Vol. 33, 2000, pp.309–320.

6 C. Cooper and R. Uehara: Scale free properties of random k-trees; Mathematics in Computer Science, Vol. 3, 2010, pp. 489–496.

7 A. Czumaj, M. Monemizadeh, K. Onak, and C. Sohler: Planar Graphs: Random Walks and Bipartiteness Testing, Proceedings of FOCS 2011, pp. 423–432.

8 G. Elek: L^2-spectral invariants and convergent sequence of finite graphs, Journal of Functional Analysis, Vol. 254, No. 10, 2008, pp. 2667–2689.

9 Y. Gao: The degree distribution of random k-trees, Theoretical Computer Science, Vol. 410, 2009, pp. 688–695.

10 O. Goldreich (Ed.): Property Testing – Current Research and Surveys, LNSC 6390, 2010.

11 O. Goldreich and D. Ron: Property testing in bounded degree graphs: Proc. STOC 1997, 1997, pp. 406–415.

12 O. Goldreich, S. Goldwasser, and D. Ron: Property testing and its connection to learning and approximation: Journal of the ACM, Vol. 45, No. 4, July, 1998, pp. 653–750.

13 A. Hassidim, J. A. Kelner, H. N. Nguyen, and K. Onak: Local graph partitions for approximation and testing, Proc. FOCS 2009, IEEE, pp. 22–31.

14 H. Ito, Every property is testable on a natural class of scale-free multigraphs, arXiv: 1504.00766, Cornell University, April 6, 2015.

15 H. Ito and K. Iwama, Enumeration of isolated cliques and pseudo-cliques, ACM Transactions on Algorithms, Vol. 5, Issue 4, Oct. 2009, Article 40 (pp. 1–13).

16 H. Ito and K. Iwama, and T. Osumi: Linear-time enumeration of isolated cliques, Proc. ESA2005, LNCS, 3669, Springer, 2005, pp. 119–130.

17 J. Kleinberg and S. Lawrence: The structure of the Web, Science, Vol. 294, 2001, pp. 1894–1895.

18 M. Kusumoto and Y. Yoshida: Testing forest-isomorphizm in the adjacency list model, Proc. of ICALP2014 (1), LNSC 8572, 2014, pp. 763–774.

19 A. Lancichinetti, S. Fortunato, and F. Radicchi: Benchmark graphs for testing community detection algorithms, arXiv: 0805.4770v4, Cornell University, Oct. 3, 2008.

20 R. Levi and D. Ron: A quasi-polynomial time partition oracle for graphs with an excluded minor, Proc. ICALP 2013 (1), LNCS, 7965, Springer, 2013, pp. 709–720. (Journal version: R. Levi and D. Ron: A quasi-polynomial time partition oracle for graphs with an excluded minor, ACM Transactions on Algorithms, Vol. 11, No. 3, 2014, Article 24 (pp. 1–13).)

21 M. Mahdian and Y. Xu: Stochastic Kronecker graphs, Proc. WAW 2007, LNCS, 4863, Springer, 2007, pp. 179–186.

22 S. Marko, D. Ron: Approximating the distance to properties in bounded-degree and general sparse graphs., ACM Transactions on Algorithms, Vol. 5, Issue. 2, 2009, Article No. 22.

23 I. Newman and C. Sohler: Every property of hyperfinite graphs is testable, Proc. STOC 2011, ACM, 2011, pp. 675–784. (Journal version: I. Newman and C. Sohler: Every property of hyperfinite graphs is testable, SIAM J. Comput., Vol. 42, No. 3, 2013, pp. 1095–1112.)

24 M. E. J. Newman: The structure and function of complex networks, SIAM Review, Vol. 45, 2003, pp. 167–256.

25 T. Shigezumi, Y. Uno, and O. Watanabe: A new model for a scale-free hierarchical structure of isolated cliques, Journal of Graph Algorithms and Applications, Vol. 15, No. 5, 2011, pp. 661–682.

26 D.J. Watts and S.H. Strogatz: Collective dynamics of 'small-world' networks, Nature, Vol. 393, 1998, pp. 440–442.

27 Z. Zhang, L. Rong, and F. Comellas: High-dimensional random apollonian networks, Physica A: Statistical Mechanics and its Applications, Vol. 364, 2006, 610–618.

Explicit Correlation Amplifiers for Finding Outlier Correlations in Deterministic Subquadratic Time[*][†]

Matti Karppa[1], Petteri Kaski[‡][2], Jukka Kohonen[3], and Padraig Ó Catháin[4]

1 **Helsinki Institute for Information Technology HIIT and Department of Computer Science, Aalto University, Helsinki, Finland**
2 **Helsinki Institute for Information Technology HIIT and Department of Computer Science, Aalto University, Helsinki, Finland**
3 **Helsinki Institute for Information Technology HIIT and Department of Computer Science, Aalto University, Helsinki, Finland**
4 **Helsinki Institute for Information Technology HIIT and Department of Computer Science, Aalto University, Helsinki, Finland**

Abstract

We derandomize G. Valiant's [*J. ACM* 62 (2015) Art. 13] subquadratic-time algorithm for finding outlier correlations in binary data. Our derandomized algorithm gives deterministic subquadratic scaling essentially for the same parameter range as Valiant's randomized algorithm, but the precise constants we save over quadratic scaling are more modest. Our main technical tool for derandomization is an explicit family of *correlation amplifiers* built via a family of zigzag-product expanders in Reingold, Vadhan, and Wigderson [*Ann. of Math.* 155 (2002) 157–187]. We say that a function $f : \{-1,1\}^d \to \{-1,1\}^D$ is a *correlation amplifier* with threshold $0 \le \tau \le 1$, error $\gamma \ge 1$, and strength p an even positive integer if for all pairs of vectors $x, y \in \{-1,1\}^d$ it holds that (i) $|\langle x,y\rangle| < \tau d$ implies $|\langle f(x), f(y)\rangle| \le (\tau\gamma)^p D$; and (ii) $|\langle x,y\rangle| \ge \tau d$ implies $\bigl(\frac{\langle x,y\rangle}{\gamma d}\bigr)^p D \le \langle f(x), f(y)\rangle \le \bigl(\frac{\gamma\langle x,y\rangle}{d}\bigr)^p D$.

1998 ACM Subject Classification F.2.1 Numerical Algorithms and Problems

Keywords and phrases correlation, derandomization, outlier, similarity search, expander graph

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.52

1 Introduction

Identifying weak correlations in data. We consider the task of finding outlier-correlated pairs from large collections of weakly correlated binary vectors in $\{-1,1\}^d$. In more precise terms, we are interested in the following computational problem.

▶ **Problem 1** (Outlier correlations). *Suppose we are given as input two sets $X, Y \subseteq \{-1,1\}^d$ with $|X| = |Y| = n$ and two thresholds, the* outlier *threshold $\rho > 0$ and the* background *threshold $\tau < \rho$. Our task is to output all* outlier *pairs $(x,y) \in X \times Y$ with $|\langle x,y\rangle| \ge \rho d$ subject to the assumption that at most q of the pairs $(x,y) \in X \times Y$ satisfy $|\langle x,y\rangle| > \tau d$.*

[*] We use the dagger-symbol ("†") to indicate a result whose proof for reasons of space has been relegated to the full version available at `http://arxiv.org/abs/1606.05608`.

[†] The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement 338077 "Theory and Practice of Advanced Search and Enumeration" (M.K., P.K., J.K.) and from the Academy of Finland, Grants 276031, 282938, 283262, and 283437 (P.Ó C.).

[‡] Work done in part while the second author was visiting the Simons Institute for the Theory of Computing.

▶ Remark. This setting of binary vectors and (Pearson) correlation is directly motivated, among others, by the connection to Hamming distance. Indeed, for two vectors $x, y \in \{-1, 1\}^d$ we have $d - 2D_H(x, y) = \langle x, y\rangle$, where $D_H(x, y) = |\{u = 1, 2, \ldots, d : x(u) \neq y(u)\}|$ is the *Hamming distance* between x and y.

A naïve way to solve Problem 1 is to compute all the n^2 inner products $\langle x, y\rangle$ for $(x, y) \in X \times Y$ and filter out everything but the outliers. Our interest is in algorithms that scale *subquadratically* in n when both d and q are bounded from above by slowly growing functions of n. That is, we seek running times of the form $O(n^{2-\epsilon})$ for a constant $\epsilon > 0$. Furthermore, we seek to do this without *a priori* knowledge of q.

Running times of the form $O(n^{2-c\rho})$ for a constant $c > 0$ are immediately obtainable using techniques such as the seminal *locality-sensitive hashing* of Indyk and Motwani [17] and its variants[1]; however, such algorithms converge to quadratic running time in n unless ρ is bounded from below by a positive constant. Our interest is in algorithms that avoid such a "curse of weak outliers" and run in subquadratic time essentially *independently of the magnitude of ρ, provided that ρ is sufficiently separated from τ*. Such ability to identify weak outliers from large amounts of data is useful, among others, in machine learning from noisy data.

One strategy to circumvent the curse of weak outliers is to pursue the following intuition: (i) partition the input vectors into *buckets* of at most s vectors each, (ii) aggregate each bucket into a single vector by taking the vector sum, and (iii) compute the inner products between the $\lceil n/s\rceil \times \lceil n/s\rceil$ pairs of aggregate vectors. With sufficient separation between τ and ρ, at most q of these inner products between aggregates will be large, and every outlier pair is discoverable among the at most $s \times s$ input pairs that correspond to each large inner product of aggregates. Furthermore, a strategy of this form is oblivious to q until we actually start searching inside the buckets, which enables adjusting ρ and τ based on the number of large aggregate inner products.

Randomized amplification. Such bucketing strategies have been studied before with the help of randomization. In 2012, G. Valiant [33] presented a breakthrough algorithm that, before bucketing, replaces each input vector with a randomly subsampled[2] version of its p^{th} Kronecker power. Because of the tensor-power identity

$$\langle x^{\otimes p}, y^{\otimes p}\rangle = \langle x, y\rangle^p\,, \tag{1}$$

the ratio between outlier and background correlations gets *amplified* to essentially its p^{th} power, assuming that the sample is large enough so that sufficient concentration bounds hold with high probability. This amplification makes the outliers stand out from the background even after bucketing, which enables detection in subquadratic time using fast matrix multiplication.

A subset of the present authors [20] further improved on Valiant's algorithm by a modified sampling scheme that *simultaneously* amplifies and aggregates the input by further use of fast matrix multiplication. With this improvement, Problem 1 can be solved in subquadratic time if the logarithmic ratio $\log_\tau \rho = (\log \rho)/(\log \tau)$ is bounded from above by a constant less than 1. Also this improved algorithm relies on randomization.

[1] We postpone a more detailed discussion of related work and applications to the end of this section.
[2] Random sampling is used to reduce the dimension because the full d^p-dimensional Kronecker power is too large to be manipulated explicitly to yield subquadratic running times.

Explicit amplification. In this paper we seek *deterministic* subquadratic algorithms. As with the earlier randomized algorithms, we seek to map the d-dimensional input vectors to a higher dimension D so that inner products are sufficiently amplified in the process. Towards this end, we are interested in explicit functions $f : \{-1,1\}^d \to \{-1,1\}^D$ that approximate the tensor-power identity (1).

▶ **Definition 2** (Correlation amplifier). Let d, D and p be positive integers, with p even, and let $0 \le \tau \le 1$ and $\gamma \ge 1$. A function $f : \{-1,1\}^d \to \{-1,1\}^D$ is a *correlation amplifier* with parameters (d, D, p, τ, γ) if for all pairs of vectors $x, y \in \{-1,1\}^d$ we have

$$\text{if } \big|\langle x,y\rangle\big| < \tau d, \text{ then } \big|\langle f(x), f(y)\rangle\big| \le (\tau\gamma)^p D\,; \text{ and} \tag{2}$$

$$\text{if } \big|\langle x,y\rangle\big| \ge \tau d, \text{ then } \left(\tfrac{\langle x,y\rangle}{\gamma d}\right)^p D \le \langle f(x), f(y)\rangle \le \left(\tfrac{\gamma\langle x,y\rangle}{d}\right)^p D\,. \tag{3}$$

▶ Remark. A correlation amplifier f guarantees by (2) that correlations below τ in absolute value stay bounded; and by (3) that correlations at least τ in absolute value *become positive* and are governed by the two-sided approximation with multiplicative error $\gamma \ge 1$. In particular, (3) implies that correlations at least τ cannot mask outliers under bucketing because all such correlations get positive sign under amplification.

It is immediate that correlation amplifiers exist. For example, take $f(x) = x^{\otimes p}$, with p even, to obtain a correlation amplifier with $D = d^p$, $\tau = 0$, and $\gamma = 1$ by (1). For our present purposes, however, we seek correlation amplifiers with D substantially smaller than d^p. Furthermore, we seek constructions that are *explicit* in the strong[3] form that there exists a deterministic algorithm that computes any individual coordinate of $f(x)$ in time $\mathrm{poly}(\log D, p)$ by accessing $\mathrm{poly}(p)$ coordinates of a given $x \in \{-1,1\}^d$. In what follows explicitness always refers to this strong form.

Our results. The main result of this paper is that sufficiently powerful explicit amplifiers exist to find outlier correlations in deterministic subquadratic time.

▶ **Theorem 3** (Explicit amplifier family). *There exists an explicit correlation amplifier $f : \{-1,1\}^d \to \{-1,1\}^{2^K}$ with parameters $(d, 2^K, 2^\ell, \tau, \gamma)$ whenever $0 < \tau < 1$, $\gamma > 1$, and d, K, ℓ are positive integers with*

$$2^K \ge d\left(2^{10}\left(1-\gamma^{-1/2}\right)^{-1}\right)^{20\ell+1}\left(\frac{\gamma}{\tau}\right)^{60\cdot 2^\ell}. \tag{4}$$

As a corollary we obtain a deterministic algorithm for finding outlier correlations in subquadratic time using bucketing and fast matrix multiplication. Let us write α for the limiting exponent of rectangular integer matrix multiplication. That is, for all constants $\eta > 0$ there exists an algorithm that multiplies an $m \times \lfloor m^\alpha \rfloor$ integer matrix with an $\lfloor m^\alpha \rfloor \times m$ integer matrix in $O(m^{2+\eta})$ arithmetic operations. In particular, it is known that $0.3 < \alpha \le 1$ [22].

▶ **Theorem 4** (Deterministic subquadratic algorithm for outlier correlations). *For any constants $0 < \epsilon < 1$, $0 < \tau_{\max} < 1$, $0 < \delta < \alpha$, and $C > 60$, there exists a deterministic algorithm that solves a given instance of Problem 1 in time*

$$O\left(n^{2-\frac{0.99\epsilon(\alpha-\delta)}{4C+1}} + qn^{\delta+\frac{1.99\epsilon(\alpha-\delta)}{4C+1}}\right) \tag{5}$$

[3] In comparison, a weaker form of explicitness could require, for example, that there exists a deterministic algorithm that computes the entire vector $f(x)$ from a given x in time $D \cdot \mathrm{poly}(\log D, p)$.

assuming that the parameters n, d, ρ, τ satisfy the following three constraints
1. $d \le n^{\delta}$,
2. $n^{-\Theta(1)} \le \tau \le \tau_{\max}$, *and*
3. $\log_\tau \rho \le 1 - \epsilon$.

▶ Remark. Observe in particular that (5) is subquadratic regardless of the magnitude of ρ provided that the separation between ρ and τ via $\log_\tau \rho \le 1 - \epsilon$ holds.[4] The constants in (4) and (5) have not been optimized beyond our desired goal of obtaining deterministic subquadratic running time when d and q are bounded by slowly growing functions of n. In particular, (5) gives substantially worse subquadratic running times compared with the existing randomized strategies [20, 33]. The algorithm in Theorem 4 needs no *a priori* knowledge of q and is oblivious to q until it starts searching inside the buckets.

Overview and discussion of techniques. A straightforward application of the probabilistic method establishes that low-dimensional correlation amplifiers can be obtained by subsampling uniformly at random the dimensions of the tensor power $x^{\otimes p}$ as long as the sample size D is large enough.

▶ **Lemma 5** (Existence †). *There exists a correlation amplifier $f : \{-1,1\}^d \to \{-1,1\}^D$ with parameters (d, D, p, τ, γ) whenever $0 < \tau < 1$, $\gamma > 1$, and d, p, D are positive integers satisfying*

$$D \ge 3d\left(\gamma^p - 1\right)^{-2}\left(\frac{\gamma}{\tau}\right)^{2p} . \tag{6}$$

Thus, in essence our Theorem 3 amounts to *derandomizing* such a subsampling strategy by presenting an explicit sample that is, up to the error bounds (2) and (3), indistinguishable from the "perfect" amplifier $x \mapsto x^{\otimes p}$ under taking of inner products.

The construction underlying Theorem 3 amounts to an ℓ-fold composition of explicit *squaring* amplifiers ($p = 2$) with increasingly strong control on the error (γ) and the interval of amplification ($[\tau, 1]$) at each successive composition. Towards this end, we require a flexible explicit construction of squaring amplifiers with strong control on the error and the interval. We obtain such a construction from an explicit family of expander graphs (Lemma 9) obtainable from the explicit zigzag-product constructions of Reingold, Vadhan, and Wigderson [31]. In particular, the key to controlling the error and the interval is that the expander family gives *Ramanujan-like*[5] concentration $\lambda/\Delta \le 16\Delta^{-1/4}$ of the normalized second eigenvalue λ/Δ by increasing the degree Δ. In essence, since we are working with $\{-1,1\}$-valued vectors, by increasing the degree we can use the Expander Mixing Lemma (Lemma 8) and the Ramanujan-like concentration to control (Lemma 11) how well the restriction x^G to the edges of an expander graph G approximates the full tensor square $x^{\otimes 2}$ under taking of inner products.

Our construction has been motivated by the paradigm of *gradually increasing independence* [6, 11, 12, 18] in the design of pseudorandom generators. Indeed, we obtain the final amplifier gradually by successive squarings, taking care that the degree Δ_i of the expander

[4] The technical constraint $n^{-\Theta(1)} \le \tau$ only affects inputs where the dimension d grows essentially as a root function of n since $\tau \ge 1/d$. The constant subsumed by $\Theta(1)$ depends on the chosen constants $\epsilon, \tau_{\max}, \delta, C$ but not on the other parameters.

[5] Actual *Ramanujan graphs* (see [15, 23]) would give somewhat stronger concentration $\lambda/\Delta = O(\Delta^{-1/2})$ and hence improved constants in (4). However, we are not aware of a sufficiently fine-grained family of explicit Ramanujan graphs to comfortably support successive squaring.

that we apply in each squaring $i = 0, 1, \ldots, \ell - 1$ increases with a similar squaring schedule given by (10) and (12) to simultaneously control the error and the interval, and to bound the output dimension roughly by the square of the degree of the last expander in the sequence.[6] The analogy with pseudorandom generators can in fact be pushed somewhat further. Namely, a correlation amplifier can be roughly seen as a pseudorandom generator that by (3) seeks to fool a "truncated family of uniform combinatorial rectangles" with further control requested by (2) below the truncation threshold τ.[7] Our goal to obtain a small output dimension D roughly corresponds to optimizing the seed length of a pseudorandom generator.

While our explicit construction (4) does not reach the exact output dimension obtainable by Lemma 5, it should be observed that in our parameter range of interest (with $\gamma > 1$ a constant and $0 < \tau \leq \tau_{\max}$ for a constant $0 < \tau_{\max} < 1$), both (4) and (6) are of the form $D \geq d\tau^{-\Theta(p)}$; only the constants hidden by the asymptotic notation differ between the explicit and nonconstructive bounds. Moreover, using results of Alon [3] we show a *lower bound* on the output dimension D of any correlation amplifier: namely, that $D \geq d\tau^{-\Theta(p)}$ if $6p\tau^{-2}d^{-1}\log\frac{1}{\gamma\tau}$ is bounded from above by a constant strictly less than 1 (†). Thus, viewed as a pseudorandom generator with "seed length" $\log D$, Theorem 3 essentially does not admit improvement except possibly at the multiplicative constants.

Related work and applications. Problem 1 is a basic problem in data analysis and machine learning admitting many extensions, restrictions, and variants. A large body of work exists studying *approximate near neighbour search* via techniques such as locality-sensitive hashing (e.g. [4, 5, 17, 10, 26, 27]), with recent work aimed at derandomization (see Pagh [28] and Pham and Pagh [30]) and resource tradeoffs (see Kapralov [19]) in particular. However, these techniques enable subquadratic scaling in n only when ρ is bounded from below by a positive constant, whereas the algorithm in Theorem 4 remains subquadratic even in the case of weak outliers when ρ tends to zero with increasing n, as long as ρ and τ are separated. Ahle, Pagh, Razenshteyn, and Silvestri [1] show that subquadratic scaling in n is not possible for $\log_\tau \rho = 1 - o(1/\sqrt{\log n})$ unless both the Orthogonal Vectors Conjecture and the Strong Exponential Time Hypothesis [16] fail.

In small dimensions, Alman and Williams [2] present a randomized algorithm that finds *exact* Hamming-near neighbours in a batch-query setting analogous to Problem 1 in subquadratic time in n when the dimension is constrained to $d = O(\log n)$. Recently, Chan and Williams [7] show how to derandomize related algorithm designs, but the probabilistic polynomials for symmetric Boolean functions used in [2] to our knowledge have not yet been derandomized.

[6] The term "gradual" is of course not particularly descriptive since growth under successive squaring amounts to *doubly* exponential growth in the number of squarings. Yet such growth *can* be seen as gradual and controlled since we obtain strong amplification compared with the final output dimension precisely because the first $\ell - 1$ squarings "come for free" since $\Delta_0\Delta_1\cdots\Delta_{\ell-2}$ is (up to low-order multiplicative terms) no more than $\Delta_{\ell-1}^2$, essentially because we are taking the sum of powers of 2 in the exponent.

[7] To see the rough analogy, let $z \in \{-1, 1\}^d$ be the Hadamard product of the vectors $x, y \in \{-1, 1\}^d$ and observe that (3) seeks to approximate (with multiplicative error) the expectation of a uniform random entry in the d^p-length Kronecker power $z^{\otimes p}$ by instead taking the expectation over an explicit D-dimensional sample given by f. The Kronecker power $z^{\otimes p}$ is a uniform special case (with $z = z_1 = z_2 = \cdots = z_p$) of a "combinatorial rectangle" formed by a Kronecker product $z_1 \otimes z_2 \otimes \cdots \otimes z_p$, and truncation means that we only seek approximation in cases where $|\sum_{u=1}^d z(u)| \geq \tau d$, and accordingly want constructions that take this truncation into account—that is, we do not seek to fool all combinatorial rectangles and accordingly want stronger control on the dimension D (that is, the "seed length" $\log D$). For a review of the state of the art in pseudorandom generators we refer to Gopalan, Kane, and Meka [11] and Kothari and Meka [21].

One special case of Problem 1 is the problem of learning a weight 2 parity function in the presence of noise, or *the light bulb problem.*

▶ **Problem 6** (Light bulb problem, L. Valiant [34]). *Suppose we are given as input a parameter $0 < \rho < 1$ and a set of n vectors in $\{-1, 1\}^d$ such that one planted pair of vectors has inner product at least ρd in absolute value, and all other $n - 2$ vectors are chosen independently and uniformly at random. Our task is to find the planted pair among the n vectors.*

▶ Remark. From e.g. the Hoeffding bound (20) it follows that there exists a constant c such that when $d \geq c\rho^{-2} \log n$ the planted pair is with high probability (as n increases) the unique pair in the input with the maximum absolute correlation.

For a problem whose instances are drawn from a random ensemble, we say that an algorithm solves *almost all* instances of the problem if the probability of drawing an instance where the algorithm fails tends to zero as n increases.

Paturi, Rajasekaran, and Reif [29], Dubiner [8], and May and Ozerov [24] present randomized algorithms that can be used to solve almost all instances of the light bulb problem in subquadratic time if we assume that ρ is bounded from below by a positive constant; if ρ tends to zero these algorithms converge to quadratic running time in n.

G. Valiant [33] showed that a randomized algorithm can identify the planted correlation in subquadratic time on almost all inputs even when ρ tends to zero as n increases. As a corollary of Theorem 4, we can derandomize Valiant's design and still retain subquadratic running time (but with a worse constant) for almost all inputs, except for extremely weak planted correlations with $\rho \leq n^{-\Omega(1)}$ that our amplifier is not in general able to amplify with sufficiently low output dimension to enable an overall subquadratic running time.

▶ **Corollary 7** (Deterministic subquadratic algorithm for the light bulb problem). *For any constants $0 < \delta < \alpha$, $C > 60$, $0 < \rho_{\max} < 1$, and $\kappa > 1$, there exists a deterministic algorithm that solves almost all instances of Problem 6 in time*

$$O\left(n^{2-\frac{0.99(1-1/\kappa)(\alpha-\delta)}{4C+1}}\right)$$

assuming the parameters n, d, ρ satisfy the two constraints

1. $5\rho^{-2\kappa} \log n \leq d \leq n^{\delta}$ *and*
2. $n^{-\Theta(1)} \leq \rho \leq \rho_{\max}$.[8]

Corollary 7 extends to parity functions of larger (constant) weight (cf. [13, 20, 33]), however, we omit the details from this conference abstract. Algorithms for learning parity functions enable extensions to further classes of Boolean functions such as sparse juntas and DNFs (cf. [9, 25, 33]).

Conventions and notation. All vectors in this paper are integer-valued. For a vector $x \in \mathbb{Z}^d$ we denote the entry $u = 1, 2, \ldots, d$ of x by $x(u)$. For two vectors $x, y \in \mathbb{Z}^d$ we write $\langle x, y\rangle = \sum_{u=1}^{d} x(u)y(u)$ for the inner product of x and y. We write log for the logarithm with base 2 and ln for the logarithm with base exp(1).

[8] The constant hidden by the $\Theta(1)$ notation depends on the constants $\delta, \alpha, C, \rho_{\max}$ but not on the other parameters. For details consult the proof.

2 Explicit amplifiers by approximate squaring

This section proves Theorem 3. We start with preliminaries on expanders, show an approximate squaring identity using expander mixing, and then rely on repeated approximate squaring for our main construction. The proof is completed by some routine preprocessing.

Preliminaries on expansion and mixing. We work with undirected graphs, possibly with self-loops and multiple edges. A graph G is Δ-*regular* if every vertex is incident to exactly Δ edges, with each self-loop (if present) counting as one edge. Suppose that G is Δ-regular with vertex set V, and let L be a set of Δ labels such that the Δ edge-ends incident to each vertex have been labeled with unique labels from L. The *rotation map* $\mathrm{Rot}_G : V \times L \to V \times L$ is the bijection such that for all $u \in V$ and $i \in L$ we have $\mathrm{Rot}_G(u,i) = (v,j)$ if the edge incident to vertex u and labeled with i at u leads to the vertex v and has the label j at v.

For $S, T \subseteq V(G)$, let us write $E(S,T)$ for the set of edges of G with one end in S and the other end in T. Suppose that G has D vertices and let $\lambda_1, \lambda_2, \ldots, \lambda_D$ be the eigenvalues of the adjacency matrix of G with $|\lambda_1| \geq |\lambda_2| \geq \cdots \geq |\lambda_D|$. Let us say that a graph G is a (D,Δ,λ)-*graph* if G has D vertices, G is Δ-regular, and $|\lambda_2| \leq \lambda$. For an excellent survey on expansion and expander graphs, we refer to Hoory, Linial, and Wigderson [15].

▶ **Lemma 8** (Expander mixing lemma, [15, Lemma 2.5]). *For all $S, T \subseteq V(G)$ we have*

$$\left| |E(S,T)| - \frac{\Delta |S||T|}{D} \right| \leq \lambda \sqrt{|S||T|}\,.$$

We work with the following family of graphs obtained from the zig-zag product of Reingold, Vadhan, and Wigderson [31]. In particular Lemma 9 gives us $\lambda/\Delta \leq 16\Delta^{-1/4}$, which will enable us to control relative inner products by increasing Δ.

▶ **Lemma 9.** *For all integers $t \geq 1$ and $b \geq 10$ there exists a $(2^{16bt}, 2^{4b}, 16 \cdot 2^{3b})$-graph whose rotation map can be evaluated in time* $\mathrm{poly}(b,t)$.[9]

Proof. See Appendix A. ◀

Main construction. The main objective of this section is to prove the following lemma, which we will then augment to Theorem 3 by routine preprocessing of the input dimension.

▶ **Lemma 10** (Repeated approximate squaring). *There exists an explicit correlation amplifier $\hat{f} : \{-1,1\}^{2^k} \to \{-1,1\}^{2^K}$ with parameters $(2^k, 2^K, 2^\ell, \tau_0, \gamma_0)$ whenever $0 < \tau_0 < 1$, $\gamma_0 > 1$, and k, K, ℓ are positive integers with*

$$2^K \geq 2^k \left(2^{10} \left(1 - \gamma_0^{-1}\right)^{-1} \right)^{20\ell} \left(\frac{\gamma_0}{\tau_0} \right)^{40 \cdot 2^\ell - 20}. \tag{7}$$

Approximate squaring via expanders. For a vector $x \in \{-1,1\}^D$, let us write $x^{\otimes 2} \in \{-1,1\}^{D^2}$ for the Kronecker product of x with itself. Our construction for correlation amplifiers will rely on approximating the *squaring identity*

$$\langle x^{\otimes 2}, y^{\otimes 2} \rangle = \langle x, y \rangle^2\,,$$

[9] *Caveat.* Reingold, Vadhan, and Wigderson [31] work with eigenvalues of the *normalized* adjacency matrix (with $|\lambda_1| = 1$) whereas we follow Hoory, Linial, and Wigderson [15] and work with unnormalized adjacency matrices (with $|\lambda_1| = \Delta$) in the manuscript proper. Appendix A works with normalized adjacency matrices for compatibility with Reingold, Vadhan, and Wigderson [31].

for vectors in $\{-1,1\}^D$. In more precise terms, let G be a (D,Δ,λ)-graph and let $x^G \in \{-1,1\}^{\Delta D}$ be a vector that contains each coordinate $x(u)x(v)$ of $x^{\otimes 2}$ with $(u,v) \in V(G) \times V(G)$ exactly once for each edge of G that joins the vertex u to the vertex v. Equivalently, let $\mathrm{Rot}_G : V \times L \to V \times L$ be a rotation map for G, and define x^G for all $u \in V$ and all $i \in L$ by $x^G(u,i) = x(u)x(v)$ where $v \in V$ is given by $\mathrm{Rot}_G(u,i) = (v,j)$. In particular, x^G has exactly ΔD coordinates.

▶ **Lemma 11** (Approximate squaring). *For all $x, y \in \{-1,1\}^D$ we have*

$$\left| \langle x^G, y^G \rangle - \frac{\Delta}{D} \langle x^{\otimes 2}, y^{\otimes 2} \rangle \right| \le 2\lambda D\,.$$

Proof. Let $S = \{u \in V(G) : x(u) = y(u)\}$ and let us write $\bar{S} = V(G) \setminus S$. Since x, y are $\{-1,1\}$-valued, we have

$$\langle x^G, y^G \rangle = |E(S,S)| + |E(\bar{S},\bar{S})| - |E(S,\bar{S})| - |E(\bar{S},S)|\,.$$

Observing that

$$|S|^2 + |\bar{S}|^2 - |S||\bar{S}| - |\bar{S}||S| = \big(2|S| - D\big)^2 = \langle x, y \rangle^2 = \langle x^{\otimes 2}, y^{\otimes 2} \rangle$$

and applying Lemma 8 four times, we have

$$\left| \langle x^G, y^G \rangle - \frac{\Delta}{D} \langle x^{\otimes 2}, y^{\otimes 2} \rangle \right| \le \lambda\big(D + 2\sqrt{|S|(D - |S|)}\big) \le 2\lambda D\,.$$ ◀

The amplifier function. We now construct an amplifier function $\hat{f}$ that uses ℓ approximate squarings, $\ell \ge 1$, with the graphs drawn from the graph family in Lemma 9. Accordingly, we assume that all vectors have lengths that are positive integer powers of 2.

The input $x = \tilde{x}_0 \in \{-1,1\}^{d_0}$ to the amplifier has dimension $d_0 = 2^k$ for a positive integer k. For $i = 0, 1, \ldots, \ell - 1$, suppose we have the vector $\tilde{x}_i \in \{-1,1\}^{d_i}$. Let b_i be a positive integer whose value will be fixed later. Let t_i be the unique positive integer with

$$d_i \le D_i = 2^{16 b_i t_i} < 2^{16 b_i} d_i\,.$$

Note in particular that d_i divides D_i since d_i is a power of 2. Let G_i be a $(2^{16 b_i t_i}, 2^{4 b_i}, 16 \cdot 2^{3 b_i})$-graph from Lemma 9. Take D_i / d_i copies of $\tilde{x}_i$ to obtain the vector $x_i \in \{-1,1\}^{D_i}$. Let $\tilde{x}_{i+1} = x_i^{G_i} \in \{-1,1\}^{d_{i+1}}$ with $d_{i+1} = \Delta_i D_i$ and $\Delta_i = 2^{4 b_i}$. The amplifier outputs $\hat{f}(x) = \tilde{x}_\ell$ with $\tilde{x}_\ell \in \{-1,1\}^{d_\ell}$.

Since the graph family in Lemma 9 admits rotation maps that can be computed in time $\mathrm{poly}(b,t)$, we observe that $\hat{f}$ is explicit. Indeed, from the construction it is immediate that to compute any single coordinate of $\hat{f}(x)$ it suffices to (i) perform in total $2^{\ell-1-i}$ evaluations of the rotation map of the graph G_i for each $i = 0, 1, \ldots, \ell - 1$, and (ii) access at most 2^ℓ coordinates of x. Since $b_i t_i = O(\log d_\ell)$ for all $i = 0, 1, \ldots, \ell - 1$, we have that we can compute any coordinate of $\hat{f}(x)$ in time $\mathrm{poly}(\log d_\ell, 2^\ell)$ and accessing at most 2^ℓ coordinates of x.

Parameterization and analysis. Fix $\tau_0 > 0$ and $\gamma_0 > 1$. To parameterize the amplifier (that is, it remains to fix the values b_i), let us track a pair of vectors as it proceeds through the ℓ approximate squarings for $i = 0, 1, \ldots, \ell - 1$.

We start by observing that copying preserves *relative* inner products. That is, for any pair of vectors $\tilde{x}_i, \tilde{y}_i \in \{-1,1\}^{d_i}$ we have $\langle \tilde{x}_i, \tilde{y}_i \rangle = \nu_i d_i$ if and only if $\langle x_i, y_i \rangle = \nu_i D_i$ for $0 \le \nu_i \le 1$.

An easy manipulation of Lemma 11 using the parameters in Lemma 9 gives us additive control over an approximate squaring via

$$\nu_i^2 - 32\Delta_i^{-1/4} \le \nu_{i+1} \le \nu_i^2 + 32\Delta_i^{-1/4} . \tag{8}$$

For all inner products that are in absolute value above a threshold, we want to turn this additive control into multiplicative control via

$$\nu_i^2\gamma_0^{-1} \le \nu_{i+1} \le \nu_i^2\gamma_0 . \tag{9}$$

Let us insist this multiplicative control holds whenever $|\nu_i| \ge \tau_i$ for the threshold parameter τ_i defined for all $i = 0, 1, \ldots, \ell - 1$ by

$$\tau_{i+1} = \gamma_0^{-1}\tau_i^2 . \tag{10}$$

Enforcing (9) via (8) at the threshold, let us assume that

$$\tau_i^2\gamma_0^{-1} \le \tau_i^2 - 32\Delta_i^{-1/4} . \tag{11}$$

The next lemma confirms that assuming (11) gives two-sided control of inner products which is retained to the next approximate squaring. The following lemma shows that small inner products remain small.

▶ **Lemma 12** (†). *If $\tau_i \le |\nu_i|$, then $\nu_i^2\gamma_0^{-1} \le \nu_{i+1} \le \nu_i^2\gamma_0$ and $\tau_{i+1} \le \nu_{i+1}$.*

▶ **Lemma 13** (†). *If $|\nu_i| < \tau_i$, then $|\nu_{i+1}| \le \tau_i^2\gamma_0$.*

Let us now make sure that (11) holds. Solving for Δ_i in (11), we have

$$\Delta_i \ge \left(32(1-\gamma_0^{-1})^{-1}\tau_i^{-2}\right)^4 . \tag{12}$$

In particular, we can make sure that (12) and hence (11) holds by simply choosing a large enough Δ_i (that is, a large enough b_i).

Before proceeding with the precise choice of b_i for $i = 0, 1, \ldots, \ell-1$, let us analyze the input–output relationship of the amplifier $\hat{f}$ using Lemma 12 and Lemma 13. Let $x, y \in \{-1, 1\}^{d_0}$ be two vectors given as input with $\langle x, y\rangle = \nu_0 d_0$. The outputs $\hat{f}(x), \hat{f}(y) \in \{-1, 1\}^{d_\ell}$ then satisfy $\langle \hat{f}(x), \hat{f}(y)\rangle = \nu_\ell d_\ell$, where the following two lemmas control ν_ℓ via ν_0.

▶ **Lemma 14** (†). *If $|\nu_0| \ge \tau_0$, then $\nu_0^{2^\ell}\gamma_0^{-2^\ell+1} \le \nu_\ell \le \nu_0^{2^\ell}\gamma_0^{2^\ell-1}$.*

▶ **Lemma 15** (†). *If $|\nu_0| < \tau_0$, then $|\nu_\ell| \le \tau_0^{2^\ell}\gamma_0^{2^\ell-1}$.*

Since $\gamma_0 > 1$, from Lemma 14 and Lemma 15 it now follows that $\hat{f}$ meets the required amplification constraints (2) and (3) with $p = 2^\ell$, $\tau = \tau_0$, and $\gamma = \gamma_0$.

Let us now complete the parameterization and derive an upper bound for d_ℓ. For each $i = 0, 1, \ldots, \ell-1$, take b_i to be the smallest nonnegative integer so that $b_i \ge 10$ and $\Delta_i = 2^{4b_i}$ satisfies (12). Since $D_i \le 2^{16b_i}d_i = \Delta_i^4 d_i$, we have $d_{i+1} = \Delta_i D_i \le \Delta_i^5 d_i$, and hence

$$d_\ell \le (\Delta_{\ell-1}\Delta_{\ell-2}\cdots\Delta_0)^5 d_0 .$$

Recall that $d_0 = 2^k$. From (12) we have that

$$\Delta_i = 2^{4b_i} \le \max\left(2^{40}, 2^4\left(32(1-\gamma_0^{-1})^{-1}\tau_i^{-2}\right)^4\right) \le \left(2^{10}(1-\gamma_0^{-1})^{-1}\tau_i^{-2}\right)^4 .$$

Since $\tau_i = \tau_0^{2^i}\gamma_0^{-2^i+1}$ by (10), it follows that

$$d_\ell \le 2^k\left(2^{10}\left(1-\gamma_0^{-1}\right)^{-1}\right)^{20\ell}\left(\frac{\gamma_0}{\tau_0}\right)^{20(2^{\ell+1}-1)} .$$

Repeatedly taking two copies of the output as necessary, for all 2^K with $2^K \ge d_\ell$ we obtain a correlation amplifier with parameters $(2^k, 2^K, 2^\ell, \tau_0, \gamma_0)$. This completes the proof of Lemma 10. ◀

Copy-and-truncate preprocessing of the input dimension. We still want to remove the assumption from Lemma 10 that the input dimension is a positive integer power of 2. The following copy-and-truncate preprocessing will be sufficient towards this end.

Let $x \in \{-1,1\}^d$ and let k be a positive integer. Define the vector $\hat{x} \in \{-1,1\}^{2^k}$ by concatenating $\lceil 2^k/d \rceil$ copies of x one after another, and truncating the result to the 2^k first coordinates to obtain $\hat{x}$.

Let us study how the map $x \mapsto \hat{x}$ operates on a pair of vectors $x, y \in \{-1,1\}^d$. For notational compactness, let us work with relative inner products $\nu, \hat{\nu}$ with $\langle x, y\rangle = \nu d$ and $\langle \hat{x}, \hat{y}\rangle = \hat{\nu} 2^k$.

▶ **Lemma 16** (†). *For any $0 < \tau_0 < 1$, $\gamma_0 > 1$, and $2^k \geq 2d\tau_0^{-1}(1-\gamma_0^{-1})^{-1}$ we have that*
1. *$|\nu| < \tau_0$ implies $|\hat{\nu}| \leq \gamma_0\tau_0$,*
2. *$|\nu| \geq \tau_0$ implies $\gamma_0^{-1}\nu \leq |\hat{\nu}| \leq \gamma_0\nu$.*

Completing the proof of Theorem 3. Let d, K, ℓ, τ, γ be parameters meeting the constraints in Theorem 3, in particular the constraint (4). To construct a required amplifier f, we preprocess each input vector x with copy-and-truncate, obtaining a vector $\hat{x}$ of length 2^k. We then then apply an amplifier $\hat{f} : \{-1,1\}^{2^k} \to \{-1,1\}^{2^K}$ given by Lemma 10. In symbols, we define $f : \{-1,1\}^d \to \{-1,1\}^{2^K}$ for all $x \in \{-1,1\}^d$ by $f(x) = \hat{f}(\hat{x})$. It is immediate from Lemma 10 and Lemma 16 that the resulting composition is explicit.

We begin by relating the given parameters of Theorem 3 to those of Lemma 10. Take $\gamma_0 = \gamma^{1/2}$, $\tau_0 = \tau\gamma^{-1}$, and select the minimal value of k so that the constraint in Lemma 16 is satisfied; that is 2^k is constrained as follows,

$$2d(1-\gamma^{-1/2})^{-1}\gamma\tau^{-1} \leq 2^k < 4d(1-\gamma^{-1/2})^{-1}\gamma\tau^{-1} .$$

Substituting this upper bound into the bound of Lemma 10, we get a lower bound for 2^K,

$$2^K \geq 2^{-8} d \left(2^{-10}(1-\gamma^{-1/2})^{-1}\right)^{20\ell+1} \frac{\gamma}{\tau} \left(\frac{\gamma^{60}}{\tau^{40}}\right)^{2^\ell} \frac{\tau^{20}}{\gamma^{30}} . \tag{13}$$

Observe that an integer 2^K satisfying (4) also satisfies (13). We have not attempted to optimise our construction, and prefer the the statement of Theorem 3 as it is reasonably clean and is sufficient to prove Theorem 4.

Let us study how the map $x \mapsto f(x)$ operates on a pair of vectors $x, y \in \{-1,1\}^d$. For notational compactness, again we work with relative inner products $\nu, \hat{\nu}, \phi$ with $\langle x, y\rangle = \nu d$, $\langle \hat{x}, \hat{y}\rangle = \hat{\nu}2^k$, and $\langle f(x), f(y)\rangle = \phi 2^K$. Observe that in the notation of the proof of Lemma 10, we have $\hat{\nu} = \nu_0$ and $\phi = \nu_\ell$.

▶ **Lemma 17** (†). *If $|\nu| < \tau$ then $|\phi| \leq (\gamma\tau)^{2^\ell}$.*

▶ **Lemma 18** (†). *If $|\nu| \geq \tau$ then $(\nu\gamma^{-1})^{2^\ell} \leq \phi \leq (\nu\gamma)^{2^\ell}$.*

Now, f satisfies (2) and (3) with $p = 2^\ell$ by Lemmas 17 and 18 respectively.
This completes the proof of Theorem 3. ◀

3 A deterministic algorithm for outlier correlations

This section proves Theorem 4. We start by describing the algorithm, then parameterize it and establish its correctness, and finally proceed to analyze the running time.

The algorithm. Fix the constants $\epsilon, \tau_{\max}, \delta, C$ as in Theorem 4. Based on these constants, fix the constants $0 < \sigma < 1$ and $\gamma > 1$. (We fix the precise values of σ and γ later during the analysis of the algorithm, and stress that σ, γ do not depend on the given input.)

Suppose we are given as input the parameters $0 < \tau < \rho < 1$ and $X, Y \subseteq \{-1,1\}^d$ with $|X| = |Y| = n$ so that the requirements in Theorem 4 hold. We work with a correlation amplifier $f : \{-1,1\}^d \to \{-1,1\}^D$ with parameters (d, D, p, τ, γ). (We fix the precise values of the parameters p and D later during the analysis of the algorithm so that f originates from Theorem 3.)

The algorithm proceeds as follows. First, apply f to each vector in X and Y to obtain the sets X_f and Y_f. Let $s = \lfloor n^\sigma \rfloor$. Second, partition the n vectors in both X_f and Y_f into $\lceil n/s \rceil$ buckets of size at most s each, and take the vector sum of the vectors in each bucket to obtain the sets $\tilde{X}_f, \tilde{Y}_f \subseteq \{-s, -s+1, \ldots, s-1, s\}^D$ with $|\tilde{X}_f|, |\tilde{Y}_f| \leq \lceil n/s \rceil$. Third, using fast rectangular matrix multiplication on $\tilde{X}_f$ and $\tilde{Y}_f$, compute the matrix Z whose entries are the inner products $\langle \tilde{x}, \tilde{y} \rangle$ for all $\tilde{x} \in \tilde{X}_f$ and all $\tilde{y} \in \tilde{Y}_f$. Fourth, iterate over the entries of Z, and whenever the *detection inequality*

$$\langle \tilde{x}, \tilde{y} \rangle > n^{2\sigma}(\tau\gamma)^p \tag{14}$$

holds, brute-force search for outliers among the at most $s \times s$ inner products in the corresponding pair of buckets. Output any outliers found.

Parameterization and correctness. Let us now parameterize the algorithm and establish its correctness. Since $\gamma > 1$ is a constant and assuming that p is large enough, by Theorem 3 we can select D to be the integer power of 2 with

$$\frac{1}{2} d \left(\frac{\gamma}{\tau}\right)^{Cp} < D \leq d \left(\frac{\gamma}{\tau}\right)^{Cp} .$$

Recall that we write α for the exponent of rectangular matrix multiplication. To apply fast rectangular matrix multiplication in the third step of the algorithm, we want

$$D \leq 2 \left(\frac{n}{s}\right)^{\alpha} , \tag{15}$$

so recalling that $d \leq n^\delta$ and $n^\sigma - 1 < s$, it suffices to require that

$$\left(\frac{\gamma}{\tau}\right)^{Cp} \leq n^{(1-\sigma)\alpha - \delta} .$$

Let us assume for the time being that $(1-\sigma)\alpha - \delta > 0$. (We will justify this assumption later when we choose a value for σ.) Let p be the unique positive-integer power of 2 such that

$$\frac{((1-\sigma)\alpha - \delta)\log n}{2C \log \frac{\gamma}{\tau}} < p \leq \frac{((1-\sigma)\alpha - \delta)\log n}{C \log \frac{\gamma}{\tau}} . \tag{16}$$

Observe that p exists and is positive for all large enough n since $\gamma > 1$ is a constant and $n^{-\Theta(1)} \leq \tau$ by our assumption.[10] By the detection inequality (14), we require each entry of Z to have value strictly greater than $n^{2\sigma}(\tau\gamma)^p$ if among the correspoding at most $s \times s$

[10] In particular, since $\sigma, \delta, \alpha, C, \gamma$ are constants, we can choose the constant hidden by the $\Theta(1)$ so that $1 \leq (((1-\sigma)\alpha - \delta)\log n)/(2C \log \frac{\gamma}{\tau})$.

inner products between the two buckets there is at least one inner product with absolute value at least ρd. Furthermore, we want (14) to hold only if among the at most $s \times s$ inner products between the two buckets there is at least one inner product with absolute value strictly greater than τd. Since f satisfies (2) and (3), and recalling that $s \le n^\sigma$, it suffices to require that

$$s^2\big(\tau\gamma\big)^p \le n^{2\sigma}\big(\tau\gamma\big)^p < \big(\rho\gamma^{-1}\big)^p - n^{2\sigma}\big(\tau\gamma\big)^p\,. \tag{17}$$

Rearranging the right-hand side of (17) and solving for p, we require that

$$p > \frac{1+2\sigma\log n}{\log\frac{\rho}{\tau\gamma^2}}\,. \tag{18}$$

From (16) and (18) we thus see that it suffices to have

$$p > \frac{((1-\sigma)\alpha-\delta)\log n}{2C\log\frac{\gamma}{\tau}} \ge \frac{1+2\sigma\log n}{\log\frac{\rho}{\tau\gamma^2}}\,,$$

or equivalently,

$$\frac{\log\frac{\rho}{\tau\gamma^2}}{\log\frac{\gamma}{\tau}} \ge \frac{\frac{2C}{\log n}+4C\sigma}{(1-\sigma)\alpha-\delta}\,. \tag{19}$$

Let us derive a lower bound for the left-hand side of (19). Fix the constant $\gamma > 1$ so that $\log\gamma = -\frac{\epsilon\log\tau_{\max}}{100000}$. By our assumptions we have $\tau \le \tau_{\max}$ and $1-\log_\tau\rho \ge \epsilon$, so we have the lower bound

$$\frac{\log\frac{\rho}{\tau\gamma^2}}{\log\frac{\gamma}{\tau}} = \frac{\log\rho-\log\tau-2\log\gamma}{\log\gamma-\log\tau} = \frac{1-\log_\tau\rho+\frac{2\log\gamma}{\log\tau}}{1-\frac{\log\gamma}{\log\tau}} \ge \frac{\epsilon+\frac{2\log\gamma}{\log\tau_{\max}}}{1-\frac{\log\gamma}{\log\tau_{\max}}} > 0.99\epsilon\,.$$

Thus, (19) holds for all large enough n when we require

$$0.99\epsilon \ge \frac{4C\sigma}{(1-\sigma)\alpha-\delta}\,.$$

Since $\alpha\epsilon < 1$, we have that (19) holds when we set

$$\sigma = \frac{0.99\epsilon(\alpha-\delta)}{4C+1} \le \frac{0.99\epsilon(\alpha-\delta)}{4C+0.99\alpha\epsilon}\,.$$

We also observe that $(1-\sigma)\alpha-\delta > 0$, or equivalently, $\sigma < (\alpha-\delta)/\alpha$ holds for our choice of σ. This completes the parameterization of the algorithm.

Running time. Let us now analyze the running time of the algorithm. The first and second steps run in time $\tilde{O}(nD)$ since $p = O(\log n)$ by (16) and f originates from Theorem 3 and hence is explicit. From (15) and $n^\sigma - 1 < s$, we have $nD \le 4n^{1+(1-\sigma)\alpha} \le 4n^{2-\sigma}$. Since (15) holds, the third step of the algorithm runs in time $O\big((n/s)^{2+\eta}\big)$ for any constant $\eta > 0$ that we are free to choose. Since $n/s \le 2n^{1-\sigma}$ for all large enough n, we can choose $\eta > 0$ so that $(2+\eta)(1-\sigma) \le 2-\sigma$. Thus, the first, second, and third steps together run in time $O(n^{2-\sigma})$. The fourth step runs in time $O(n^{2-\sigma} + qs^2d)$. Indeed, observe from (17) that the inequality (14) holds for at most q entries in Z. We have $qs^2d \le qn^{2\sigma+\delta}$, which completes the running time analysis and the proof of Theorem 4. ◀

4 Proof of Corollary 7

A useful variant of the Problem 1 asks for all outlier pairs of distinct vectors drawn from a *single* set $S \subseteq \{-1,1\}^d$ rather than two sets X, Y. We observe that the single-set variant reduces to $\lceil \log |S| \rceil$ instances of the two-set variant by numbering the vectors in S with binary numbers from 0 to $|S|-1$ and splitting S into two sets X_i, Y_i based on the value of the i^{th} bit for each $i = 0, 1, \ldots, \lceil \log |S| \rceil - 1$.

We will need the following bound due to Hoeffding which provides an exponentially small upper bound on the deviation of a sum of bounded independent random variables from its expectation.

▶ **Theorem 19** (Hoeffding [14, Theorem 2]). *Let $Z_1, Z_2, \ldots, Z_D$ be independent random variables satisfying $\ell_i \le Z_i \le u_i$ for all $1 \le i \le D$, and let $Z = \sum_{i=1}^{D} Z_i$. Then, for all $c > 0$, the following holds:*

$$\Pr\left(Z - \mathrm{E}[Z] \ge c\right) \le \exp\left(-\frac{2c^2}{\sum_{i=1}^{D}(u_i - \ell_i)^2}\right). \tag{20}$$

▶ **Corollary 7.** *For any constants $0 < \delta < \alpha$, $C > 60$, $0 < \rho_{\max} < 1$, and $\kappa > 1$, there exists a deterministic algorithm that solves almost all instances of Problem 6 in time*

$$O\left(n^{2-\frac{0.99(1-1/\kappa)(\alpha-\delta)}{4C+1}}\right)$$

assuming the parameters n, d, ρ satisfy the two constraints

1. $5\rho^{-2\kappa}\log n \le d \le n^{\delta}$ *and*
2. $n^{-\Theta(1)} \le \rho \le \rho_{\max}$. [11]

Proof. We reduce to (the single-set version of) Problem 1 and apply Theorem 4. Towards this end, in Theorem 4 set $\epsilon = 1 - 1/\kappa$ and $\tau_{\max} = \rho_{\max}^{\kappa}$. Suppose we are given an instance of Problem 6 whose parameters n, d, ρ satisfy the constraints. Set $\tau = \rho^{\kappa}$. We observe that the constraints in Theorem 4 are satisfied since (i) $d \le n^{\delta}$ holds by assumption, (ii) $\tau \le \tau_{\max}$ holds since $\tau = \rho^{\kappa} \le \rho_{\max}^{\kappa}$, (iii) since $\kappa > 1$ is a constant and $\tau = \rho^{\kappa}$ we can satisfy the requirement that $\tau \ge n^{-\Theta(1)}$ for any desired constant hidden by the $\Theta(1)$ notation [12] by our assumption that $\rho \ge n^{-\Theta(1)}$, and (iv) $\log_{\tau}\rho = \frac{\log \rho}{\log \tau} = \frac{\log \rho}{\log \rho^{\kappa}} = 1/\kappa \le 1 - \epsilon$.

We claim that $q = 1$ for almost all instances of Problem 6 whose parameters satisfy the constraints in Corollary 7. Indeed, by the Hoeffding bound (20) and the union bound, the probability that some other pair than the planted pair in an instance has inner product that exceeds τd in absolute value is at most

$$2n^2 \exp\left(-\tau^2 d/2\right) \le 2n^2 \exp\left(-\rho^{2\kappa} \cdot 5\rho^{-2\kappa}\log n\right) = 2n^{-1/2},$$

so $q = 1$ with high probability as n increases. The claimed running time follows by substituting the chosen constants and $q = 1$ to (5). ◀

References

1 Thomas D. Ahle, Rasmus Pagh, Ilya Razenshteyn, and Francesco Silvestri. On the complexity of inner product similarity join. *arXiv*, abs/1510.02824, 2015.

[11] The constant hidden by the $\Theta(1)$ notation depends on the constants $\delta, \alpha, C, \rho_{\max}$ but not on the other parameters. For details consult the proof.

[12] Consult the proof of Theorem 4 for the details of this constant.

2 Josh Alman and Ryan Williams. Probabilistic polynomials and Hamming nearest neighbors. In *Proc. 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 136–150, Los Alamitos, CA, USA, 2015. IEEE Computer Society.

3 Noga Alon. Problems and results in extremal combinatorics – I. *Discrete Math.*, 273(1-3):31–53, 2003.

4 Alexandr Andoni, Piotr Indyk, Huy L. Nguyen, and Ilya Razenshteyn. Beyond locality-sensitive hashing. In *Proc. 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1018–1028, Philadelphia, PA, USA, 2014. Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611973402.76.

5 Alexandr Andoni and Ilya Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In *Proc. 47th ACM Annual Symposium on the Theory of Computing (STOC)*, pages 793–801, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2746539.2746553.

6 L. Elisa Celis, Omer Reingold, Gil Segev, and Udi Wieder. Balls and bins: Smaller hash families and faster evaluation. *SIAM J. Comput.*, 42(3):1030–1050, 2013.

7 Timothy M. Chan and Ryan Williams. Deterministic APSP, orthogonal vectors, and more: Quickly derandomizing Razborov-Smolensky. In Robert Krauthgamer, editor, *Proc. 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1246–1255, Arlington, VA, USA, 2016. Society for Industrial and Applied Mathematics.

8 Moshe Dubiner. Bucketing coding and information theory for the statistical high-dimensional nearest-neighbor problem. *IEEE Trans. Inf. Theory*, 56(8):4166–4179, 2010. doi:10.1109/TIT.2010.2050814.

9 Vitaly Feldman, Parikshit Gopalan, Subhash Khot, and Ashok Kumar Ponnuswami. On agnostic learning of parities, monomials, and halfspaces. *SIAM J. Comput.*, 39(2):606–645, 2009. doi:10.1137/070684914.

10 Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In Malcolm P. Atkinson, Maria E. Orlowska, Patrick Valduriez, Stanley B. Zdonik, and Michael L. Brodie, editors, *Proc. 25th International Conference on Very Large Data Bases (VLDB'99)*, pages 518–529, Edinburgh, Scotland, UK, 1999. Morgan Kaufmann.

11 Parikshit Gopalan, Daniek Kane, and Raghu Meka. Pseudorandomness via the Discrete Fourier Transform. In *Proc. IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 903–922, Berkeley, CA, USA, 2015. IEEE Computer Society.

12 Parikshit Gopalan, Raghu Meka, Omer Reingold, and David Zuckerman. Pseudorandom generators for combinatorial shapes. *SIAM J. Comput.*, 42(3):1051–1076, 2013.

13 Elena Grigorescu, Lev Reyzin, and Santosh Vempala. On noise-tolerant learning of sparse parities and related problems. In *Proc. 22nd International Conference on Algorithmic Learning Theory (ALT)*, pages 413–424, Berlin, Germany, 2011. Springer. doi:10.1007/978-3-642-24412-4_32.

14 Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *J. Amer. Statist. Assoc.*, 58:13–30, 1963.

15 Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bull. Amer. Math. Soc.*, 43(4):439–561, 2006.

16 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

17 Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proc. 30th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 604–613, New York, NY, USA, 1998. Association for Computing Machinery. doi:10.1145/276698.276876.

18 Daniel M. Kane, Raghu Meka, and Jelani Nelson. Almost optimal explicit Johnson-Lindenstrauss families. In *Proc. 14th International Workshop on Approximation, Randomization, and Combinatorial Optimization, RANDOM and 15th International Workshop on Algorithms and Techniques, APPROX*, pages 628–639, Princeton, NJ, USA, 2011.

19 Michael Kapralov. Smooth tradeoffs between insert and query complexity in nearest neighbor search. In *Proc. 34th ACM Symposium on Principles of Database Systems (PODS)*, pages 329–342, New York, NY, USA, 2015. Association for Computing Machinery. `doi:10.1145/2745754.2745761`.

20 Matti Karppa, Petteri Kaski, and Jukka Kohonen. A faster subquadratic algorithm for finding outlier correlations. In *Proc. 27th Annual ACM-SIAM Symposium on Discrete Algorithms, (SODA)*, pages 1288–1305, Arlington, VA, USA, 2016. Society for Industrial and Applied Mathematics.

21 Pravesh K. Kothari and Raghu Meka. Almost optimal pseudorandom generators for spherical caps. In *Proc. 47th Annual ACM Symposium on Theory of Computing (STOC)*, pages 247–256, Portland, OR, USA, 2015.

22 François Le Gall. Faster algorithms for rectangular matrix multiplication. In *Proc. 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 514–523, Los Alamitos, CA, USA, 2012. IEEE Computer Society. `doi:10.1109/FOCS.2012.80`.

23 A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8:261–277, 1988.

24 Alexander May and Ilya Ozerov. On computing nearest neighbors with applications to decoding of binary linear codes. In *Proc. EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 203–228, Berlin, Germany, 2015. Springer. `doi:10.1007/978-3-662-46800-5_9`.

25 Elchanan Mossel, Ryan O'Donnell, and Rocco A. Servedio. Learning functions of k relevant variables. *J. Comput. Syst. Sci.*, 69(3):421–434, 2004. `doi:10.1016/j.jcss.2004.04.002`.

26 Rajeev Motwani, Assaf Naor, and Rina Panigrahy. Lower bounds on locality sensitive hashing. *SIAM J. Discrete Math.*, 21(4):930–935, 2007. `doi:10.1137/050646858`.

27 Ryan O'Donnell, Yi Wu, and Yuan Zhou. Optimal lower bounds for locality-sensitive hashing (except when q is tiny). *ACM Trans. Comput. Theory*, 6(1):Article 5, 2014. `doi:10.1145/2578221`.

28 Rasmus Pagh. Locality-sensitive hashing without false negatives. In *Proc. 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1–9, Philadelphia, PA, USA, 2016. Society for Industrial and Applied Mathematics.

29 Ramamohan Paturi, Sanguthevar Rajasekaran, and John H. Reif. The light bulb problem. In *Proc. 2nd Annual Workshop on Computational Learning Theory (COLT)*, pages 261–268, New York, NY, USA, 1989. Association for Computing Machinery.

30 Ninh Pham and Rasmus Pagh. Scalability and total recall with fast CoveringLSH. *arXiv*, abs/1602.02620, 2016.

31 Omer Reingold, Salil Vadhan, and Avi Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders. *Ann. of Math.*, 155(1):157–187, 2002.

32 Victor Shoup. New algorithms for finding irreducible polynomials over finite fields. *Math. Comp.*, 54:435–447, 1990.

33 Gregory Valiant. Finding correlations in subquadratic time, with applications to learning parities and the closest pair problem. *J. ACM*, 62(2):Article 13, 2015. `doi:10.1145/2728167`.

34 Leslie G. Valiant. Functionality in neural nets. In *Proc. 1st Annual Workshop on Computational Learning Theory (COLT)*, pages 28–39, New York, NY, USA, 1988. Association for Computing Machinery.

A An expander family

This section proves Lemma 9 following Reingold, Vadhan and Wigderson [31]; we present the proof for completeness of exposition only with no claim of originality. Following Reingold, Vadhan and Wigderson [31] we will work with *normalized* eigenvalues. To avoid confusion with the unnormalized treatment in the manuscript proper, we say that a graph is a $[D,\Delta,\lambda]$-*graph* if the graph has D vertices, is Δ-regular, and $|\lambda_2|/\Delta \leq \lambda$. (Here $|\lambda_2|$ is the unnormalized second eigenvalue as defined in the manuscript proper.)

We refer to Sections 2.3 and 3.1 of Reingold, Vadhan, and Wigderson [31] for the definition of the square G^2 of a graph G, the tensor product $G_1 \otimes G_2$ of graphs G_1, G_2, and the zigzag product $G \circledz H$ of graphs G, H. The following omnibus result collects elements of Propositions 2.3, Proposition 2.4, Theorem 3.2 and Theorem 4.3 of [31] which will be sufficient to control the second normalized eigenvalue for our present purposes. (We choose to omit the details of the rotation maps with the understanding that they can be found in [31].)

▶ **Lemma 20** (Reingold, Vadhan, and Wigderson [31]). *The following bounds hold.*

1. *If G is a $[D,\Delta,\lambda]$-graph, then G^2 is a $[D,\Delta^2,\lambda^2]$-graph.*
2. *If G_1 is a $[D_1,\Delta_1,\lambda_1]$-graph and G_2 is a $[D_2,\Delta_2,\lambda_2]$-graph, then $G_1 \otimes G_2$ is a $[D_1D_2,\Delta_1\Delta_2,\max(\lambda_1,\lambda_2)]$-graph.*
3. *If G is a $[D_1,\Delta_1,\lambda_1]$-graph and H a $[\Delta_1,\Delta_2,\lambda_2]$-graph, then $G \circledz H$ is a $[D_1\Delta_1,\Delta_2^2,f(\lambda_1,\lambda_2)]$-graph with*

$$f(\lambda_1,\lambda_2) = \frac{1}{2}\left(1-\lambda_2^2\right)\lambda_1 + \frac{1}{2}\sqrt{\left(1-\lambda_2^2\right)^2\lambda_1^2 + 4\lambda_2^2} \leq \lambda_1 + \lambda_2\,.$$

Let us study the following sequence of graphs. Let H be a $[D,\Delta,\lambda]$-graph. Let $G_1 = H^2$, $G_2 = H \otimes H$, and for $t = 3,4,\ldots$ let

$$G_t = \left(G_{\lceil\frac{t-1}{2}\rceil} \otimes G_{\lfloor\frac{t-1}{2}\rfloor}\right)^2 \circledz H\,. \tag{21}$$

From Lemma 20 it is easily seen that G_t is a $[D^t,\Delta^2,\lambda_t]$-graph with λ_t defined by

$$\begin{aligned}
\lambda_1 &= \lambda^2\,, \\
\lambda_2 &= \lambda\,, \\
\lambda_{2t-1} &= \lambda + \lambda_{t-1}^2\,, && \text{for } t = 2,3\ldots\,, \text{ and} \\
\lambda_{2t} &= \max(\lambda+\lambda_t^2, \lambda+\lambda_{t-1}^2)\,, && \text{for } t = 2,3,\ldots\,.
\end{aligned}$$

▶ **Lemma 21** (Reingold, Vadhan, and Wigderson [31, Theorem 3.3]). *The rotation map* Rot_{G_t} *can be computed in time* $\mathrm{poly}(t,\log D)$ *and by making* $\mathrm{poly}(t)$ *evaluations of* Rot_H.

▶ **Lemma 22.** *If $0 \leq \lambda \leq 1/4$ then $\lambda_t \leq \lambda + 4\lambda^2$ for all $t \geq 1$.*

Proof. The conclusion is immediate for $t \leq 2$. So suppose that the conclusion holds up to $2t-2$. We need to show that the conclusion holds for λ_{2t-1} and λ_{2t}. By induction, it suffices to show that

$$\lambda_{2t-1} \leq \lambda + (\lambda+4\lambda^2)^2 \leq \lambda + 4\lambda^2\,.$$

Observing that $\lambda^2 + 8\lambda^3 + 16\lambda^4 \leq 4\lambda^2$ holds for $0 \leq \lambda \leq 1/4$ yields the desired conclusion. The proof for λ_{2t} is identical. ◀

Finally, we construct the expanders that we require in the manuscript proper.

▶ **Lemma 23** (Lemma 9 stated with normalized eigenvalue notation)**.** *For all integers* $t \geq 1$ *and* $b \geq 10$ *there exists a* $[2^{16bt}, 2^{4b}, 16 \cdot 2^{-b}]$*-graph whose rotation map can be evaluated in time* $\mathrm{poly}(b, t)$*.*

Proof. Take $q = 2^b$ and $d = 15$ in Proposition 5.3 of Reingold, Vadhan, and Wigderson [31] to obtain a $[2^{16b}, 2^{2b}, 15 \cdot 2^{-b}]$-graph H whose rotation map can be computed in time $\mathrm{poly}(b)$. (Indeed, observe that an irreducible polynomial to perform the required arithmetic in the finite field of order 2^b can be constructed in deterministic time $\mathrm{poly}(b)$ by an algorithm of Shoup [32].) Let us study the sequence G_t given by (21). The time complexity of the rotation map follows immediately from Lemma 21. Since $b \geq 10$, Lemma 22 gives that $\lambda_t \leq \lambda + 4\lambda^2$ for all $t \geq 1$. Take $\lambda = 15 \cdot 2^{-b}$ and observe that since $b \geq 10$ we have $2^{-b} < 1/900$. Thus, $\lambda_t \leq 15 \cdot 2^{-b} + 4(15 \cdot 2^{-b})^2 = 15 \cdot 2^{-b} + 900 \cdot 2^{-2b} \leq 16 \cdot 2^{-b}$. ◀

Faster Worst Case Deterministic Dynamic Connectivity*

Casper Kejlberg-Rasmussen[1], **Tsvi Kopelowitz**[2], **Seth Pettie**[3], **and Mikkel Thorup**[†4]

1 **MADALGO, Aarhus University, Aarhus, Denmark**
`ckr@cs.au.dk`
2 **University of Michigan, Ann Arbor, USA**
`pettie,tsvi@umich.edu`
3 **University of Michigan, Ann Arbor, USA**
`pettie,tsvi@umich.edu`
4 **University of Copenhagen, Copenhagen, Denmark**
`mthorup@di.ku.dk`

Abstract

We present a deterministic dynamic connectivity data structure for undirected graphs with worst case update time $O\left(\sqrt{\frac{n(\log\log n)^2}{\log n}}\right)$ and constant query time. This improves on the previous best deterministic worst case algorithm of Frederickson (*SIAM J. Comput.*, 1985) and Eppstein Galil, Italiano, and Nissenzweig (*J. ACM*, 1997), which had update time $O(\sqrt{n})$. All other algorithms for dynamic connectivity are either randomized (Monte Carlo) or have only amortized performance guarantees.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

Keywords and phrases dynamic graph, spanning tree

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.53

1 Introduction

Dynamic Connectivity is perhaps the single most fundamental unsolved problem in the area of dynamic graph algorithms. The problem is simply to maintain a dynamic undirected graph $G = (V, E)$ subject to edge updates and connectivity queries:
INSERT(u, v) : Set $E \leftarrow E \cup \{(u, v)\}$.
DELETE(u, v) : Set $E \leftarrow E \setminus \{(u, v)\}$.
CONN?(u, v) : Determine whether u and v are in the same connected component in G.

Over thirty years ago Frederickson [10] introduced *topology trees* and *2-dimensional* topology trees, which gave the first non-trivial solution to the problem. Each edge insertion/deletion is handled in $O(\sqrt{m})$ time and each query is handled in $O(1)$ time. Here m is the current number of edges and n the number of vertices. On sparse graphs (where

* This work is supported in part by the Danish National Research Foundation grant DNRF84 through the Center for Massive Data Algorithmics (MADALGO), and by NSF grants CCF-1217338, CNS-1318294, and CCF-1514383.

† M. Thorup's research is partly supported by Advanced Grant DFF-0602-02499B from the Danish Council for Independent Research under the Sapere Aude research career programme.

$m = O(n)$) Frederickson's data structure has not been improved by any deterministic worst case algorithm. However, when the graph is dense Frederickson's data structure can be improved using the general sparsification method of Eppstein, Galil, Italiano, and Nissenzweig [7]. Using simple sparsification [6] the update time becomes $O(\sqrt{n}\log(m/n))$ and using more sophisticated sparsification [7] the running time becomes $O(\sqrt{n})$. This last bound has not been improved in twenty years.

1.1 New Results

In this paper we return to the classical model of deterministic worst case complexity. We give a new dynamic connectivity structure with worst case update time on the order of

$$\min\left\{\sqrt{\frac{m(\log\log n)^2}{\log n}},\quad \sqrt{\frac{m\log^5 w}{w}}\right\},$$

where $w = \Omega(\log n)$ is the word size.[1] These are the first improvements to Frederickson's 2D-topology trees [10] in over 30 years. Using the sparsification reduction of Eppstein et al. [7] the running time expressions can be made to depend on 'n' rather than 'm', so we obtain $O(\sqrt{\frac{n(\log\log n)^2}{\log n}})$ bounds (or faster) for all graph densities.

1.2 Related Work

Most research on the dynamic connectivity problem has settled for *amortized* update time guarantees. Following [15, 16], Holm et al. [17] gave a very simple deterministic algorithm with amortized update time $O(\log^2 n)$ and query time $O(\log n/\log\log n)$.[2] However, in the worst case Holm et al.'s [17] update takes $\Omega(m)$ time, the same as computing a spanning tree from scratch! Recently Wulff-Nilsen [25] improved the update time of [17] to $O(\log^2 n/\log\log n)$. Using Las Vegas randomization, Thorup [24] gave a dynamic connectivity data structure with an $O(\log n(\log\log n)^3)$ amortized update time. In other words, the algorithm answers all connectivity queries correctly but the amortized update time holds with high probability.

In a major breakthrough Kapron, King, and Mountjoy [18] used Monte Carlo randomization to achieve a worst case update time of $O(\log^5 n)$. However, this algorithm has three notable drawbacks. The first is that it is susceptible to undetected false negatives: $\textsc{Conn?}(u, v)$ may report that u, v are disconnected when they are, in fact, connected. The second is that even when $\textsc{Conn?}(u, v)$ (correctly) reports that u, v are connected, it is forbidden from exhibiting a connectivity witness, i.e., a spanning forest in which u, v are joined by a path. The Kapron et al. [18] algorithm <u>does</u> maintain such a spanning forest internally, but if this witness were made public, a very simple attack could force the algorithm to answer connectivity queries incorrectly. Lastly, the algorithm uses $\Omega(n\log^2 n)$ space, which for sparse graphs is superlinear in m. Very recently Gibb et al. [13] reduced the update time of [18] to $O(\log^4 n)$.

On special graph classes, dynamic connectivity can often be handled more efficiently. For example, Sleator and Tarjan [22] maintain a dynamic set of trees in $O(\log n)$ worst-case update time subject to $O(\log n)$ time connectivity queries. (See also [1, 3, 15, 23].)

[1] Our algorithms use the standard repertoire of AC^0 operations: left and right shifts, bitwise operations on words, additions and comparisons. They do not assume unit-time multiplication.

[2] Any connectivity structure that maintains (internally) a spanning forest can have query time $O(\log_{t_u/\log n} n)$ if the update time is $t_u = \Omega(\log n)$.

Table 1 A survey of dynamic connectivity results. The lower bounds hold in the cell probe model with word size $w = \Theta(\log n)$.

Worst Case Data Structures

Ref.	Update Time	Query Time	Notes
[10]	$O(\sqrt{m})$	$O(1)$	
[7, 10]	$O(\sqrt{n})$	$O(1)$	[10] + sparsification [7].
[18]	$O(c\log^5 n)$	$O\left(\frac{\log n}{\log\log n}\right)$	Randomized Monte Carlo; no connectivity witness;
[13]	$O(c\log^4 n)$	$O\left(\frac{\log n}{\log\log n}\right)$	n^c opers. err with prob. n^{-c}.
new	$O\left(\sqrt{\frac{n(\log\log n)^2}{\log n}}\right)$ $O\left(\sqrt{\frac{n\log^5 w}{w}}\right)$	$O(1)$	$w = \Omega(\log n)$

Amortized Data Structures

Ref.	Amort. Update	W.C. Query	Notes
[15]	$O(\log^3 n)$	$O\left(\frac{\log n}{\log\log n}\right)$	Randomized Las Vegas.
[16]	$O(\log^2 n)$	$O\left(\frac{\log n}{\log\log n}\right)$	Randomized Las Vegas.
[17]	$O(\log^2 n)$	$O\left(\frac{\log n}{\log\log n}\right)$	
[24]	$O(\log n(\log\log n)^3)$	$O\left(\frac{\log n}{\log\log\log n}\right)$	Randomized Las Vegas.
[25]	$O\left(\frac{\log^2 n}{\log\log n}\right)$	$O\left(\frac{\log n}{\log\log n}\right)$	

Amort./Worst Case Lower Bounds

Ref.	Update Time t_u	Query Time t_q	Notes
[11, 14, 19]		$t_q = \Omega\left(\frac{\log n}{\log(t_u \log n)}\right)$	
[20]	$t_u = \Omega\left(\frac{\log n}{\log(t_q/t_u)}\right)$	$t_q = \Omega\left(\frac{\log n}{\log(t_u/t_q)}\right)$	Implies $\max\{t_u, t_q\} = \Omega(\log n)$.
[21]	$o(\log n)$ implies	$\Omega(n^{1-o(1)})$	

Connectivity in dynamic planar graphs can be reduced to the dynamic tree problem [8, 9], and therefore solved in $O(\log n)$ time per operation. The cell probe lower bounds of Pătraşcu and Demaine [20] show that Sleator and Tarjan's algorithm is optimal in the sense that *some* operation must take $\Omega(\log n)$ time. Superlogarithmic updates can be used to get modestly sublogarithmic queries, but Pătraşcu and Thorup [21] prove the reverse is not possible. In particular, any dynamic connectivity algorithm with $o(\log n)$ update time has $n^{1-o(1)}$ query time. Refer to Table 1 for a history of upper and lower bounds for dynamic connectivity.

Compared to the amortized algorithms [17, 24, 25], ours is better suited to *online* applications that demand a bound on the latency of *every* operation.[3] Compared to the Monte Carlo algorithms [13, 18], ours is attractive in applications that demand linear space, zero probability of error, and a public witness of connectivity.

2 The High Level Algorithm

The algorithm maintains a spanning tree of each connected component of the graph as a *witness* of connectivity. Each such witness tree T is represented as an Euler tour $\text{Euler}(T)$.[4] $\text{Euler}(T)$ is the sequence of vertices encountered in some Euler tour around T, as if each undirected edge were replaced by two oriented edges. It has length precisely $2(|V(T)|-1)$ if $|V(T)| \geq 2$ (the last vertex is excluded from the list, which is necessarily the same as the first) or length 1 if $|V(T)| = 1$. Vertices may appear in $\text{Euler}(T)$ several times. We designate one copy of each vertex the *principal copy*, which is responsible for all edges incident to the vertex. Each vertex in the graph maintains a pointer to its principal copy. Each T-edge (u, v) maintains two pointers to the (possibly non-principal) copies of u and v that precede the oriented occurrences of (u, v) and (v, u) in $\text{Euler}(T)$, respectively. Note that cyclic rotations of $\text{Euler}(T)$ are also valid Euler tours; if $\text{Euler}(T) = (u, \ldots, v)$ the last element of the list is associated with the tree edge (v, u).

When an edge (u, v) that connects distinct witness trees T_0 and T_1 is inserted, (u, v) becomes a tree edge and we need to construct $\text{Euler}(T_0 \cup \{(u, v)\} \cup T_1)$ from $\text{Euler}(T_0)$ and $\text{Euler}(T_1)$. In the reverse situation, if a tree edge (u, v) is deleted from $T = T_0 \cup \{(u, v)\} \cup T_1$ we first construct $\text{Euler}(T_0)$ and $\text{Euler}(T_1)$ from $\text{Euler}(T)$, then look for a *replacement edge*, $(\hat{u}, \hat{v})$ with $\hat{u} \in V(T_0)$ and $\hat{v} \in V(T_1)$. If a replacement is found we construct $\text{Euler}(T_0 \cup \{(\hat{u}, \hat{v})\} \cup T_1)$ from $\text{Euler}(T_0)$ and $\text{Euler}(T_1)$. Lemma 1 establishes the nearly obvious fact that the new Euler tours can be obtained from the old Euler tours using $O(1)$ of the following *surgical operations*: splitting and concatenating lists of vertices, and creating and destroying singleton lists containing non-principal copies of vertices.

▶ **Lemma 1.** *If $T = T_0 \cup \{(u, v)\} \cup T_1$ and (u, v) is deleted,* $\text{Euler}(T_0)$ *and* $\text{Euler}(T_1)$ *can be constructed from* $\text{Euler}(T)$ *with $O(1)$ surgical operations. In the opposite direction, from* $\text{Euler}(T_0)$ *and* $\text{Euler}(T_1)$ *we can construct* $\text{Euler}(T_0 \cup \{(u, v)\} \cup T_1)$ *with $O(1)$ surgical operations. It takes $O(1)$ time to determine which surgical operations to perform.*

Proof. Recall that cyclic shifts of Euler tours are valid Euler tours. Suppose without loss of generality that $\text{Euler}(T) = (P_0, u, v, P_1, v, u, P_2)$ where P_0, P_1, and P_2 are sequences of

[3] Amortized data structures are most useful when employed by offline algorithms that do not care about individual operation times. The canonical example is the use of amortized Fibonacci heaps [12] to implement Dijkstra's algorithm [5].

[4] Henzinger and King [15] were the first to use Euler tours to represent dynamic trees. G. Italiano (personal communication) observed that Euler tours could be used in lieu of Frederickson's topology trees to obtain an $O(\sqrt{m})$-time dynamic connectivity structure.

vertices. (Note that Euler tours never contain immediate repetitions. If P_1 is empty then Euler(T) would be just (P_0, u, v, u, P_2); if both P_0 and P_2 are empty then Euler$(T) = (u, v, P_1, v)$.) Then we obtain Euler$(T_0) = (P_0, u, P_2)$ and Euler$(T_1) = (v, P_1)$ with $O(1)$ surgical operations, which includes the destruction of non-principal copies of u and v; at least one of the two copies must be non-principal. We could also set Euler$(T_1) = (P_1, v)$, which would be more economical if the v following P_1 in Euler(T) were the principal copy.

In the reverse direction, write Euler$(T_0) = (P_0, u, P_1)$ and Euler$(T_1) = (P_2, v, P_3)$, where the labeled occurrences are the principal copies of u and v. Then Euler$(T_0 \cup \{(u, v)\} \cup T_1) = (P_0, u, v, P_3, P_2, v, u, P_1)$, where the new copies of u and v are clearly non-principal copies. If P_2 and P_3 were empty (or P_0 and P_1 were empty) then we would not need to add a non-principal copy of v (or a non-principal copy of u.) ◀

Thus, we have *reduced* dynamic connectivity in graphs to implementing several simple operations on dynamic lists. Our algorithm maintains a pair $(\mathcal{L}, E)$, where $\mathcal{L}$ is a set of lists (containing principal and non-principal copies of vertices) and E is the dynamic set of edges joining principal copies of vertices. In addition to the creation and destruction of single element lists we must support the following primitive operations.

List(x) : Return the list in $\mathcal{L}$ containing element x.

Join(L_0, L_1) : Set $\mathcal{L} \leftarrow \mathcal{L} \setminus \{L_0, L_1\} \cup \{L_0L_1\}$, that is, replace L_0 and L_1 with their concatenation L_0L_1.

Split(x) : Let $L = L_0L_1 \in \mathcal{L}$, where x is the last element of L_0. Set $\mathcal{L} \leftarrow \mathcal{L}\setminus\{L\}\cup\{L_0, L_1\}$.

ReplacementEdge(L_0, L_1) : Return any edge joining elements in L_0 and L_1.

Our implementations of these operations will only be efficient if, after each Insert or Delete operation, there are no edges connecting distinct lists. That is, the ReplacementEdge operation is only employed by Delete when deleting a tree edge in order to restore Invariant 2.

▶ **Invariant 2.** *Each list $\mathcal{L}$ corresponds to the Euler tour of a spanning tree of some connected component.*

The dynamic connectivity operations are implemented as follows. To answer a Conn?(u, v) query we simply check whether List(u) = List(v). To insert an edge (u, v) we do Insert(u, v), and if List$(u) \neq$ List(v) then make (u, v) a tree edge and perform suitable Splits and Joins to merge the Euler tours List(u) and List(v). To delete an edge (u, v) we do Delete(u, v), and if (u, v) is a tree edge in $T = T_0 \cup \{(u, v)\} \cup T_1$, perform suitable Splits and Joins to create Euler(T_0) and Euler(T_1) from Euler(T). At this point Invariant 2 may be violated as there could be an edge joining T_0 and T_1. We call ReplacementEdge(Euler(T_0), Euler(T_1)) and if it finds an edge, say $(\hat{u}, \hat{v})$, we perform more Splits and Joins to form Euler$(T_0 \cup \{(\hat{u}, \hat{v})\} \cup T_1)$.

Henzinger and King [15] observed that most off-the-shelf balanced binary search trees can support Split, Join, and other operations in logarithmic time. However, they provide no direct support for the ReplacementEdge operation, which is critical for the dynamic connectivity application.

Section 3 gives a relatively simple instantiation of the high-level approach with update time $O(\sqrt{n}/w^{1/4})$, $w = \Omega(\log n)$ being the word size. This is slightly slower than our claimed result. In Section 4 we describe the modifications needed to achieve the claimed bounds.

3 A New Data Structure for Dynamic Lists

3.1 Chunks and Superchunks

In order to simplify the maintenance of Invariant 3, stated below, we shall make two simplifying assumptions. We assume that we have a fixed upper bound $\hat{m}$ on the number of edges and that the maximum degree never exceeds K, where $K \approx \sqrt{\hat{m}/\operatorname{poly}(w)}$. The first assumption is justified by the fact that the sparsification method of [7] creates instances in which $\hat{m}$ is known to be linear in the number of vertices. (It can also be removed by the standard technique of periodic rebuilding.) Refer to Section 5.1 for clean ways to remove the degree-bound assumption.

If L' is a sublist of a list $L \in \mathcal{L}$, define $\operatorname{mass}(L')$ to be the number of edges incident to elements of L', counting an edge twice if both endpoints are in L'.[5] The sum of list masses, $\sum_{L\in\mathcal{L}} \operatorname{mass}(L)$, is clearly at most $2\hat{m}$, where $\hat{m}$ is the fixed upper bound on the number of edges. We maintain a partition of each list $L \in \mathcal{L}$ into *chunks* satisfying Invariant 3.

▶ **Invariant 3.** *Let $L \in \mathcal{L}$ be an Euler tour. If $\operatorname{mass}(L) < K$ then L consists of a single chunk. Otherwise $L = C_0C_1 \cdots C_{p-1}$ is partitioned into $\Theta(\operatorname{mass}(L)/K)$ chunks such that $\operatorname{mass}(C_l) \in [K, 3K]$ for all $l \in [p] \overset{\text{def}}{=} \{0, \ldots, p-1\}$.*

The chunks are partitioned into contiguous sequences of $\Theta(h)$ superchunks according to Invariant 4. For the time being define $h = 2\lfloor\sqrt{w}/2\rfloor$, where w is the word size.

▶ **Invariant 4.** *A list in $\mathcal{L}$ having fewer than $h/2$ chunks forms a single superchunk with ID $\perp$. A list in $\mathcal{L}$ with at least $h/2$ chunks is partitioned into superchunks, each consisting of between $h/2$ and $h-1$ consecutive chunks. Each such superchunk has a unique ID in $[J] \overset{\text{def}}{=} \{0, \ldots, J-1\}$, where $J = 4\hat{m}/(Kh)$. (IDs are completely arbitrary. They do not encode any information about the order of superchunks within a list.)*

Call an Euler tour list *short* if it consists of fewer than $h/2$ chunks. We shall assume that no lists are ever short, as this simplifies the description of the data structure and its analysis. In particular, all superchunks have proper IDs in $[J]$. Refer to Section 5.2 for a description of how to handle $\perp$ IDs and short lists.

3.2 Word Operations

When $h \le \lfloor\sqrt{w}\rfloor$, Invariant 4 implies that we can store a matrix $A \in \{0,1\}^{h\times h}$ in one word that represents the adjacency between the chunks within two superchunks i and j. This matrix will always be represented in row-major order; rows and columns are indexed by $[h] = \{0, \ldots, h-1\}$. In this format it is straightforward to insert a new all-zero row above a specified row k (and destroy row $h-1$) by shifting the old rows $k, \ldots, h-2$ down by one. It is also easy to copy an interval of rows from one matrix to another. Lemma 5 shows that the corresponding operations on *columns* can also be effected in $O(1)$ time with a fixed mask μ precomputable in $O(\log w)$ time.

▶ **Lemma 5.** *Let $h = 2\lfloor\sqrt{w}/2\rfloor$ and let μ be the word $(1^h0^h)^{h/2}$. Given μ we can in $O(1)$ time copy/paste any interval of columns from/to a matrix $A \in \{0,1\}^{h\times h}$, represented in row-major order.*

[5] Remember that edges are only incident to principal copies of vertices, so non-principal copies never contribute any mass.

Proof. Recall that the rows and columns are indexed by integers in $[h] = \{0, \ldots, h-1\}$. We first describe how to build a mask ν_k for columns $k, \ldots, h-1$ then illustrate how it is used to copy/paste intervals of columns. In C notation,[6] the word $\nu'_k = (\mu \gg k) \,\&\, \mu$ is a mask for the intersection of the even rows and columns $k, \ldots, h-1$, so $\nu_k = \nu'_k \mid (\nu'_k \gg h)$ is a mask for columns k through $h-1$.

To insert an all-zero column before column k of A (and delete column $h-1$) we first copy columns $k, \ldots, h-2$ to $A' = A \,\&\, (\nu_{k+1} \ll 1)$ then set $A = (A \,\&\, (\sim\nu_k)) \mid (A' \gg 1)$. Other operations can be effected in $O(1)$ time with copying/pasting intervals of columns, e.g., splitting an array into two about a designated column, or merging two arrays having at most h columns together. ◀

3.3 Adjacency Data Structures

In order to facilitate the efficient implementation of ReplacementEdge we maintain an $O(\hat{m}/K) \times O(\hat{m}/K)$ adjacency matrix between chunks, and a $J \times J$ adjacency matrix between superchunks. However, in order to allow for efficient dynamic updates it is important that these matrices be represented in a non-standard format described below. The data structure maintains the following information.

- Each list element maintains a pointer to the chunk containing it. Each chunk maintains a pointer to the superchunk containing it, as well as an index in $[h]$ indicating its position within the superchunk. Each superchunk maintains its ID in $[J] \cup \{\perp\}$ and a pointer to the list containing it.
- ChAdj is a $J \times J$ array of h^2-bit words ($h^2 \leq w$) indexed by superchunk IDs. The entry $\text{ChAdj}(i,j)$ is interpreted as an $h \times h$ 0-1 matrix that keeps the adjacency information between all pairs of chunks in superchunk i and superchunk j. (It may be that $i = j$.) In particular, $\text{ChAdj}(i,j)(k,l) = 1$ iff there is an edge with endpoints in the kth chunk of superchunk i and the lth chunk of superchunk j, so $\text{ChAdj}(i,j) = 0$ (i.e., the all-zero matrix) if no edge joins superchunks i and j. The matrix $\text{ChAdj}(i,j)$ is stored in row-major order.
- Let S be a superchunk with $ID(S) = \perp$. By Invariants 2 and 4, S is not incident to any other superchunks and has fewer than $h/2$ chunks. We maintain a single word ChAdj_S which stores the adjacency matrix of the chunks within S.
- For each superchunk with ID $i \in [J]$ we keep length-J bit-vectors SupAdj_i and Memb_i, where

  $$\begin{aligned} \text{SupAdj}_i(j) &= 1 \text{ if } \text{ChAdj}(i,j) \neq 0 \text{ and } 0 \text{ otherwise, whereas} \\ \text{Memb}_i(j) &= 1 \text{ if } j = i \text{ and } 0 \text{ otherwise.} \end{aligned}$$

  These vectors are packed into $\lceil J/w \rceil$ machine words, so scanning one takes $O(\lceil J/w \rceil)$ time.
- We maintain a *list-sum* data structure that allows us to take the bit-wise OR of the SupAdj_i vectors or Memb_i vectors, over all superchunks in an Euler tour. It is responsible for maintaining the $\{\text{SupAdj}_i, \text{Memb}_i\}$ vectors described above and supports the following operations. At all times the superchunks are partitioned into a set $\mathcal{S}$ of disjoint lists of superchunks. Each $S \in \mathcal{S}$ (a list of superchunks) is associated with an $L \in \mathcal{L}$ (an Euler tour), though short lists in $\mathcal{L}$ have no need for a corresponding list in $\mathcal{S}$.

[6] The operations &, |, and $\sim$ are bit-wise AND, OR, and NOT; $\ll$ and $\gg$ are left and right shift.

SCInsert(i) : Retrieve an unused ID, say i', and allocate a new superchunk with ID i' and all-zero vector $\mathrm{SupAdj}_{i'}$. Insert superchunk i' immediately after superchunk i in i's list in $\mathcal{S}$. If no i is given, create a new list in $\mathcal{S}$ consisting of superchunk i'.

SCDelete(i) : Delete superchunk i from its list and make ID i unused.

SCJoin(S_0, S_1) : Replace superchunk lists $S_0, S_1 \in \mathcal{S}$ with their concatenation S_0S_1.

SCSplit(i) : Let $S = S_0S_1 \in \mathcal{S}$ and i be the last superchunk in S_0. Replace S_0S_1 with two lists S_0, S_1.

UpdateAdj($i, x \in \{0,1\}^J$) : Set $\mathrm{SupAdj}_i \leftarrow x$ and update $\mathrm{SupAdj}_j(i) \leftarrow x(j)$ for all $j \neq i$.

AdjQuery(S) : Return the vector $\alpha \in \{0,1\}^J$ where

$$\alpha(j) = \bigvee_{i \in S} \mathrm{SupAdj}_i(j)$$

The index i ranges over the IDs of all superchunks in S.

MembQuery(S) : Return the vector $\beta \in \{0,1\}^J$, where

$$\beta(j) = \bigvee_{i \in S} \mathrm{Memb}_i(j)$$

We use the following implementation of the *list-sum* data structure. Each list of superchunks is maintained as any $O(1)$-degree search tree that supports logarithmic time inserts, deletes, splits, and joins. Each leaf is a superchunk that stores its two bit-vectors. Each internal node z keeps two bit-vectors, SupAdj^z and Memb^z, which are the bit-wise OR of their leaf descendants' respective bit-vectors. Because length-J bit-vectors can be updated in $O(\lceil J/w \rceil)$ time, all "logarithmic time" operations on the tree actually take $O(\log J \cdot J/w)$ time. The UpdateAdj(i, x) operation takes $O(\log J \cdot J/w)$ time to update superchunk i and its $O(\log J)$ ancestors. We then need to update the ith bit of potentially every other node in the tree, in $O(J)$ time. Since $w = \Omega(\log n) = \Omega(\log J)$ the cost per UpdateAdj is $O(J)$. The answer to an AdjQuery(S) or MembQuery(S) is stored at the root of the tree on S.

3.4 Creating and Destroying (Super)Chunks

There are essentially two causes for the creation and destruction of (super)chunks. The first is in response to a Split operation that forces a (super)chunk to be broken up. (The Split may itself be instigated by the insertion or deletion of an edge.) The second is to restore Invariants 3 and 4 after a Join or Insert or Delete operation. In this section we consider the problem of updating the adjacency data structures after four types of operations: (i) splitting a chunk in two, keeping both chunks in the same superchunk, (ii) merging two adjacent chunks in the same superchunk, (iii) splitting a superchunk along a chunk boundary, and (iv) merging adjacent superchunks. Once we have bounds on (i)–(iv), implementing the higher-level operations in the stated bounds is relatively straightforward. Note that (i)–(iv) may temporarily violate Invariants 3 and 4.

3.4.1 Splitting Chunks

Suppose we want to split the kth chunk of superchunk i into two pieces, both of which will (at least temporarily) stay within superchunk i.[7] We first zero-out all bits of $\mathrm{ChAdj}(i, \star)(k, \star)$

[7] Remember that 'k' refers to the actual position of the chunk within its superchunk whereas 'i' is an arbitrary ID that does not relate to its position within the list.

and $\mathrm{ChAdj}(\star, i)(\star, k)$ in $O(J)$ time. For each j we need to insert an all-zero row below row k in $\mathrm{ChAdj}(i, j)$ and an all-zero column after column k of $\mathrm{ChAdj}(j, i)$. This can be done in $O(1)$ time for each j, or $O(J)$ in total; see Lemma 5.

In $O(K)$ time we scan the edges incident to the new chunks k and $k+1$ and update the corresponding bits in $\mathrm{ChAdj}(i, \star)(k', \star)$ and $\mathrm{ChAdj}(\star, i)(\star, k')$, for $k' \in \{k, k+1\}$.

3.4.2 Merging Adjacent Chunks

In order to merge chunks k and $k+1$ of superchunk i we need to replace row k of $\mathrm{ChAdj}(i, j)$, for all j, with the bit-wise OR of rows k and $k+1$ of $\mathrm{ChAdj}(i, j)$, zero out row $k+1$, then scoot rows $k+2, \cdots$ back one row. A similar transformation is performed on columns k and $k+1$ of $\mathrm{ChAdj}(j, i)$, which takes $O(1)$ time per j, by Lemma 5. In total the time is $O(J)$, independent of K.

3.4.3 Splitting Superchunks

Suppose we want to split superchunk i after its kth chunk. We first call SCInsert(i), which allocates an empty superchunk with ID i' and inserts i' after i in its superchunk list in $\mathcal{S}$. In $O(J)$ time we transfer rows $k+1, \ldots, h-1$ from $\mathrm{ChAdj}(i, j)$ to $\mathrm{ChAdj}(i', j)$ and transfer columns $k+1, \ldots, h-1$ from $\mathrm{ChAdj}(j, i)$ to $\mathrm{ChAdj}(j, i')$. By Lemma 5 this takes $O(1)$ time per j.

At this point ChAdj is up-to-date but the list-sum data structure and $\{\mathrm{SupAdj}_j\}$ bit-vectors are not. We update $\mathrm{SupAdj}_i, \mathrm{SupAdj}_{i'}$ with calls to UpdateAdj(i, x) and UpdateAdj(i', x'). Using ChAdj, each bit of x and x' can be generated in constant time. This takes $O(J)$ time.

3.4.4 Merging Superchunks

Let the two adjacent superchunks have IDs i and i'. It is guaranteed that they will be merged only if they contain at most h chunks together. In $O(J)$ time we transfer the non-zero rows of $\mathrm{ChAdj}(i', j)$ to $\mathrm{ChAdj}(i, j)$ and transfer the non-zero columns of $\mathrm{ChAdj}(j, i')$ to $\mathrm{ChAdj}(j, i)$. A call to SCDelete(i') deletes superchunk i' from its list in $\mathcal{S}$ and retires ID i'. We then call UpdateAdj(i, x) with the new incidence vector x. In this case we can generate x in $O(J/w)$ time since it is merely the bit-wise OR of the old vectors SupAdj_i and $\mathrm{SupAdj}_{i'}$, with bit i' set to zero. Updating the list-sum data structure takes $O(J)$ time.

3.5 Joining and Splitting Lists

Once we have routines for splitting and merging adjacent (super)chunks, implementing Join and Split on lists in $\mathcal{L}$ is much easier. The goal is to restore Invariant 3 governing chunk masses and Invariant 4 on the number of chunks per superchunk.

3.5.1 Performing Join(L_0, L_1)

Write $L_0 = C_0, \ldots, C_{p-1}$ and $L_1 = D_0, \ldots, D_{q-1}$ as a list of chunks. If both L_0 and L_1 are not short then they have corresponding superchunk lists $S_0, S_1 \in \mathcal{S}$. Call SCJoin(S_0, S_1) to join S_0, S_1 in $\mathcal{S}$, in $O(J)$ time.

3.5.2 Performing $\textsc{Split}(x)$

Suppose x is contained in chunk C_l of $L = C_0 \cdots C_{l-1}C_lC_{l+1} \cdots C_{p-1}$. We split C_l into two chunks $C_l'C_l''$, and split the superchunk containing C_l along this line. Let S be the superchunk list corresponding to L and i be the ID of the superchunk ending at C_l'. We split S using a call to $\text{SC}\textsc{Split}(i)$, which corresponds to splitting L into $L_0 = C_0 \cdots C_{l-1}C_l'$ and $L_1 = C_l''C_{l+1} \cdots C_{p-1}$. At this point C_l' or C_l'' may violate Invariant 3 if $\text{mass}(C_l') < K$ or $\text{mass}(C_l'') < K$. Furthermore, Invariant 4 may be violated if the number of chunks in the superchunks containing C_l' and C_l'' is too small. We first correct Invariant 3 by possibly merging and resplitting $C_{l-1}C_l'$ and $C_l''C_{l+1}$ along new boundaries. If the superchunk containing C_l' has fewer than $h/2$ chunks, it and the superchunk to its left have strictly between $h/2$ and $3h/2$ chunks together, and so can be merged (and possibly resplit) into one or two superchunks satisfying Invariant 4. The same method can correct a violation of C_l'''s superchunk. This takes $O(K + J)$ time.

3.5.3 Performing $\textsc{ReplacementEdge}(L_0, L_1)$

The list-sum data structure makes implementing the $\textsc{ReplacementEdge}(L_0, L_1)$ operation easy. Let S_0 and S_1 be the superchunk lists corresponding to Euler tours L_0 and L_1. We compute the vectors $\alpha \leftarrow \textsc{AdjQuery}(S_0)$ and $\beta \leftarrow \textsc{MembQuery}(S_1)$ and their bit-wise AND $\alpha \wedge \beta$ with a linear scan of both vectors. If $\alpha \wedge \beta$ is the all-zero vector then there is no edge between L_0 and L_1. On the other hand, if $(\alpha \wedge \beta)(j) = 1$, then j must be the ID of a superchunk in S_1 that is incident to *some* superchunk in S_0. To determine *which* superchunk in S_0 we walk down from the root of S_0's list-sum tree to a leaf, say with ID i, in each step moving to a child z of the current node for which $\text{SupAdj}^z(j) = 1$. Once i and j are known we retrieve any 1-bit in the matrix $\text{ChAdj}(i, j)$, say at position (k, l), indicating that the kth chunk of superchunk i and the lth chunk of superchunk j are adjacent. We scan all its adjacent edges in $O(K)$ time and retrieve an edge joining L_0 and L_1. The total time is $O(J/w + \log J + K) = O(J/w + K)$.

3.5.4 Performing $\textsc{Insert}(u, v)$

If $\textsc{List}(u) \neq \textsc{List}(v)$, first perform $O(1)$ $\textsc{Split}$s and $\textsc{Join}$s to restore the Euler tour Invariant 2. Now u and v are in the same list in $\mathcal{L}$. Let i, j be the IDs of the superchunks containing the principal copies of u and v and let k, l be the positions of u and v's chunks within their respective superchunks. We set $\text{ChAdj}(i, j)(k, l) \leftarrow 1$. If $\text{ChAdj}(i, j)$ was formerly the all-zero matrix, we call $\textsc{UpdateAdj}(i, x)$ to update superchunk i's adjacency information with the correct vector x.[8] Inserting one edge changes the mass of the chunks containing u and v, which could violate Invariant 3. Invariants 3 and 4 are restored by splitting/merging $O(1)$ chunks and superchunks.

3.5.5 Performing $\textsc{Delete}(u, v)$

Compute i, j, k, l as defined above, in $O(1)$ time. After we delete (u, v) the correct value of the bit $\text{ChAdj}(i, j)(k, l)$ is uncertain. We scan chunk k of superchunk i in $O(K)$ time, looking for an edge connected to chunk l of superchunk j. If we do *not* find such an edge we set $\text{ChAdj}(i, j)(k, l) \leftarrow 0$, and if that makes $\text{ChAdj}(i, j) = 0$ (the all-zero matrix), we call

[8] Since x only differs from the former SupAdj_i at position $\text{SupAdj}_i(j)$, this update to the list-sum tree takes just $O(\log J)$ time since it only affects ancestors of leaves i and j.

UpdateAdj(i, x), where x is the new adjacency vector of superchunk i; it only differs from the former SupAdj_i at position j.

If (u, v) is a tree edge in $T = T_0 \cup \{(u, v)\} \cup T_1$ we perform Splits and Joins to replace $\mathrm{Euler}(T)$ with $\mathrm{Euler}(T_0), \mathrm{Euler}(T_1)$, which may violate Invariant 2 if there is a replacement edge between T_0 and T_1. We call ReplacementEdge$(\mathrm{Euler}(T_0), \mathrm{Euler}(T_1))$ to find a replacement edge. If one is found, say $(\hat{u}, \hat{v})$, we form $\mathrm{Euler}(T_0 \cup \{(\hat{u}, \hat{v})\} \cup T_1)$ with a constant number of Splits and Joins.

3.6 Running Time Analysis

Each operation ultimately involves splitting/merging $O(1)$ chunks, superchunks, and lists, which takes time $O(K + J + \log J \cdot J/w) = O(K + J) = O(K + \hat{m}/(K\sqrt{w}))$. We balance the terms by setting $K = \sqrt{\frac{\hat{m}}{\sqrt{w}}}$ so the running time is $O(K)$.

By the sparsification transformation of Eppstein, Galil, Italiano, and Nissenzweig [7] this implies an update time of $O\left(\frac{\sqrt{n}}{w^{1/4}}\right)$. Each instance of dynamic connectivity created by [7] has a fixed set of vertices, say of size $\hat{n}$, and a fixed upper bound $\hat{m} = O(\hat{n})$ on the number of edges.

4 Speeding Up the Algorithm

Observe that there are $\Theta((\hat{m}/(Kh))^2)$ matrices $(\mathrm{ChAdj}(i, j))$ but only $\hat{m}$ edges, so for $K = \sqrt{\hat{m}/h}$, the average $h \times h$ matrix has $O(h)$ 1s. Thus, storing each such matrix verbatim, using h^2 bits, is information theoretically inefficient on average. By storing only the locations of the 1s in each matrix we can represent each matrix in $O(h \log h)$ bits on average and thereby hope to solve dynamic connectivity faster with a larger 'h' parameter.

4.1 The Encoding

In this encoding we index rows and columns by indices in $\{1, \ldots, h\}$ rather than $[h]$. Let $m_{i,j} = m_{j,i}$ be the number of 1s in $\mathrm{ChAdj}(i, j)$. We encode $\mathrm{ChAdj}(i, j)$ by listing its 1 positions in $O(m_{i,j} \log h/w)$ lightly packed words. Each word is partitioned into *fields* of $1 + 2\lceil\log(h + 1)\rceil$ bits: each field consists of a control bit (normally 0), a row index, and a column index. Each word is between half-full and full, the fields in use being packed contiguously in the word. This invariant allows us to insert a new field after a given field in $O(1)$ time. We list the 1s of *either* $\mathrm{ChAdj}(i, j)$ *or* $\mathrm{ChAdj}(j, i) = \mathrm{ChAdj}(i, j)^\top$ in row-major order, with a bit indicating which of the two representations is used.

4.2 Fast Operations

Given $\mathrm{ChAdj}(i, j)$ in row-major order, we can determine if $\mathrm{ChAdj}(i, j)(k, l) = 1$ in $O(\log h)$ time by doing a binary search to find the correct word in the list, then a binary search within the word to find an entry $\langle k, l\rangle$, if any.[9]

[9] The time for a search can actually be improved to $O(\log((m_{i,j} \log h)/w))$, which is faster when $\mathrm{ChAdj}(i, j)$ has average density ($m_{i,j}$ is close to h) but is still $O(\log h)$ in the worst case. We still do a binary search over the first field in each word to determine which word (if any) has a field containing $\langle k, l\rangle$: the binary encoding of (k, l). This takes $O(\log((m_{i,j} \log h)/w))$ time since there are $\Theta((m_{i,j} \log h)/w)$ lightly packed words. If we add $2^{2\lceil\log(h+1)\rceil} - \langle k, l\rangle$ to each field in the word, the control bits for all fields that are equal to or greater than $\langle k, l\rangle$ will be flipped to 1. Similarly, if we set all control bits to 1

In the same time bound we can also identify the positions of the first and last 1s in row k. Thus, we can perform the following operations on $\mathrm{ChAdj}(i,j)$ in $O((m_{i,j}\log h)/w)$ time: setting a row to zero, incrementing/decrementing the row-index of some interval of rows, or copying an interval of rows.

The operations sketched above are only efficient if $\mathrm{ChAdj}(i,j)$ is in row-major order. If we have $\mathrm{ChAdj}(i,j)^\top$ in row-major order we can effect a transpose by (1) swapping the row and column indices in each field using masks and shifts, and (2) sorting the fields. In general, sorting x words of $O(w/\log h)$ fields takes $O(x(\log^2(w/\log h) + \log x \log(w/\log h)))$ time using Albers and Hagerup's implementation [2] of Batcher's bitonic mergesort [4].[10] We sort each word in $O(\log^2(w/\log h))$ time, resulting in x sorted lists, then iteratively merge the two shortest lists until one list remains. Merging two lists containing y words takes $O(y\log(w/\log h))$ time: we can merge the next $w/\log h$ fields of each list in $O(\log(w/\log h))$ time [2] and output at least $w/\log h$ items to the merged list.

Alternatively, if $w = \log n$ we can sort and merge lists of $\epsilon\log n/\log h$ fields in unit time using table lookup to precomputed tables of size $O(n^\epsilon)$. In this case sorting x packed words takes $O(x\log x)$ time.

4.3 Splitting and Joining

The cost of splitting and joining (super)chunks is now slightly more expensive. When handling superchunk i (or any chunk within it) we first put each $\mathrm{ChAdj}(i,j)$ in row-major order, in $\sum_{j=1}^{J} O(\lceil \frac{m_{i,j}\log h}{w}\rceil \log^2 h) = O(J\log^2 h + (Kh/w)\log^3 h)$ since, by Invariants 3 and 4, $\sum_j m_{i,j} = O(Kh)$. Once the relevant superchunks are in the correct format, splitting or joining $O(1)$ (super)chunks takes $O(K\log h + J + (Kh/w)\log h)$ time. Since $J = O(\hat{m}/(Kh))$, the overall update time is

$$O\left(K\log h + \frac{\hat{m}\log^2 h}{Kh} + \frac{Kh\log^3 h}{w}\right).$$

Setting $h = w$ and $K = \sqrt{\frac{\hat{m}}{w\log w}}$, the overall time is $O(\sqrt{\frac{\hat{m}\log^5 w}{w}})$. When $w = O(\log n)$ the cost of taking the transpose is cheaper since sorting and merging a packed word takes unit time via table lookup. Setting $h = \log n$, the total time is

$$O\left(K\log\log n + \frac{\hat{m}}{K\log n} + K(\log\log n)^2\right),$$

which is $O(\sqrt{\frac{\hat{m}(\log\log n)^2}{\log n}})$ when $K = \sqrt{\frac{\hat{m}}{\log n(\log\log n)^2}}$.

and subtract $\langle k,l\rangle + 1$ from each field, the control bits of fields that are equal to or less than $\langle k,l\rangle$ will be flipped to 0. Thus, we can single out the control bit for an occurrence of $\langle k,l\rangle$ (if any) with $O(1)$ bit-wise operations. If $\langle k,l\rangle$ is not present, the control bits reveal the field in the word after which it could be inserted, if we need to set $\mathrm{ChAdj}(i,j)(k,l) \leftarrow 1$.

[10] Albers and Hagerup also require that the fields to be sorted begin with control bits.

5 Loose Ends

5.1 Removing the Bounded Degree Assumption

Invariants 3 and 4 imply that there are $J = \Theta(\hat{m}/(Kh))$ superchunks with non-$\perp$ IDs. However, Invariant 3 cannot be satisfied (as stated) unless the maximum degree is bounded by $O(K)$. One way to guarantee this is to physically split up high degree vertices, replacing each v with a cycle on new vertices $v_1, \ldots, v_{\lceil \deg(v)/\Theta(K) \rceil}$, each of which is responsible for $\Theta(K)$ of v's edges. This is the method used by Frederickson [10], who actually demanded that the maximum degree be 3 at all times!

This vertex-splitting can be effectively simulated in our algorithm as follows. If $\deg(v) \geq K/2$, replace the principal copy of v in its Euler tour with an interval of artificial principal vertices $v_1, \ldots, v_{\lceil \deg(v)/(K/2) \rceil}$, each of which is responsible for between $K/2$ and K of v's edges. Invariant 3 is therefore maintained w.r.t. this modified tour. To keep the mass of artificial vertices between $K/2$ and K, each edge insertion/deletion may require splitting an artificial vertex or merging two consecutive artificial vertices. When the Euler tour changes we always preserve the invariant that v's artificial vertices form a contiguous interval in the tour.

5.2 Dealing with Short Lists

Until now we have assumed for simplicity that all superchunks have proper IDs in $[J]$. It is important that we *not* give out IDs to short lists (consisting of less than $h/2$ chunks) because the running time of the algorithm is linear in the maximum ID J. The modifications needed to deal with short lists are tedious but minor.

Consider an Insert(u, v) operation where u and v are in lists L_0, L_1 and L_1 is a short list consisting of one superchunk S with $\mathrm{ID}(S) = \perp$. If L_0 is not short (or if it is short but the combined list L_0L_1 will not be short) then we retrieve an unused ID, say i, set $\mathrm{ID}(S) \leftarrow i$, set $\mathrm{ChAdj}(i, i) \leftarrow \mathrm{ChAdj}_S$, and destroy ChAdj_S. By Invariant 2, S was not incident to any other superchunk, so $\mathrm{ChAdj}(i, j) = 0$ (the all-zero matrix) for all $j \neq i$. At this point S violates Invariant 4 (it is too small), so we need to merge it with the last superchunk in L_0 and resplit it along a different chunk boundary, in $O(J)$ time.

The modifications to Delete(u, v) are analogous. If we delete a tree edge (u, v), splitting its component into T_0 and T_1 having associated Euler tours L_0 and L_1, and ReplacementEdge(u, v) fails to find an edge joining L_0 and L_1, we need to check whether L_0 (and L_1) are short. If so let S be the superchunk in L_0. We allocate and set $\mathrm{ChAdj}_S \leftarrow \mathrm{ChAdj}(\mathrm{ID}(S), \mathrm{ID}(S))$, then set $\mathrm{ChAdj}(\mathrm{ID}(S), \mathrm{ID}(S)) \leftarrow 0$ and finally retire $\mathrm{ID}(S)$.

The implementation of ReplacementEdge(L_0, L_1) is different if L_0 and L_1 were originally in a short list $L = \mathrm{Euler}(T)$ before a tree edge in T was deleted. Suppose L originally had one superchunk S, whose chunk adjacency was stored in ChAdj_S. After $O(1)$ splits and joins, both L_0's chunks and L_1's chunks occupy $O(1)$ intervals of the rows and columns of ChAdj_S. Of course ChAdj_S is represented as a list of its 1 positions in row-major order, so we can isolate the correct intervals of rows and columns in $O(h^2 \log^3 h/w)$ time. If there is any 1 there, say at location $\mathrm{ChAdj}_S(k, l)$, then we know that there is an edge between L_0 and L_1, and can find it in $O(K)$ time by examining chunks k and l. The permutation of rows/columns in ChAdj_S must be updated to reflect any splits and joins that take place, and if no replacement edge is discovered, ChAdj_S must be split into two lists representing matrices ChAdj_{S_0} and ChAdj_{S_1}, to be identified with the single superchunks S_0 and S_1 in L_0 and L_1, respectively.

References

1 U. A. Acar, G. E. Blelloch, R. Harper, J. L. Vittes, and S. L. M. Woo. Dynamizing static algorithms, with applications to dynamic trees and history independence. In *Proceedings 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 531–540, 2004.

2 S. Albers and T. Hagerup. Improved parallel integer sorting without concurrent writing. *Information and Computation*, 136(1):25–51, 1997. doi:10.1006/inco.1997.2632.

3 S. Alstrup, J. Holm, K. de Lichtenberg, and M. Thorup. Maintaining information in fully dynamic trees with top trees. *ACM Trans. on Algorithms*, 1(2):243–264, 2005. doi:10.1145/1103963.1103966.

4 T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, 3rd ed.* MIT Press, 2009.

5 E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

6 D. Eppstein, Z. Galil, G. F. Italiano, and A. Nissenzweig. Sparsification–a technique for speeding up dynamic graph algorithms. In *Proceedings 33rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 60–69, 1992.

7 D. Eppstein, Z. Galil, G. F. Italiano, and A. Nissenzweig. Sparsification – a technique for speeding up dynamic graph algorithms. *J. ACM*, 44(5):669–696, 1997.

8 D. Eppstein, G. F. Italiano, R. Tamassia, R. E. Tarjan, J. Westbrook, and M. Yung. Maintenance of a minimum spanning forest in a dynamic plane graph. *J. Algor.*, 13(1):33–54, 1992.

9 D. Eppstein, G. F. Italiano, R. Tamassia, R. E. Tarjan, J. Westbrook, and M. Yung. Corrigendum: Maintenance of a minimum spanning forest in a dynamic plane graph. *J. Algor.*, 15(1):173, 1993.

10 G. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM J. Comput.*, 14(4):781–798, 1985.

11 M. L. Fredman and M. Saks. The cell probe complexity of dynamic data structures. In *Proceedings 21st Annual ACM Symposium on Theory of Computing (STOC)*, pages 345–354, 1989.

12 M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987.

13 D. Gibb, B. M. Kapron, V. King, and N. Thorn. Dynamic graph connectivity with improved worst case update time and sublinear space. *CoRR*, abs/1509.06464, 2015.

14 M. R. Henzinger and M. L. Fredman. Lower bounds for fully dynamic connectivity problems in graphs. *Algorithmica*, 22(3):351–362, 1998.

15 M. R. Henzinger and V. King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *J. ACM*, 46(4):502–516, 1999.

16 M. R. Henzinger and M. Thorup. Sampling to provide or to bound: With applications to fully dynamic graph algorithms. *J. Random Structures and Algs.*, 11(4):369–379, 1997.

17 J. Holm, K. de Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, 2001.

18 B. M. Kapron, V. King, and B. Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1131–1142, 2013.

19 P. B. Miltersen, S. Subramanian, J. S. Vitter, and R. Tamassia. Complexity models for incremental computation. *Theoretical Computer Science*, 130(1):203–236, 1994.

20 M. Pǎtraşcu and E. Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM J. Comput.*, 35(4):932–963, 2006.

21 M. Patrascu and M. Thorup. Don't rush into a union: take time to find your roots. In *Proceedings of the 43rd ACM Symposium on Theory of Computing (STOC)*, pages 559–568, 2011. Technical report available as arXiv:1102.1783.

22 D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983.

23 R. E. Tarjan and R. F. Werneck. Self-adjusting top trees. In *Proceedings 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 813–822, 2005.

24 M. Thorup. Near-optimal fully-dynamic graph connectivity. In *Proceedings 32nd ACM Symposium on Theory of Computing (STOC)*, pages 343–350, 2000.

25 C. Wulff-Nilsen. Faster deterministic fully-dynamic graph connectivity. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1757–1769, 2013.

Think Eternally: Improved Algorithms for the Temp Secretary Problem and Extensions*

Thomas Kesselheim†1 and Andreas Tönnis‡2

1 Max-Planck-Insitut für Informatik and Saarland University, Saarbrücken, Germany
thomas.kesselheim@mpi-inf.mpg.de

2 Department of Computer Science, RWTH Aachen University, Germany
toennis@cs.rwth-aachen.de

Abstract

The *Temp Secretary Problem* was recently introduced by Fiat et al. [11]. It is a generalization of the Secretary Problem, in which commitments are temporary for a fixed duration. We present a simple online algorithm with improved performance guarantees for cases already considered by Fiat et al. and give competitive ratios for new generalizations of the problem. In the classical setting, where candidates have identical contract durations $\gamma \ll 1$ and we are allowed to hire up to B candidates simultaneously, our algorithm is $(1/2 - O(\sqrt{\gamma}))$-competitive. For large B, the bound improves to $1 - O\left(1/\sqrt{B}\right) - O(\sqrt{\gamma})$.

Furthermore we generalize the problem from cardinality constraints towards general packing constraints. We achieve a competitive ratio of $1 - O\left(\sqrt{(1+\log d + \log B)/B}\right) - O(\sqrt{\gamma})$, where d is the sparsity of the constraint matrix and B is generalized to the capacity ratio of linear constraints. Additionally we extend the problem towards arbitrary hiring durations.

Our algorithmic approach is a relaxation that aggregates all temporal constraints into a non-temporal constraint. Then we apply a linear scaling algorithm that, on every arrival, computes a tentative solution on the input that is known up to this point. This tentative solution uses the non-temporal, relaxed constraints scaled down linearly by the amount of time that has already passed.

1998 ACM Subject Classification F.1.2 Models of Computation/Online computation

Keywords and phrases Secretary Problem, Online Algorithms, Scheduling Problems

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.54

1 Introduction

Online resource allocation problems have a notion of time: Choices have to be made at some point in time without knowing the future input. Each decision may make a future one infeasible. The standard example of such a setting is the secretary problem where candidates of different value arrive over time. After each arrival, the algorithm has to decide whether to permanently accept or reject this candidate. Every decision is final. That is, once rejected a candidate will never come back again. Once a candidate is accepted, no other candidate can be accepted anymore.

* A full version of the paper is available at http://arxiv.org/abs/1606.06926.
† Supported in part by the DFG through Cluster of Excellence MMCI.
‡ Supported by the DFG GRK/1298 "AlgoSyn".

In many practical applications, however, commitments are not eternal but affect only a finite time horizon. They may limit options for the upcoming days but not for the rest of the year or even longer. Nevertheless, even with such an assumption, traditional worst-case competitive analysis is typically too strong a benchmark. It is trivial to see that for the respective version of the secretary problem no algorithm achieves a bounded competitive ratio.

Therefore, we consider a partly stochastic model introduced by Fiat et al. [11]. First an adversary chooses which items will arrive. However, it does not determine the arrival times, which are instead drawn from a probability distribution, typically the uniform distribution on $[0, 1]$. In more detail, in the *temp secretary problem*, an adversary defines values of items $v_1, \dots, v_n$. Afterwards, arrival times τ_j are drawn independently uniformly from $[0, 1]$. As time proceeds, the values and arrival times are revealed to the algorithm. Upon each arrival, the algorithm has to decide whether to accept or to reject the respective item. Each item is accepted for a duration of γ, which is assumed to be much smaller than 1. At any point in time t at most B items may overlap, that is, during time $t - \gamma$ and t at most B items may be accepted.

The objective is to maximize $\sum_{j \in \mathrm{ALG}} v_j$, where ALG $\subseteq [n]$ denotes the selection by the algorithm. By OPT we denote the optimal selection OPT $\subseteq [n]$, which maximizes $\sum_{j \in \mathrm{OPT}} v_j$. As the arrival times $\tau_1, \dots, \tau_n$ are random, both ALG and OPT are random variables. We evaluate the performance of an algorithm by its competitive ratio, defined as $\mathbf{E}\left[\sum_{j \in \mathrm{ALG}} v_j\right] / \mathbf{E}\left[\sum_{j \in \mathrm{OPT}} v_j\right]$.

1.1 Our Contribution

We introduce a new algorithmic approach to online packing problems with temporal constraints. As key idea we consider a relaxation to OPT by removing the temporal constraints and exchanging them with global ones. In the special case of the temp secretary problem, we exploit that for every realization of the arrival dates $\tau_1, \dots, \tau_n$ the optimal offline solution OPT never contains more that $B\lceil 1/\gamma \rceil$ elements. Therefore, we exchange the constraints by only requiring $B\lceil 1/\gamma \rceil$ items to be picked throughout the process. An online solution to this relaxation can be found using algorithms for online linear packing problems. It then remains to derive a solution to the original constraints.

For the temp secretary problem, this approach allows us to derive a light-weight, easy to state algorithm. We show it to be $\frac{1}{2}\left(1 - O(\sqrt{\gamma})\right)$-competitive for all values of B. Furthermore, for large values of B, a different analysis shows a better competitive ratio of $1 - O\left(1/\sqrt{B}\right) - O(\sqrt{\gamma})$. The previous best results for this setting were $\frac{1}{2}\left(1 - O\left(\sqrt{\gamma \ln(1/\gamma)}\right)\right)$ for $B = 1$ and $1 - O\left(\sqrt{(\ln B)/B}\right) - O\left(\sqrt{\gamma \ln(1/\gamma)}\right)$ for large values of B, both by Fiat et al. [11]. Note that $1/2$ is known to be an asymptotic upper bound to the competitive ratio for $B = 1$ [11].

We also generalize the cardinality constraint in the temp secretary problem to arbitrary linear constraints. This enables us to capture more general combinatorial problems, like multiple knapsack constraints that have to be fulfilled simultaneously. For example, we could model scenarios in which the algorithm has to select production orders online in such a way that none of the involved machines is overloaded. Our algorithm is $1 - O\left(\sqrt{(1 + \log d + \log B)/B}\right) - O(\sqrt{\gamma})$-competitive, where d denotes the maximum number of constraints a single item is contained in. By B we denote the *capacity ratio*, which is defined to be the minimum ratio of a constraint's capacity and the usage of a single item. For non-timed constraints, there are lower bounds in the order of $1 - O\left(\sqrt{\log m/B}\right)$, where m is the number of constraints and $d = m$ [2, 6].

Our algorithm also has a natural generalization to settings with items of different lengths. For the temp secretary problem, we show a competitive ratio of $\frac{1}{4} - \Theta\left(\sqrt{\gamma}\right)$.

The main technical contribution are bounds on the probability that tentative selections made by the algorithm are actually feasible. In related work, it is usually enough to pretend all previous tentative choices were actually feasible. As these can be considered independent, a concentration bound can be applied. These techniques are apparently not strong enough here and we have to bound the actual commitments. We do so by analyzing coupled random variables that provide an upper bound on the random process. For the case of large B, this analysis is based on a symmetric random walk, representing arrivals and departures of items.

1.2 Related Work

Secretary problems have gained a lot of attention over the last decade, even though the most famous variant was already introduced and solved in the 1960s [12, 21, 7].

The most famous combinatorial generalization is the matroid secretary problem, introduced by Babaioff et al [4]. As of now, the big question of whether there is a constant competitive algorithm for the matroid secretary problem is still open. The best known algorithms for the problem are $O(\log\log\rho)$-competitive [10, 20]. Constant competitive algorithms are known for most special cases, e.g. there is a $1/2e$-competitive algorithm for graphical matroids [19], a $1/9.6$-competitive algorithm for laminar matroids [23] and there is an optimal $1/e$-competitive algorithm for transversal matroids [16]. For k-uniform matroids, the problem is also known as multiple-choice secretary problem and was solved by Kleinberg, who gave a $(1 - O(1/\sqrt{k}))$-competitive algorithm and showed that this is optimal [18].

Furthermore, online models with random arrival order have been used for online packing problems. The knapsack secretary problem was introduced by Babaioff et al. [3] and the currently best known competitive ratio is $1/8.1$ [17]. This problem was generalized towards general packing linear programs with special attention on the case with large capacities. There are several known algorithms [1, 13, 17] that feature a competitive ratio of $1 - O\left(\sqrt{\log m/B}\right)$, where m is the total number of constraints and B is a lower bound on the capacities of the constraints. These results match the lower bound by Agrawal et al.[2] and Devanur et al. [6] for the random order and i.i.d. model respectively. Note that the result in [17] is stronger in case of sparse matrices: If the maximal number of non-zero entries in any column is bounded by d, the guarantee only depends on d rather than m.

Another important way of generalizing the secretary problem is the submodular secretary problem introduced by Bateni et al. [5]. The problem generalizes the multiple-choice secretary problem towards submodular objective functions. The currently best known competitive ratio is $\frac{e-1}{e^2+e}$ by Feldmann et al. [9]. For submodular, transversal matroids the best known algorithm is $1/95$-competitive [23] and for linear packing constraints the best algorithm is known to be $\Omega(1/m)$-competitive [5].

The temp secretary problem that we consider and generalize in this paper was introduced in 2015 by Fiat et al. [11]. It introduces temporal constraints to the field of online algorithms with random order in a way that had only been considered before in the worst-case model for online interval scheduling [22, 25]. Fiat et al. give an algorithm that is inspired by Kleinbergs algorithm for the multiple-choice secretary problem. Their algorithm iteratively refines the sample $\log n$ times, while our algorithm updates the sample in every round. Both algorithm are closely related, but the one presented here can be described much more compact and allows for a more simple analysis.

Fiat et al. achieve a competitive ratio of $1/(1+k\gamma)\left(1 - 5/\sqrt{k} - 7.4\sqrt{\gamma\ln(1/\gamma)}\right)$ for the case where at most one candidate can be hired simultaneously and the sum of hires cannot

Algorithm 1: Scaling Algorithm for length γ and capacity B

```
for every arriving item $j$ do
    Set $t := \tau_j$;                                          // arrival time of $j$
    Let $S^{(t)}$ be the $\lfloor tB/\gamma \rfloor$ highest-valued items $j'$ with $\tau_{j'} \le t$;
    if $j \in S^{(t)}$ then                                     // if among best items
        if $S \cup \{j\}$ is a feasible schedule then           // and if feasible
            Set $S := S \cup \{j\}$;                            // then select $j$
```

exceed the budget k. To compare this result to ours, consider the unconstrained budget case $k = 1/\gamma$. In this case, they achieve a competitive ratio of $\frac{1}{2}\left(1 - O\left(\sqrt{\gamma \ln(1/\gamma)}\right)\right)$. Additionally, they show a lower bound $\frac{1+\gamma}{2}$ for this case, thus both algorithms, ours and theirs, are asymptotically tight for $\gamma \to 0$. For up to B concurrent hires, their algorithm is $1 - \Theta\left(\sqrt{(\ln B)/B}\right) - \Theta\left(\sqrt{\gamma \ln(1/\gamma)}\right)$-competitive. Additionally, they describe a black-box procedure that transforms any algorithm for a combinatorial secretary problem into an algorithm for the respective combinatorial temp secretary problem. This transformation loses a factor of $1/2$ in the competitive ratio, but also works for general arrival distributions as long as all items have an identical duration.

2 The Temp Secretary Problem

As our first result, we present a simplified and improved algorithm for the temp secretary problem. Here, an adversary chooses a value v_j for each of the n items and after values have been determined arrival times $\tau_1, \ldots, \tau_n$ are drawn independently uniformly at random from $[0, 1]$. Each item when selected stays active for γ time. At any point in time, at most B elements may be active simultaneously.

The optimal selection $\mathrm{OPT} \subseteq [n]$ is a random variable that depends on the arrival times. However, pointwise we have $|\mathrm{OPT}| \le B\lceil 1/\gamma \rceil$ for any realization of the arrival times $\tau_1, \ldots, \tau_n$. Therefore, the expected value of OPT can be upper-bounded by the value of the $B\lceil 1/\gamma \rceil$ highest-valued elements, which we denote by $\mathrm{OPT}^* \subseteq [n]$.

Algorithm 1 is inspired by online approximation algorithms for OPT^*, particularly [17]. If item j arrives at time t, then we determine whether it is among the $\lfloor tB/\gamma \rfloor$ highest-valued items seen so far, called $S^{(t)}$. In this case, we call it tentatively selected. If it is also feasible to accept j, we do so. Otherwise, we reject j.

Note that in expectation at time t we have seen a t fraction of OPT^*, so approximately tB/γ items from OPT^*. The set $S^{(t)}$ approximates this set by including the best $\lfloor tB/\gamma \rfloor$ up to this point.

We give two performance bounds for this algorithm. First, in Section 2.2, we show that it is $\frac{1}{2}\left(1 - O(\sqrt{\gamma})\right)$-competitive for all values of B. Afterwards, in Section 2.3, we perform a different analysis for large values of B giving a competitive ratio of $1 - O\left(1/\sqrt{B}\right) - O(\sqrt{\gamma})$.

2.1 Analysis Preliminaries

The analyses for both cases follow a similar pattern. First, we analyze the expected value of the set $S^{(t)}$ and thereby the expected value of the tentative selection. Then, we bound the probability that such a tentative selection is feasible.

For analysis purposes, we discretize time into N uniform intervals of length $1/N$, which we call rounds. If $N \gg n$, then the probability that two items fall into the same interval is negligible. Also, if N is large enough, we can effectively assume that all items arrive at times which are multiples of $1/N$ because the value of $\lfloor tB/\gamma \rfloor$ stays constant in almost all intervals. From time to time, it will be helpful to fill up rounds in which no actual item arrives with dummy items that do not have value but also do not use space. The order of these items and dummy items is then a uniformly drawn permutation. Furthermore, to avoid cumbersome notation, we assume that γN and $\sqrt{\gamma} N$ are integer. We overload notation and write $S^{(\ell)}$ instead of $S^{(\ell/N)}$.

To discuss the probability of feasibility, we introduce 0/1 random variables $(C_\ell)_{\ell \in [N]}$ and $(F_\ell)_{\ell \in [N]}$. We set $C_\ell = 1$ if and only if a tentative selection is made in round ℓ. Furthermore, let $F_\ell = 1$ for every round ℓ in which it would be feasible to actually select an item. Finally, let V_ℓ denote the value of the item tentatively selected in round ℓ if any, otherwise set $V_\ell = 0$. So formally $V_\ell = v_j C_\ell$ if item i arrives in round ℓ. The value achieved by the algorithm is given as $\sum_{j \in \mathrm{ALG}} v_j = \sum_{\ell=1}^N V_\ell F_\ell = \sum_{\ell=1}^N V_\ell C_\ell F_\ell$.

Observe that the value of a single random variable C_ℓ is already fully determined by the set $S^{(\ell)}$ and which of these items arrives in round ℓ. Neither the mutual order in rounds $1, \ldots, \ell - 1$ nor in $\ell + 1, \ldots, N$ matters. Furthermore, conditioned on any set $S^{(\ell)}$ and any order in rounds $\ell + 1, \ldots, N$, the probability of $C_\ell = 1$ is at most $\frac{|S^{(\ell)}|}{\ell} \leq \frac{B}{\gamma N}$. As a consequence, for every sequence of values $a_{\ell'} \in \{0, 1\}$ for $\ell < \ell' \leq N$, we have

$$\mathbf{Pr}\left[C_\ell = 1 \mid C_{\ell'} = a_{\ell'} \forall \ell < \ell' \leq N\right] \leq \frac{B}{\gamma N} \quad .$$

Note that there are still complicated dependencies among these random variables. For example, at most n of them can be 1. Therefore, based on the above observation, we define coupled random variables that dominate the actual ones but are easier to deal with. We introduce random variables $(\tilde{C}_\ell)_{\ell \in [N]}$ such that pointwise $\tilde{C}_\ell \geq C_\ell$ for which the above relation holds with equality. To define these formally, we iterate over the rounds from large to small index. Conditioned on any value of the variables $\tilde{C}_{\ell+1} = a_{\ell+1}, \ldots, \tilde{C}_N = a_N$, the probability $p = \mathbf{Pr}\left[C_\ell = 1 \;\middle|\; \tilde{C}_{\ell'} = a_{\ell'} \forall \ell < \ell' \leq N\right]$ is always at most $\frac{B}{\gamma N}$. Now let $\tilde{C}_\ell = 1$ whenever $C_\ell = 1$ and additionally $\tilde{C}_\ell = 1$ with probability $\frac{\frac{B}{\gamma N} - p}{1 - p}$ if $C_\ell = 0$. This guarantees

$$\mathbf{Pr}\left[\tilde{C}_\ell = 1 \;\middle|\; \tilde{C}_{\ell'} = a_{\ell'} \forall \ell < \ell' \leq N\right] = \frac{B}{\gamma N} \quad ,$$

and therefore, by induction on all subsets of $[N]$, the random variables $\tilde{C}_1, \ldots, \tilde{C}_N$ are independent and identically distributed. Note that $\tilde{C}_\ell = 1$ whenever $C_\ell = 1$. So therefore also $\sum_{j \in \mathrm{ALG}} v_j = \sum_{\ell=1}^n V_\ell \tilde{C}_\ell F_\ell$.

Next, we can bound the expected value of the items contained in the set $S^{(\ell)}$.

▶ **Lemma 1.** *For* $\ell \geq 2\sqrt{\frac{\gamma}{B}} N$

$$\mathbf{E}\left[\sum_{j \in S^{(\ell)}} v_j\right] \geq \left(1 - 9\sqrt{\frac{1}{\frac{\ell}{N} \cdot \frac{B}{\gamma}}}\right) \frac{1 - \frac{1}{2}\sqrt{\frac{\gamma}{B}}}{1 + \gamma} \frac{\ell}{N} \sum_{j \in \mathrm{OPT}^*} v_j \quad .$$

The general idea is as follows. In round ℓ, we have seen an $\frac{\ell}{N}$-fraction of OPT^* in expectation, so ignoring rounding these are $\frac{\ell}{N} \frac{B}{\gamma}$ items in expectation. This approximately matches the size of $S^{(\ell)}$. The set $S^{(\ell)}$ has a slightly smaller value due to variance and rounding. The first two factors compensate these effects.

For the proof, we show that the non-temporal relaxation described here corresponds to a packing linear program and then we apply a result in [17]. For the detailed proof, please refer to the full version.

2.2 General Analysis

We are now ready to analyze the algorithm.

▶ **Theorem 2.** *Algorithm 1 is at least* $\frac{1}{2}\left(1 - \frac{7}{2}\sqrt{\gamma} - \frac{37}{2}\sqrt{\frac{\gamma}{B}} - \gamma\right)$*-competitive for the temp secretary problem with duration* γ *for all items.*

The key ingredient to the analysis is a bound on the probability that a tentative selection is feasible.

▶ **Lemma 3.** *For all* L*, we have*

$$\mathbf{E}\left[\sum_{\ell=L}^{L+\sqrt{\gamma}N-1} \tilde{C}_\ell F_\ell\right] \geq \left(\frac{1}{2} - \sqrt{\gamma} - \frac{1}{4\sqrt{\gamma}N}\right)\mathbf{E}\left[\sum_{\ell=L}^{L+\sqrt{\gamma}N-1} \tilde{C}_\ell\right] \ .$$

Proof. The crux when proving this lemma is that that F_ℓ depends on $\tilde{C}_\ell$ in a non-trivial way. Therefore, we instead introduce variables $\tilde{F}_\ell$ defined as follows. We set $\tilde{F}_\ell = 1$ for $\ell < L$ and $\tilde{F}_\ell = \max\{0, 1 - \frac{1}{B}\sum_{i=1}^{\gamma N} \tilde{C}_{\ell-i}\tilde{F}_{\ell-i}\}$ for $\ell \geq L$. The motivation behind this definition is that $F_\ell = 1$ if and only if it is feasible to select an item in round ℓ. So therefore, $F_\ell = 1$ if and only if $\sum_{i=1}^{\gamma N} C_{\ell-i}F_{\ell-i} < B$, implying $F_\ell \geq \max\{0, 1 - \frac{1}{B}\sum_{i=1}^{\gamma N} C_{\ell-i}F_{\ell-i}\}$. The definition of $\tilde{F}_\ell$ captures this last bound in a pessimistic way. Note that due to the independence of $(\tilde{C}_\ell)_{\ell\in[N]}$, $\tilde{C}_\ell$ and $\tilde{F}_\ell$ are now independent.

We first show that pointwise $\sum_{\ell=L}^{L+k} \tilde{C}_\ell F_\ell \geq \sum_{\ell=L}^{L+k} \tilde{C}_\ell \tilde{F}_\ell$ for all $k \in \mathbb{Z}$ by induction on k. Observe that the statement is trivial for $k < 0$ because then both sums are empty. So, let us consider $k \geq 0$ for the induction step. If $\tilde{C}_{L+k} = 0$ or $\tilde{F}_{L+k} = 0$, then also the statement follows trivially from the induction hypothesis. The only interesting case is $\tilde{C}_{L+k} = 1$ and $\tilde{F}_{L+k} > 0$. In this case, we get

$$\begin{aligned}\sum_{\ell=L}^{L+k} \tilde{C}_\ell\tilde{F}_\ell &= \tilde{F}_{L+k} + \sum_{\ell=L}^{L+k-1} \tilde{C}_\ell\tilde{F}_\ell = 1 - \frac{1}{B}\sum_{i=1}^{\gamma N} \tilde{C}_{L+k-i}\tilde{F}_{L+k-i} + \sum_{\ell=L}^{L+k-1} \tilde{C}_\ell\tilde{F}_\ell \\ &= 1 - \frac{1}{B}\sum_{\ell=L+k-\gamma N}^{L-1} \tilde{C}_\ell\tilde{F}_\ell + \left(1 - \frac{1}{B}\right)\sum_{\ell=L}^{L+k-1} \tilde{C}_\ell\tilde{F}_\ell + \frac{1}{B}\sum_{\ell=L}^{L+k-\gamma N-1} \tilde{C}_\ell\tilde{F}_\ell \ .\end{aligned}$$

At this point, we can apply the induction hypothesis, which states that in the second and third sum we can replace all occurrences of $\tilde{F}_\ell$ by F_ℓ to get a lower bound. Furthermore, $1 = \tilde{F}_\ell \geq F_\ell$ for $\ell \leq L - 1$. So, we can do the same in the first sum. Therefore

$$\begin{aligned}\sum_{\ell=L}^{L+k} \tilde{C}_\ell\tilde{F}_\ell &\leq 1 - \frac{1}{B}\sum_{\ell=L+k-\gamma N}^{L-1} \tilde{C}_\ell F_\ell + \left(1 - \frac{1}{B}\right)\sum_{\ell=L}^{L+k-1} \tilde{C}_\ell F_\ell + \frac{1}{B}\sum_{\ell=L}^{L+k-\gamma N-1} \tilde{C}_\ell F_\ell \\ &= 1 - \frac{1}{B}\sum_{i=1}^{\gamma N} \tilde{C}_{L+k-i}F_{L+k-i} + \sum_{\ell=L}^{L+k-1} \tilde{C}_\ell F_\ell \ .\end{aligned}$$

Now, we use that $F_{L+k} = 1$ if and only if less than B items have been selected in rounds $L + k - \gamma N$ to $L + k - 1$. This gives us $F_{L+k} \geq 1 - \frac{1}{B}\sum_{i=1}^{\gamma N} C_{L+k-i}F_{L+k-i} \geq$

$1 - \frac{1}{B} \sum_{i=1}^{\gamma N} \tilde{C}_{L+k-i} F_{L+k-i}$. We immediately get

$$\sum_{\ell=L}^{L+k} \tilde{C}_\ell \tilde{F}_\ell \leq \tilde{C}_{L+k} F_{L+k} + \sum_{\ell=L}^{L+k-1} \tilde{C}_\ell F_\ell \enspace .$$

Now, it only remains to bound $\mathbf{E}\left[\sum_{\ell=L}^{L+\sqrt{\gamma}N-1} \tilde{C}_\ell \tilde{F}_\ell\right] = \sum_{\ell=L}^{L+\sqrt{\gamma}N-1} \mathbf{E}\left[\tilde{C}_\ell\right] \mathbf{E}\left[\tilde{F}_\ell\right]$. By definition $\mathbf{E}\left[\tilde{C}_\ell\right] = \frac{B}{\gamma N}$ for every ℓ, it is enough to show that $\frac{1}{\sqrt{\gamma}N} \sum_{\ell=L}^{L+\sqrt{\gamma}N-1} \mathbf{E}\left[\tilde{F}_\ell\right] \geq \left(\frac{1}{2} - \sqrt{\gamma} - \frac{1}{4\sqrt{\gamma}N}\right)$.

Define $a_\ell = \mathbf{E}\left[\tilde{F}_\ell\right]$. We have $a_\ell \geq 1 - \frac{1}{B} \sum_{i=1}^{\gamma N} \mathbf{E}\left[\tilde{C}_{\ell-i}\right] a_{\ell-i} = 1 - \frac{1}{\gamma N} \sum_{i=1}^{\gamma N} a_{\ell-i}$ for $\ell \geq L$ and $a_\ell = 1$ for $\ell < L$. Averaging over the rounds $L, \ldots, L + \sqrt{\gamma}N - 1$ we get

$$\frac{1}{\sqrt{\gamma}N} \sum_{\ell=L}^{L+\sqrt{\gamma}N-1} a_\ell \geq \frac{1}{\sqrt{\gamma}N} \sum_{\ell=L}^{L+\sqrt{\gamma}N-1} \left(1 - \frac{1}{\gamma N} \sum_{i=1}^{\gamma N} a_{\ell-i}\right)$$
$$= \frac{1}{\sqrt{\gamma}N} \left(\sqrt{\gamma}N - \frac{1}{\gamma N} \sum_{\ell=L}^{L+\sqrt{\gamma}N-1} \sum_{i=1}^{\gamma N} a_{\ell-i}\right) \enspace .$$

Here, we change the order of summation and split the inner sum into two parts which we bound separately

$$\frac{1}{\sqrt{\gamma}N} \sum_{\ell=L}^{L+\sqrt{\gamma}N-1} a_\ell \geq \frac{1}{\sqrt{\gamma}N} \left(\sqrt{\gamma}N - \frac{1}{\gamma N} \sum_{i=1}^{\gamma N} \left(\sum_{\ell=L}^{L-1+i} a_{\ell-i} + \sum_{\ell=L+i}^{L+\sqrt{\gamma}N-1} a_{\ell-i}\right)\right) \enspace .$$

Since $a_{\ell-i} \leq 1$, we bound the first sum with $\sum_{\ell=L}^{L-1+i} a_{\ell-i} \leq i$. Furthermore, as $a_\ell \geq 0$ for all ℓ, we can pad the second sum so that $\sum_{\ell=L+i}^{L+\sqrt{\gamma}N-1} a_{\ell-i} \leq \sum_{\ell=L}^{L+\sqrt{\gamma}N-1} a_\ell$, thus we have $\frac{1}{\gamma N} \sum_{i=1}^{\gamma N} \sum_{\ell=L+i}^{L+\sqrt{\gamma}N-1} a_{\ell-i} \leq \sum_{\ell=L}^{L+\sqrt{\gamma}N-1} a_\ell$.

We use both bounds and get

$$\frac{1}{\sqrt{\gamma}N} \sum_{\ell=L}^{L+\sqrt{\gamma}N-1} a_\ell \geq \frac{1}{\sqrt{\gamma}N} \left(\sqrt{\gamma}N - \frac{1}{\gamma N} \sum_{i=1}^{\gamma N} i - \sum_{\ell=L}^{L+\sqrt{\gamma}N-1} a_\ell\right) \enspace .$$

This implies

$$\frac{1}{\sqrt{\gamma}N} \sum_{\ell=L}^{L+\sqrt{\gamma}N-1} a_\ell = \frac{1}{2} - \frac{\gamma N + 1}{4\sqrt{\gamma}N} = \frac{1}{2} - \frac{\sqrt{\gamma}}{4} - \frac{1}{4\sqrt{\gamma}N} \enspace .$$ ◀

Now we have all parts required to prove the theorem.

Proof of Theorem 2. If in round ℓ an item is tentatively selected, let V_ℓ denote its value. Otherwise set $V_\ell = 0$. Our task is to bound the sum of $\mathbf{E}\left[V_\ell F_\ell\right] = \mathbf{E}\left[V_\ell \tilde{C}_\ell F_\ell\right]$.

Fixing which items come in rounds $1, \ldots, \ell$ fixes the set $S^{(\ell)}$. As any order among these items and the respective dummy items is still equally likely, the item coming in round ℓ can be considered being drawn uniformly at random. This way, by Lemma 1, we have for all $\ell \geq 2\sqrt{\frac{\gamma}{B}}N$

$$\mathbf{E}\left[V_\ell\right] \geq \frac{1}{\ell} \mathbf{E}\left[\sum_{j \in S^{(\ell)}} v_i\right] \geq \left(1 - 9\sqrt{\frac{1}{\frac{\ell}{N} \cdot \frac{B}{\gamma}}}\right) \frac{1}{N} \frac{1 - \frac{1}{2}\sqrt{\frac{\gamma}{B}}}{1+\gamma} \sum_{j \in \mathrm{OPT}^*} v_j \enspace .$$

Next, observe that V_ℓ given that $\tilde{C}_\ell = 1$ is independent of F_ℓ. This is due to the fact that the algorithm is comparison based. For this reason, the course of events in rounds $1, \ldots, \ell - 1$ is independent of the identity of the item from $S^{(\ell)}$ that actually arrives in round ℓ. Therefore the events leading up to round ℓ are identical, although they might involve different items. Also exploiting that $V_\ell = 0$ if $\tilde{C}_\ell = 0$, we get

$$\begin{aligned}\mathbf{E}\left[V_\ell \tilde{C}_\ell F_\ell\right] &= \mathbf{Pr}\left[\tilde{C}_\ell = 1, F_\ell = 1\right] \mathbf{E}\left[V_\ell \mid \tilde{C}_\ell = 1, F_\ell = 1\right] \\ &= \mathbf{Pr}\left[\tilde{C}_\ell = 1, F_\ell = 1\right] \mathbf{E}\left[V_\ell \mid \tilde{C}_\ell = 1\right] = \frac{\mathbf{Pr}\left[\tilde{C}_\ell = 1, F_\ell = 1\right]}{\mathbf{Pr}\left[\tilde{C}_\ell = 1\right]} \mathbf{E}\left[V_\ell\right] \ .\end{aligned}$$

To get the bound, we split the input sequence into blocks of length $\sqrt{\gamma}N$ and apply Lemma 3 on each of these blocks.

$$\begin{aligned}&\mathbf{E}\left[\sum_{j \in \mathrm{ALG}} v_j\right] \geq \sum_{k=2}^{\lfloor \frac{1}{\sqrt{\gamma}} \rfloor - 1} \sum_{\ell = k\sqrt{\gamma}N + 1}^{(k+1)\sqrt{\gamma}N} \mathbf{E}\left[V_\ell \tilde{C}_\ell F_\ell\right] \\ &\geq \sum_{k=2}^{\lfloor \frac{1}{\sqrt{\gamma}} \rfloor - 1} \sum_{\ell = k\sqrt{\gamma}N + 1}^{(k+1)\sqrt{\gamma}N} \frac{\mathbf{Pr}\left[\tilde{C}_\ell = 1, F_\ell = 1\right]}{\mathbf{Pr}\left[\tilde{C}_\ell = 1\right]} \left(1 - 9\sqrt{\frac{1}{\frac{\ell}{N} \cdot \frac{B}{\gamma}}}\right) \frac{1}{N} \frac{1 - \frac{1}{2}\sqrt{\frac{\gamma}{B}}}{1 + \gamma} \sum_{j \in \mathrm{OPT}^*} v_j \\ &\geq \left(\frac{1}{2} - \frac{7}{4}\sqrt{\gamma} - \frac{37}{4}\sqrt{\frac{\gamma}{B}} - \frac{\gamma}{2} - \frac{1}{4\sqrt{\gamma}N}\right) \sum_{j \in \mathrm{OPT}^*} v_j \ .\end{aligned}$$

Details on the calculations can be found in the full version. ◀

2.3 Improved Analysis for Large Capacities

For the same algorithm, we can show a better competitive ratio if B is large, converging to 1 asymptotically.

▶ **Theorem 4.** *Algorithm 1 is at least $\left(1 - \frac{4}{\sqrt{B}} - \frac{41}{2}\sqrt{\frac{\gamma}{B}} - 3\gamma - O\left(\frac{1}{B}\right)\right)$-competitive for the temp secretary problem with duration γ for all items.*

The proof of this theorem is very similar to the proof of Theorem 2. Again, we will bound the number of rounds in which $\tilde{C}_\ell = 1$ but $F_\ell = 0$. The main difference is that we consider blocks of γN rounds each. In this case, the duration of an item corresponds to the length of the block. Therefore, no more than B items can be feasibly selected in any block.

If B items are selected at the beginning of such a block, then it is feasible to select one item for every item that times out. We use this concept and construct a symmetric random walk. The maximum of this random walk upper bounds the number of failure events, in which the algorithm performs a tentative selection before sufficiently many previous items have timed out.

In contrast to Lemma 3, the expected ratio of failure events to successful selections of item is decreasing in B and therefore our competitive ratio in Theorem 4 tends to 1 as B increases.

▶ **Lemma 5.** *The expected number of rounds $\ell \in \{L, \ldots, L + \gamma N - 1\}$ in which $\tilde{C}_\ell = 1$ but $F_\ell = 0$ is $\left|\left\{\ell \mid \tilde{C}_\ell = 1 \wedge F_\ell = 0\right\}\right| \leq \sqrt{B} + \frac{\pi^2}{6}\sqrt{B} + 2\sqrt{\frac{4B}{3\pi}} + O(1) \leq 4\sqrt{B} + O(1)$.*

Proof. We claim that the number of rounds $\ell \in \{L, \ldots, L + \gamma N - 1\}$ for which $\tilde{C}_\ell = 1$ but $F_\ell = 0$ is bounded by

$$Q := \max_{L \leq \ell < L + \gamma N} \left(\sum_{r=L}^{\ell} \tilde{C}_r - \sum_{r=L-\gamma N}^{\ell - \gamma N} \tilde{C}_r\right) + \left|\sum_{r=L-\gamma N}^{L-1} \tilde{C}_r - B\right| \ .$$

To show this, we define an alternative sequence $(\tilde{C}'_r)_{r\in[N]}$ by setting $\tilde{C}_r = \tilde{C}'_r$ for every r except for the first Q occurrences of $\tilde{C}_r = 1$ after L, where we set $\tilde{C}'_r = 0$. Consider an $\ell \geq L$ such that $\tilde{C}'_\ell = 1$. Now observe that

$$\sum_{r=\ell-\gamma N}^{\ell} \tilde{C}'_r = \sum_{r=\ell-\gamma N}^{\ell} \tilde{C}_r - Q \leq \sum_{r=L-\gamma N}^{L-1} \tilde{C}_r - \left| \sum_{r=L-\gamma N}^{L-1} \tilde{C}_r - B \right| \leq B \enspace .$$

This implies that $\sum_{r=L}^{L+\gamma N-1} \tilde{C}_r F_r \geq \sum_{r=L}^{L+\gamma N-1} \tilde{C}'_r$ because every case of $\tilde{C}_r = 1$ but $F_r = 0$ can be matched to a case where $\tilde{C}_r = 1$ but $\tilde{C}'_r = 0$. So, to show the lemma, it only remains to show that $\mathbf{E}[Q] = O(\sqrt{B})$.

First, observe that $\sum_{r=L-\gamma N}^{L-1} \tilde{C}_r$ is drawn from a binomial distribution with γN trials and probability $\frac{B}{\gamma N}$. Therefore, its expectation is $\mu = B$ and its standard deviation is $\sigma = \sqrt{B - \frac{B}{\gamma N}} \leq \sqrt{B}$. Thus by Chebyshev inequality, we get

$$\mathbf{E}\left[\left| \sum_{r=L-\gamma N}^{L-1} \tilde{C}_r - B \right| \right] \leq \sigma + \sum_{k=1}^{\infty} \mathbf{E}\left[\left| \sum_{r=L-\gamma N}^{L-1} \tilde{C}_r - B \right| \geq k\sigma \right] \sigma \leq \sigma + \sum_{k=1}^{\infty} \frac{\sigma}{k^2} = \sqrt{B} + \frac{\pi^2}{6}\sqrt{B}.$$

So, it only remains to show that

$$\mathbf{E}\left[\max_{L \leq \ell < L+\gamma N} \left(\sum_{r=L}^{\ell} \tilde{C}_r - \sum_{r=L-\gamma N}^{\ell-\gamma N} \tilde{C}_r \right) \right] \leq 2\sqrt{\frac{4B}{3\pi}} + O(1) \enspace .$$

Let $\tilde{C}''_\ell \in \{-1, 0, 1\}$ be a random variable with $\tilde{C}''_\ell = \tilde{C}_\ell - \tilde{C}_{\ell-\gamma N}$. Each $\tilde{C}''_\ell$ takes the values 1 and -1 with probability $p = \frac{B}{\gamma N}(1 - \frac{B}{\gamma N})$ each and 0 with the remaining probability $1 - 2p = \left(1 - \frac{2B}{\gamma N} + 2(\frac{B}{\gamma N})^2\right)$. Furthermore, because the $\tilde{C}_\ell$ random variables are independent, $\tilde{C}''_L, \ldots, \tilde{C}''_{L+\gamma N-1}$ also are.

We interpret this random process on the $\tilde{C}''_\ell$ as a random walk of length γN that moves up or down with probability p and stays the same with probability $1 - 2p$. In the next part of the proof, we are going to show that the maximal deviation of this random walk is in $\Theta(\sqrt{B})$. To this end, we condition our random walk on the number of zeros that occur. The remaining random walk is symmetric, thus results from the literature apply.

It has been shown that, for a symmetric random walk that starts in position 0 and does k steps, the expected final position is $\mathbf{E}[S_k] = \sqrt{\frac{2k}{3\pi}} + O(k^{-\frac{1}{2}})$ [14]. Furthermore, it is well known that the expected maximal deviation during such a random walk is $\mathbf{E}[M_k] \leq 2\mathbf{E}[S_k]$. Now, let K be the number of times $\tilde{C}''_r \neq 0$ for $r \in \{L, \ldots, L + \gamma N - 1\}$. Then we have

$$\mathbf{E}\left[\max_{L \leq \ell < L+\gamma N} \left(\sum_{r=L}^{\ell} \tilde{C}_r - \sum_{r=L-\gamma N}^{\ell-\gamma N} \tilde{C}_r \right) \right] \leq \sum_{k=0}^{\gamma N} \mathbf{E}[M_k] \cdot \mathbf{Pr}[K = k]$$

$$\leq \sum_{k=0}^{\gamma N} 2\mathbf{E}[S_k] \cdot \mathbf{Pr}[K = k] \leq \mathbf{E}\left[2\sqrt{\frac{2K}{3\pi}} + O(K^{-\frac{1}{2}}) \right] \leq 2\sqrt{\frac{2\mathbf{E}[K]}{3\pi}} + O(1)$$

$$= 2\sqrt{\frac{4B}{3\pi}} + O(1) \enspace . \qquad \blacktriangleleft$$

We use the same proof structure as in the previous proof of Theorem 2, but now we replace Lemma 3 with Lemma 5.

Proof of Theorem 4. Again, if in round ℓ an item is tentatively selected, let V_ℓ denote its value. Otherwise set $V_\ell = 0$. By the same arguments as in the proof of Theorem 2, we have

$$\mathbf{E}\left[V_\ell \tilde{C}_\ell F_\ell\right] = \frac{\mathbf{Pr}\left[\tilde{C}_\ell = 1, F_\ell = 1\right]}{\mathbf{Pr}\left[\tilde{C}_\ell = 1\right]} \mathbf{E}\left[V_\ell\right] \enspace .$$

To get the bound, we split the input sequence into blocks of length γN and ignore the blocks in which there is a round for which Lemma 1 does not hold.

$$\mathbf{E}\left[\sum_{j \in \mathrm{ALG}} v_j\right] \geq \sum_{k=\left\lceil 2\sqrt{\frac{1}{\gamma B}}\right\rceil}^{\lfloor \frac{1}{\gamma} \rfloor - 1} \sum_{\ell = k\gamma N + 1}^{(k+1)\gamma N} \mathbf{E}\left[V_\ell \tilde{C}_\ell F_\ell\right]$$

$$\geq \sum_{k=\left\lceil 2\sqrt{\frac{1}{\gamma B}}\right\rceil}^{\lfloor \frac{1}{\gamma} \rfloor - 1} \sum_{\ell = k\gamma N + 1}^{(k+1)\gamma N} \frac{\mathbf{Pr}\left[\tilde{C}_\ell = 1, F_\ell = 1\right]}{\mathbf{Pr}\left[\tilde{C}_\ell = 1\right]} \left(1 - 9\sqrt{\frac{1}{\frac{\ell}{N} \cdot \frac{B}{\gamma}}}\right) \cdot \frac{1}{N} \frac{1 - \frac{1}{2}\sqrt{\frac{\gamma}{B}}}{1 + \gamma} \sum_{j \in \mathrm{OPT}^*} v_j$$

$$\geq \left(1 - \frac{4}{\sqrt{B}} - \frac{41}{2}\sqrt{\frac{\gamma}{B}} - 3\gamma - O\left(\frac{1}{B}\right)\right) \sum_{j \in \mathrm{OPT}^*} v_j \enspace .$$

The missing details on the calculations are included in the full version. ◀

3 The Temp Secretary Problem with Packing Constraints

Next, we turn to the temp secretary with general linear packing constraints. This generalizes the timed cardinality constraint of the temp secretary problem towards multiple knapsack constraints. Therefore we can model, e.g., production capacities of different types. Now, the problem is not about selecting a set of best candidates, but a set of contracts with different resource demands such that the value of the selected contracts is maximized and all demands are fulfilled at any point in time.

We assume that the items, or possible contracts, that arrive over time are variables of a packing LP that have to be set immediately and irrevocably at time of arrival. In more detail, we assume that an adversary defines an $n \times m$ constraint matrix A, a capacity vector b, and an objective function vector v. Again, for each variable x_j an arrival time is drawn independently uniformly at random from $[0, 1]$. We now have to find an assignment $\hat{x}_j \in \{0, 1\}$ for all $j \in [n]$, with the property that for every $t \in [0, 1]$, the set of variables that arrive between t and $t + \gamma$ solve the packing LP. That is, for every t, we need $Ax' \leq b$, where $x'_j = \hat{x}_j$ if $t_j \in [t, t + \gamma]$ and 0 otherwise. The objective is to maximize $v^T \hat{x}$. So, $\hat{x}$ represents the aggregate vector of which items are selected whenever they are present in the system. Therefore, the temp secretary problem can be captured by having only one constraint with all coefficients being 1 and the capacity being the cardinality bound. As the output of our algorithm will be integral, this algorithm also solved the temp secretary problem.

We assume that the matrix A is sparse. That is, each column of A contains at most d non-zero entries. Furthermore, we assume that the capacities in each constraint are large compared to the respective coefficients in A. Our bound will depend on the capacity ratio B, defined as $B = \min_{i \in [m]} \frac{b_i}{\max j \in [n] a_{i,j}}$. When modeling the temp secretary problem as above, this capacity ratio coincides with the capacity bound.

Again, we use a relaxation without temporal constraints like the one in Section 2. In this case, it reads $\max v^T x$ s.t. $Ax \leq \lceil \frac{1}{\gamma} \rceil b$, $0 \leq x_j \leq 1$ for all $j \in [n]$. While our algorithm chooses online which items to select and which to reject subject to the temporal constraints,

Algorithm 2: Scaling Algorithm for packing constraints

```
for every arriving item $j \in [n]$ do
    Set $t := \tau_j$;                                              // arrival time of $j$
    Let $x^{(t)}$ be an optimal solution to the LP $\max v^T x$ s.t. $Ax \leq t(1-\epsilon)\frac{b}{\gamma}$, $0 \leq x_{j'} \leq 1$
     for $j' \in U_{\leq t}$;                                       // optimal offline solution
    $\hat{x}^{(t)}_{j'} = \begin{cases} 1, & \text{with prob. } x^{(t)}_{j'} \text{ if } j' = j; \\ 0, & \text{otherwise;} \end{cases}$   // randomized rounding
    if $\hat{x} + \hat{x}^{(t)}$ is feasible with respect to temporal constraints then
        Set $\hat{x} := \hat{x} + \hat{x}^{(t)}$;                   // make allocation to $j$ permanent
```

our point of comparison is the best fractional solution to this LP relaxation. We use an algorithm similar to the one presented in [17] to solve the relaxed problem. We scale down the aggregated constraints slightly and solve the resulting linear packing problem. Next, we use randomized rounding to transform the fractional solution into our integral, tentative solution. At this point, the algorithm behaves exactly like the one in Section 2. If the item that just arrived is part of the tentative solution and it is feasible to select it, then the algorithm adds it to the online solution. In contrast to the analysis in [17], we have to show that temporal constraints are likely fulfilled although the algorithm only operates on the relaxed ones.

To define the algorithm, let $U_{\leq t}$ be the set of variables $U_{\leq t} \subseteq [n]$ that arrive before time t. Let $\epsilon = \sqrt{6\frac{(1+\log d+\log B)}{B}}$.

This algorithm can also be extended to the case in which there are not only timed constraints but also global ones. Additionally, it can be applied when multiple variables arrive at a time like in [17]. We omit these generalizations because the techniques are identical to the ones presented here, but correct notation gets a lot more involved.

▶ **Theorem 6.** *Algorithm 2 is* $\frac{1}{1+\gamma} - O\left(\sqrt{\frac{1+\log d+\log B}{B}}\right)$*-competitive for the temp secretary problem with linear packing constraints.*

Please note that this algorithm is invariant to scaling constraints and therefore we can assume without loss of generality that $\max_{j\in[n]} a_{i,j} \leq 1$ and $b_i \geq B$ for every constraint $i \in [m]$.

For the proof, we will first bound the expected consumption of the tentative solution of the algorithm. Then we will use a Chernoff bound to bound the probability that last *if*-clause of the algorithm is violated for a single constraint. Finally, we will aggregate the probabilities for all constraints in the relaxation.

We discretize time, like in Section 2, with arbitrary precision such that every discrete time interval only contains a single item of the input. We have $N \gg n$ rounds, spanning time $\frac{1}{N}$ each. For each of the $N - n$ rounds in which no variable arrives, we introduce a dummy variable with all coefficients zero. These N variables can now be considered being assigned to the rounds by a uniformly drawn permutation. In the proofs, we overload notation and write $x^{(\ell)} = x^{(t)}$ and $U_{\leq \ell} = U_{\leq t}$ if t lies within round ℓ.

▶ **Lemma 7.** *Let $U_{<\ell} \subseteq [n]$ be the set of items that arrive before round ℓ. Then, conditioned on this set, the sum of previous tentative allocations violates any constraint $i \in [m]$ with* *probability at most*

$$\mathbf{Pr}\left[\left(\sum_{\ell'=\ell-\gamma N}^{\ell-1} A\hat{x}^{(\ell')}\right)_i > b_i - 1 \;\middle|\; U_{<\ell}\right] \leq \frac{1}{dB}$$

if $\epsilon = \sqrt{6\frac{1+\log d+\log B}{B}} \leq \frac{1}{2}$.

We describe the capacity used by the tentative selection through random variables $X_{\ell'} = \left(A\hat{x}^{(\ell')}\right)_i$. These random variables are not independent but 1-correlated as defined by Panconesi and Srinivasan [24] and this allows us to apply a Chernoff bound to proof the lemma. Details can be found in the full version.

Proof of Theorem 6. We can assume without loss of generality that $\sqrt{6\frac{1+\log d+\log B}{B}} \leq \frac{1}{2}$ because otherwise the theorem statement follows trivially.

First, we bound the value of the tentative allocation performed in round $\ell \geq 2\sqrt{\frac{1+\ln d}{B}}N$ using a result in [17]. Compared to [17] our non-temporal relaxation is scaled down by an additional factor of $1-\epsilon$ this gives us

$$\mathbf{E}\left[v^T x^{(\ell)}\right] \geq \left(1 - 9\sqrt{\frac{1}{\frac{\ell}{N}\cdot(1-\epsilon)\frac{B}{\gamma}}}\right)\frac{\ell}{N}(1-\epsilon)\cdot \max_{x: Ax \leq b/\gamma, 0 \leq x_j \leq 1 \text{ for all } j \in [m]} v^T x$$

Letting x^* denote the optimal solution to the relaxation, we also have

$$\max_{x: Ax \leq b/\gamma, 0 \leq x_j \leq 1 \text{ for all } j \in [m]} v^T x \geq \frac{\frac{1}{\gamma}}{\lceil \frac{1}{\gamma} \rceil} v^T x^* \geq \frac{1}{1+\gamma} v^T x^* \enspace .$$

So, this implies

$$\mathbf{E}\left[v^T x^{(\ell)}\right] \geq \left(1 - 9\sqrt{\frac{1}{\frac{\ell}{N}\cdot(1-\epsilon)\frac{B}{\gamma}}}\right)\frac{\ell}{N}\frac{1-\epsilon}{1+\gamma} v^T x^* \enspace .$$

Observe that this outcome only depends on the set $U_{\leq \ell}$ but not the order in this set. Therefore, the variable that arrives in round ℓ can be considered being drawn uniformly from $U_{\leq \ell}$. This way, we get

$$\mathbf{E}\left[v^T \hat{x}^{(\ell)} \;\middle|\; U_{\leq \ell}\right] = \frac{1}{\ell}\mathbf{E}\left[v^T x^{(\ell)} \;\middle|\; U_{\leq \ell}\right]$$

Note that these outcomes only depend on the sets $U_{<\ell}$ and $U_{\leq \ell}$ but not the order within $U_{<\ell}$.

To bound the probability of feasibility, we will use Lemma 7, conditioning on the set $U_{<\ell}$ and $U_{\leq \ell}$. Let j be the index of the variable arriving in round ℓ. Taking a union bound over all $\leq d$ constraints in which variable j has non-zero coefficients, we get

$$\begin{aligned}&\mathbf{Pr}\left[\text{it is feasible to set } \hat{x}_j = 1 \mid U_{<\ell}, U_{\leq \ell}\right] \\ &\qquad \geq 1 - d \cdot \sum_{i: a_{i,j} > 0} \mathbf{Pr}\left[\left(\sum_{\ell'=\ell-\gamma N}^{\ell-1} A\hat{x}^{(\ell')}\right)_i > b_i - 1 \;\middle|\; U_{<\ell}\right] \geq \left(1 - \frac{1}{B}\right) \enspace .\end{aligned}$$

Overall, the expected value of the allocation performed in round ℓ is at least

$$\mathbf{E}\left[v^T \hat{x}^{(\ell)}\right] \geq \left(1 - 9\sqrt{\frac{1}{\frac{\ell}{N}\cdot(1-\epsilon)\frac{B}{\gamma}}}\right)\frac{1}{N}\frac{1-\epsilon}{1+\gamma}\left(1 - \frac{1}{B}\right) v^T x^* \enspace .$$

Algorithm 3: Scaling Algorithm for different lengths and capacity B

```
for every arriving item $j$ do
    Set $t := \tau_j$, $U_{\leq t}$ := items arrived so far;
    Let $S^{(t)} := \textsc{GreedyRoundUp}(U_{\leq t}, \alpha t B)$;
    if $j \in S^{(t)}$ then                          // if among best items
        if $S \cup \{j\}$ is a feasible schedule then    // and if feasible
            Set $S := S \cup \{j\}$;                     // then select j
```

Taking the sum of all these bounds, we get

$$\mathbf{E}\left[v^T \hat{x}\right] \geq \sum_{\ell=2\sqrt{\frac{1+\ln d}{B}}N}^{N} \left(1 - 9\sqrt{\frac{1}{\frac{\ell}{N}\cdot(1-\epsilon)\frac{B}{\gamma}}}\right)\frac{1}{N}\cdot\frac{1-\epsilon}{1+\gamma}\left(1-\frac{1}{B}\right)v^T x^*$$
$$\geq \frac{1-O(\epsilon)}{1+\gamma} v^T x^* \ . \qquad ◀$$

4 The Temp Secretary Problem with Different Lengths

We generalize the Temp Secretary Problem towards different item durations $\lambda_j \leq \gamma$ for all items $j \in [n]$. To define the relaxation, we use the fact that pointwise $\sum_{j \in \mathrm{OPT}} \lambda_j \leq B(1+\gamma)$. This is due to the fact that an item j selected at time $\tau_j = 1$ will be active until time $1 + \lambda_j$. Therefore, let OPT* denote the optimal solution to this knapsack problem with profits v_j. Due to the knapsack nature of the problem, the algorithm cannot be purely comparison-based anymore. Instead, whenever an item arrives, we compute an approximate knapsack solution and tentatively select the item if it is included in this solution. It will be crucial that these solutions only slowly change when adding or removing items. This is why, there is no obvious generalization of our algorithm and analysis to general packing LPs. In more detail, for $U' \subseteq [n]$ and $\Lambda > 0$ we define $\textsc{GreedyRoundUp}(U', \Lambda)$ as the set $U'' \subseteq U'$ which we get by ordering the items in U' in non-increasing order $\frac{v_j}{\lambda_j}$ and taking the minimal prefix such that $\sum_{j \in U''} \lambda_j \geq \Lambda$.

▶ **Theorem 8.** *With $\alpha = 1/2$, Algorithm 3 is at least $\frac{1}{4} - \Theta\left(\sqrt{\gamma}\right)$-competitive for the temp secretary problem with arbitrary durations $\lambda_j \leq \gamma$.*

We use the same proof structure like in Theorem 2. Here Lemma 9 gives a bound on the expected value of the tentative solution. The proof of this lemma is very similar to the proof of Lemma 1 and can be found in the full version.

▶ **Lemma 9.** *For $\ell \geq 2\sqrt{\frac{\gamma}{B}}N$*

$$\mathbf{E}\left[\sum_{j \in S^{(\ell)}} v_j\right] \geq \alpha\left(1 - 9\sqrt{\frac{1}{\frac{\ell}{N}\frac{B}{\gamma}}}\right)\frac{\ell}{N}\frac{1}{1+\gamma}\sum_{j \in \mathrm{OPT}^*} v_j \ .$$

The main difference to previous proofs in this paper is the fact that we cannot bound the probability of a selection for a fixed round because it depends on the lengths of the items that arrive up to this round. Fortunately, we can bound the probability that a selection would be feasible by the following lemma.

▶ **Lemma 10.** *Conditioning on any set of arrivals in rounds* $1, \dots, \ell - 1$, *the probability that an item can be feasibly selected in round* ℓ *is at least* $1 - \alpha - \frac{3\gamma N}{\ell - \gamma N}$.

As a key idea, we show that the probability of a selection does not significantly change between rounds.

▶ **Lemma 11.** *Let* $H_{i,\ell}$ *be the 0/1 indicator if an item with duration at least* i *rounds is tentatively selected in rounds* ℓ. *It holds that* $\mathbf{Pr}\left[H_{i,\ell-i} = 1\right] \leq \mathbf{Pr}\left[H_{i,\ell} = 1\right] + \frac{\alpha B + 1}{\ell - i}$.

Proof. First, we observe that an item j is tentatively selected in round ℓ' if it is contained in $S^{(\ell')}$ and arrives in round ℓ'. As the set $S^{(\ell')}$ only depends on the items arriving in rounds $1, \dots, \ell'$ but not on their order, the probability of j being tentatively selected is exactly $\frac{1}{\ell'}\mathbf{Pr}\left[j \in S^{(\ell')}\right]$. Therefore, to prove this lemma, we will compare the sets $S^{(\ell-i)}$ and $S^{(\ell)}$.

Instead of making statements about the set $S^{(\ell-i)}$ directly, we instead use a set $\tilde{S}$ defined as $\tilde{S}$:=GreedyRoundUp$(U_{\leq \ell - i}, \alpha\frac{\ell}{N}B)$. Note that by definition of GreedyRoundUp, this set $\tilde{S}$ is a superset of $S^{(\ell-i)}$ =GreedyRoundUp$(U_{\leq \ell-i}, \alpha\frac{\ell-i}{N}B)$.

Let now $\mathcal{E}^*_{i,\ell-i}$ be the event that an item of length at least i rounds arrives in round $\ell - i$ that is contained in $\tilde{S} \cap S^{(\ell)}$, and let $\mathcal{E}^-_{i,\ell-i}$ be the event that an item of length at least i rounds arrives in round $\ell - i$ that is contained in $\tilde{S} \setminus S^{(\ell)}$. As $S^{(\ell-i)}$ is contained in $\tilde{S}$, we have $\mathbf{Pr}\left[H_{i,\ell-i} = 1\right] \leq \mathbf{Pr}\left[\mathcal{E}^*_{i,\ell-i}\right] + \mathbf{Pr}\left[\mathcal{E}^-_{i,\ell-i}\right]$.

To bound the first probability, we use that every item in $S^{(\ell)}$ has already arrived in round $\ell - i$ with probability $\frac{\ell-i}{\ell}$. Every such item arrives exactly in round $\ell - i$ with probability $\frac{1}{\ell-i}$. Therefore we have $\mathbf{Pr}\left[\mathcal{E}^*_{i,\ell-i}\right] = \mathbf{Pr}\left[H_{i,\ell} = 1\right]$.

For the second probability, observe that $\tilde{S}$ contains all elements of $S^{(\ell)}$ that have arrived by round $\ell - i$, i.e., $\tilde{S} \supseteq S^{(\ell)} \cap U_{\leq \ell-i}$. Also if $\sum_{j \in S^{(\ell)}} \lambda_j < \alpha\frac{\ell}{N}B - \gamma$, it has to be $S^{(\ell)} = U_{\leq \ell}$. Therefore, $\tilde{S} = S^{(\ell)}$ and nothing has to be shown. This implies that $\mathbf{E}\left[\sum_{j \in \tilde{S} \cap S^{(\ell)}} \lambda_j\right] \geq \frac{\ell-i}{\ell}\alpha\frac{\ell}{N}B = \alpha\frac{\ell-i}{N}B$ because the probability for each item in $S^{(\ell)}$ to arrive until round $\ell - i$ is $\frac{\ell-i}{\ell}$. Therefore, the λ_j values for all but one item in $\tilde{S} \setminus S^{(\ell)}$ add up to at most $\alpha B\frac{i}{N}$.

Let K denote the number of items in $\tilde{S} \setminus S^{(\ell)}$ of length at least i blocks. On the one hand, K is bounded by $\mathbf{E}\left[K\right] \leq \alpha B + 1$ due to the above considerations. On the other hand, conditioning on K, we can bound the probability of $\mathbf{Pr}\left[\mathcal{E}^-_{i,\ell-i}\right]$ by $\mathbf{Pr}\left[\mathcal{E}^-_{i,\ell-i} \;\middle|\; K = k\right] \leq \frac{k}{\ell-i}$.

Taking the expectation over K, we get $\mathbf{Pr}\left[\mathcal{E}^-_{i,\ell-i}\right] \leq \frac{\alpha B+1}{\ell-i}$. ◀

Proof of Lemma 10. Using Lemma 11, we now have

$$\begin{aligned}\mathbf{Pr}\left[\text{round } \ell \text{ would be feasible} \mid U_{\leq \ell-1}\right] &\geq 1 - \frac{1}{B}\sum_{i=1}^{\gamma N}\mathbf{E}\left[H_{i,\ell-i} \mid U_{\leq \ell-1}\right] \\ &\geq 1 - \frac{1}{B}\sum_{i=1}^{\gamma N}\mathbf{E}\left[H_{i,\ell-1} \mid U_{\leq \ell-1}\right] - \frac{\alpha\gamma N + 1}{\ell - \gamma N} \quad .\end{aligned}$$

Observe that $\sum_{i=1}^{\gamma N}\mathbf{E}\left[H_{i,\ell-1} \mid U_{\leq \ell-1}\right]$ is exactly the length in rounds of the tentative selection in round $\ell - 1$, counting no tentative selection as 0. This length is bounded by $\frac{1}{\ell-1}\mathbf{E}\left[\sum_{j \in S^{(\ell-1)}} \lambda_j N \;\middle|\; U_{\leq \ell-1}\right]$. This sum can be bounded pointwise by $\sum_{j \in S^{(\ell-1)}} \lambda_j \leq \alpha\frac{\ell-1}{N}B + \gamma$. So we have $\sum_{i=1}^{\gamma N}\mathbf{E}\left[H_{i,\ell-1} \mid U_{\leq \ell-1}\right] \leq \alpha + \frac{\gamma N}{\ell-1}$.

This means $\mathbf{Pr}\left[\text{round } \ell \text{ would be feasible} \mid U_{\leq \ell-1}\right] \geq 1 - \alpha - \frac{(1+\alpha)\gamma N+1}{\ell-\gamma N} \geq 1 - \alpha - \frac{3\gamma N}{\ell-\gamma N}$. ◀

The remaining proof follows essentially the pattern of the previous sections.

Proof of Theorem 8. We have already shown that

$$\mathbf{Pr}\left[\text{round } \ell \text{ would be feasible} \mid U_{\leq \ell-1}\right] \geq 1 - \alpha - \frac{3\gamma N}{\ell - \gamma N} \enspace .$$

By combining this bound with Lemma 9, we get that the expected value gained from round $\ell \geq 2\sqrt{\gamma}N$ is at least

$$\frac{1}{2\ell} \cdot \left(1 - 9\sqrt{\frac{\gamma}{\frac{\ell}{N}B}}\right) \frac{\ell}{N} \frac{\sum_{j \in \mathrm{OPT}^*} v_j}{1+\gamma} \cdot \left(\frac{1}{2} - \frac{3\gamma N}{\ell - \gamma N}\right) \enspace .$$

Similar calculations like in the proof of Theorem 2 and $\alpha = \frac{1}{2}$ give a competitive ratio of

$$\begin{aligned} \mathbf{E}\left[\sum_{j \in \mathrm{ALG}} v_j\right] &\geq \sum_{k=2}^{\frac{1}{\sqrt{\gamma}}-1} \sum_{\ell = k\sqrt{\gamma}N}^{(k+1)\sqrt{\gamma}N-1} \frac{1}{2}\left(1 - 9\sqrt{\frac{\gamma}{\frac{\ell}{N}B}}\right) \frac{1}{N} \frac{\sum_{j \in \mathrm{OPT}^*} v_j}{1+\gamma} \cdot \left(\frac{1}{2} - \frac{3\gamma N}{\ell - \gamma N}\right) \\ &\geq \left(\frac{1}{4} - 5\sqrt{\gamma} - \frac{3}{2}\gamma \ln\left(\frac{1}{\sqrt{\gamma}}\right)\right) \cdot \frac{1}{1+\gamma} \sum_{j \in \mathrm{OPT}^*} v_j \enspace . \end{aligned}$$

The missing details are included in the full version. ◀

5 Future Work

The area of online problems with stochastic arrivals and temporal constraints leaves many open research directions. First of all, only very few impossibility results are known in this model. It seems natural that bounds in the related random order model should generalize in some way. For example in the temp secretary problem with large capacities, it seems plausible that the competitive ratio cannot be better than $1 - \Omega\left(\sqrt{\frac{1}{B}}\right)$ like in [18], independent of γ. A small γ increases the overall capacity, but in every step the algorithm is still tightly restricted by the timed capacity B.

Furthermore, the results in this paper apply if the arrival times are each drawn independently uniformly from $[0, 1]$. This condition can be relaxed in several ways. Firstly, other distributions than the uniform one are of important interest. Although the algorithms in this paper do admit a reasonable generalization using quantiles, our analyses as they are do not extend. Secondly, it is also very interesting to weaken the assumption that arrival times are independent and identically distributed. There is only little work in related models [15, 8] and none for this particular setting.

Other fruitful directions could be the extension to other feasibility structures, such as (special classes of) matroid, and to other objective functions, such as submodular ones. Finally, it might also be interesting to let the algorithm decide the contract starting/finishing dates or its duration.

References

1 Shipra Agrawal and Nikhil R. Devanur. Fast algorithms for online stochastic convex programming. In *Proc. 26th Symp. Discr. Algorithms (SODA)*, pages 1405–1424, 2015. doi:10.1137/1.9781611973730.93.

2 Shipra Agrawal, Zizhuo Wang, and Yinyu Ye. A dynamic near-optimal algorithm for online linear programming. *Operations Research*, 62(4):876–890, 2014. doi:10.1287/opre.2014.1289.

3 Moshe Babaioff, Nicole Immorlica, David Kempe, and Robert Kleinberg. A knapsack secretary problem with applications. In *Proc. 10thIntl. Workshop Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 16–28, 2007. doi:10.1007/978-3-540-74208-1_2.

4 Moshe Babaioff, Nicole Immorlica, and Robert Kleinberg. Matroids, secretary problems, and online mechanisms. In *Proc. 18th Symp. Discr. Algorithms (SODA)*, pages 434–443, 2007. URL: http://dl.acm.org/citation.cfm?id=1283383.1283429.

5 MohammadHossein Bateni, Mohammad Taghi Hajiaghayi, and Morteza Zadimoghaddam. Submodular secretary problem and extensions. *ACM Trans. Algorithms*, 9(4):32, 2013. doi:10.1145/2500121.

6 Nikhil R. Devanur, Kamal Jain, Balasubramanian Sivan, and Christopher A. Wilkens. Near optimal online algorithms and fast approximation algorithms for resource allocation problems. In *Proc. 12thConf. Econom. Comput. (EC)*, pages 29–38, 2011. doi:10.1145/1993574.1993581.

7 Eugene B. Dynkin. The optimum choice of the instant for stopping a markov process. In *Sov. Math. Dokl*, volume 4, pages 627–629, 1963.

8 Hossein Esfandiari, Nitish Korula, and Vahab S. Mirrokni. Online allocation with traffic spikes: Mixing adversarial and stochastic models. In *Proceedings of the Sixteenth ACM Conference on Economics and Computation, EC'15, Portland, OR, USA, June 15-19, 2015*, pages 169–186, 2015. doi:10.1145/2764468.2764536.

9 Moran Feldman, Joseph Naor, and Roy Schwartz. Improved competitive ratios for submodular secretary problems (extended abstract). In *Proc. 14thIntl. Workshop Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 218–229, 2011. doi:10.1007/978-3-642-22935-0_19.

10 Moran Feldman, Ola Svensson, and Rico Zenklusen. A simple $O(\log\log(\text{rank}))$-competitive algorithm for the matroid secretary problem. In *Proc. 26th Symp. Discr. Algorithms (SODA)*, pages 1189–1201, 2015. doi:10.1137/1.9781611973730.79.

11 Amos Fiat, Ilia Gorelik, Haim Kaplan, and Slava Novgorodov. The temp secretary problem. In *Proc. 23rdEuropean Symp. Algorithms (ESA)*, pages 631–642, 2015. doi:10.1007/978-3-662-48350-3_53.

12 M. Gardner. Scientific american, 1960.

13 Anupam Gupta and Marco Molinaro. How experts can solve LPs online. In *Proc. 22ndEuropean Symp. Algorithms (ESA)*, pages 517–529, 2014. doi:10.1007/978-3-662-44777-2_43.

14 Edward G. Coffman Jr., Philippe Flajolet, Leopold Flatto, and Micha Hofri. The maximum of a random walk and its application to rectangle packing. *Probability in the Engineering and Informational Sciences*, 12(3):373–386, 1998. doi:10.1017/S0269964800005258.

15 Thomas Kesselheim, Robert D. Kleinberg, and Rad Niazadeh. Secretary problems with non-uniform arrival order. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 879–888, 2015. doi:10.1145/2746539.2746602.

16 Thomas Kesselheim, Klaus Radke, Andreas Tönnis, and Berthold Vöcking. An optimal online algorithm for weighted bipartite matching and extensions to combinatorial auctions. In *Proc. 21st European Symp. Algorithms (ESA)*, pages 589–600, 2013. doi:10.1007/978-3-642-40450-4_50.

17 Thomas Kesselheim, Klaus Radke, Andreas Tönnis, and Berthold Vöcking. Primal beats dual on online packing lps in the random-order model. In *Proc. 46th Symp. Theory of Computing (STOC)*, pages 303–312, 2014. `doi:10.1145/2591796.2591810`.

18 Robert D. Kleinberg. A multiple-choice secretary algorithm with applications to online auctions. In *Proc. 16th Symp. Discr. Algorithms (SODA)*, pages 630–631, 2005. URL: `http://dl.acm.org/citation.cfm?id=1070432.1070519`.

19 Nitish Korula and Martin Pál. Algorithms for secretary problems on graphs and hypergraphs. In *Proc. 36th Intl. Coll. Autom. Lang. Program. (ICALP)*, pages 508–520, 2009. `doi:10.1007/978-3-642-02930-1_42`.

20 Oded Lachish. O(log log rank) competitive ratio for the matroid secretary problem. In *Proc. 55th Symp. Foundations of Computer Science (FOCS)*, pages 326–335, 2014. `doi:10.1109/FOCS.2014.42`.

21 Denis V Lindley. Dynamic programming and decision theory. *Applied Statistics*, pages 39–51, 1961.

22 Richard J. Lipton and Andrew Tomkins. Online interval scheduling. In *Proc. 5thSymp. Discr. Algorithms (SODA)*, pages 302–311, 1994. URL: `http://dl.acm.org/citation.cfm?id=314464.314506`.

23 Tengyu Ma, Bo Tang, and Yajun Wang. The simulated greedy algorithm for several submodular matroid secretary problems. In *Proc. 30th Symp. Theoret. Aspects of Computer Science (STACS)*, pages 478–489, 2013. `doi:10.4230/LIPIcs.STACS.2013.478`.

24 Alessandro Panconesi and Aravind Srinivasan. Randomized distributed edge coloring via an extension of the chernoff-hoeffding bounds. *SIAM J. Comput.*, 26(2):350–368, 1997. `doi:10.1137/S0097539793250767`.

25 Gerhard J. Woeginger. On-line scheduling of jobs with fixed start and end times. *Theoret. Comput. Sci.*, 130(1):5–16, 1994. `doi:10.1016/0304-3975(94)90150-3`.

Hardness of Bipartite Expansion

Subhash Khot[1] and Rishi Saket[2]

1 **Department of Computer Science, New York University, NY, USA**
khot@cs.nyu.edu

2 **IBM Research, Bangalore, India**
rissaket@in.ibm.com

Abstract

We study the natural problem of estimating the expansion of subsets of vertices on one side of a bipartite graph. More precisely, given a bipartite graph $G(U, V, E)$ and a parameter β, the goal is to find a subset $V' \subseteq V$ containing β fraction of the vertices of V which minimizes the size of $N(V')$, the neighborhood of V'. This problem, which we call *Bipartite Expansion*, is a special case of submodular minimization subject to a cardinality constraint, and is also related to other problems in graph partitioning and expansion. Previous to this work, there was no hardness of approximation known for Bipartite Expansion.

In this paper we show the following strong inapproximability for Bipartite Expansion: for any constants $\tau, \gamma > 0$ there is no algorithm which, given a constant $\beta > 0$ and a bipartite graph $G(U, V, E)$, runs in polynomial time and decides whether

- (YES case) There is a subset $S^* \subseteq V$ s.t. $|S^*| \geq \beta|V|$ satisfying $|N(S^*)| \leq \gamma|U|$, or
- (NO case) Any subset $S \subseteq V$ s.t. $|S| \geq \tau\beta|V|$ satisfies $|N(S)| \geq (1-\gamma)|U|$,

unless NP $\subseteq \cap_{\varepsilon>0}$DTIME $\left(2^{n^{\varepsilon}}\right)$ i.e. NP has subexponential time algorithms.

We note that our hardness result stated above is a vertex expansion analogue of the Small Set (Edge) Expansion Conjecture of Raghavendra and Steurer [23].

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases inapproximability, bipartite expansion, PCP, submodular minimization

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.55

1 Introduction

Graph partitioning and graph expansion are very well studied topics in graph theory, combinatorics and theoretical computer science. A central goal in this line of research is to decide how well a given graph can be partitioned into smaller parts. Generally speaking, a partitioning is considered good if the graph is decomposed into reasonably sized components while removing only a small number of vertices or edges. Specific variants of the graph partitioning question are addressed by a number of well known problems – such as Vertex Separator, Sparsest Cut and Balanced Separator – which have been studied extensively in several previous works [18, 16, 17, 5, 15].

Related to the above is the *Bipartite Expansion* problem which measures the vertex expansion of subsets on one of the sides of a bipartite graph. Specifically, given a bipartite graph $G(U, V, E)$, the goal is to find a subset $V' \subseteq V$ of size at least $\beta|V|$ to minimize $|N(V')|$ for some parameter $\beta \in (0, 1)$, where $N(V')$ is the neighborhood of V' in U. The absence of V' with a small neighborhood implies that there is an edge between any two large enough subsets – one each of U and V, presenting a bottleneck to a good partitioning of the graph. Such graphs are known as *bipartite expanders*. They have been studied in several applications such as parallel sorting [22, 2], constructing good codes [26], randomness extractors [12, 25] and more recently for constructing secure public key cryptographic systems [4]. Since vertex expansion is a submodular function, Bipartite Expansion is a special case of minimizing a submodular function subject to a cardinality constraint, for which Svitkina and Fleischer [27] gave an $O\left(\sqrt{n/\ln n}\right)$ pseudo-approximation. In spite of this being a natural problem, we are not aware of any hardness of approximation results for Bipartite Expansion.

In this work we establish the following strong hardness of approximation result for Bipartite Expansion under the assumption that NP does not have subexponential time algorithms.

▶ **Theorem 1.1** (Main). *For any constants $\gamma, \tau > 0$, there is no algorithm which, given a constant $\beta > 0$ and an n-vertex bipartite graph $G(U, V, E \subseteq U \times V)$, runs in $O(n^c)$ time where $c = c(\tau, \gamma, \beta)$ and decides between the following cases,*

- (YES case) *There is a subset $S^* \subseteq V$ s.t. $|S^*| \geq \beta|V|$ satisfying $|N(S^*)| \leq \gamma|U|$, or*
- (NO case) *Any subset $S \subseteq V$ s.t. $|S| \geq \tau\beta|V|$ satisfies $|N(S)| \geq (1-\gamma)|U|$,*

unless $\mathrm{NP} \subseteq \cap_{\varepsilon>0}\mathrm{DTIME}\left(2^{n^{\varepsilon}}\right)$.

More concretely, the above shows (for example) that even if there exists a good subset V^* of at least β fraction of the vertices of V such that its neighborhood is 1% of U, it is hard to find V' of $\beta/100$ fraction of the vertices of V, such that the neighborhood of V' is at most 99% of U.

We note that Bipartite Expansion problem seems to bear resemblance to the Small Set Expansion problem which has recently received attention due to its connection to Khot's [13] Unique Games Conjecture. This connection was established by Raghavendra and Steurer [23] who proved that the Small Set Expansion Conjecture (see [23] for the statement) implies the Unique Games Conjecture. Theorem 1.1 is, in some sense, a vertex expansion analogue of the statement of the Small Set Expansion Conjecture which deals with edge expansion. Louis, Raghavendra, and Vempala [19] have shown that the Small Set Expansion Conjecture implies hardness of approximation for a variant of vertex expansion on general graphs. While a similar reverse reduction from vertex expansion to edge expansion of small sets is not known, this raises the intriguing possibility that techniques similar to those of this work could throw some light on the Small Set Expansion and Unique Games conjectures.

Our result also serves as a complexity theoretic lower bound for the $O\left(\sqrt{n/\ln n}\right)$ pseudo-approximation of [27] for minimizing submodular functions subject to a cardinality constraint, even when the function is monotone like bipartite vertex expansion.

1.1 Related Work

Bipartite Expansion is related to graph partitioning problems including Vertex Separator and Balanced Separator, and is known to be NP-hard via a reduction from the Balanced Vertex Separator problem [21].

Leighton and Rao [16] gave an $O(\log n)$ (pseudo-)approximation for Balanced Separator and Vertex Separator problems. For Balanced Separator, the factor was improved to the currently best known $O(\sqrt{\log n})$ in a seminal work of Arora, Rao, and Vazirani [5]. Subsequent work by Feige *et al.* [11] and Agarwal *et al.* [1] also proved $O(\sqrt{\log n})$ approximation for Vertex Separator. For both these problems PTAS was ruled out under standard complexity assumptions by Ambuhl, Mastrolilli, and Svensson [3] using the quasi-random Probabilistically Checkable Proof (PCP) of Khot [14]. Subsequently work of Raghavendra, Steurer, and Tulsiani [24] has ruled out a constant factor approximation for Balanced Separator based on the Small Set Expansion Conjecture [23] which implies the Unique Games Conjecture of Khot [13]. The latter conjecture was used in previous works [15, 7] to prove similar inapproximability for a non-uniform version of Balanced Separator. As mentioned above, the work of Svitkina and Fleischer [27] shows that Bipartite Expansion admits a $O\left(\sqrt{n/\ln n}\right)$ pseudo-approximation: given the existence of a subset V^* of size $\beta|V|$ and $|N(V^*)| \le \gamma|U|$, the algorithm outputs a subset V' of size at least $\sigma\beta|V|$ and $|N(V')| \le \rho\gamma|U|$, with $\rho/\sigma \le O\left(\sqrt{n/\ln n}\right)$.

While not much is known about the inapproximability of Bipartite Expansion, the problem of explicitly constructing bipartite expanders has been fairly well studied [20, 22, 2]. These constructions and their variants have applications in sorting networks [22, 2], error-correcting codes [26] and randomness extractors [12, 25]. We end with a brief mention of a result by Applebaum, Barak, and Wigderson [4] who construct public-key encryption schemes based on the (assumed) average case hardness of detecting a random unbalanced bipartite graph from one which has a randomly planted "shrinking" set S of $O(\log n)$ vertices on the larger side such that $|N(S)| \le |S|/3$. While the parameters they consider are different from our setting, their work has, in part, motivated this study of bipartite expansion.

1.2 Our Techniques

The starting point of our reduction is the quasi-random PCP constructed by Khot [14]. Unlike previously constructed PCPs, Khot's construction essentially showed that the YES and NO cases differ in how randomly the queries of the verifier's tests are distributed over the locations of the proof. This crucial quasi-randomness property – which we describe below – was used by Khot [14] to rule out PTAS for Min-Bisection, Dense k-Subgraph and Bipartite Clique, results which were only known earlier assuming the average case hardness of Random-3SAT [10].

The construction in [14] proceeded by (i) proving the inapproximability of an Homogeneous Algebraic CSP over a large field, (ii) transforming the latter into an *Outer Verifier* based on an algebraic test, and (iii) composing the Outer Verifier with a d-query *Inner Verifier* based on the Hadamard Encoding over $\mathbb{F}[2]$. The quasi-randomness finally obtained by this series of reductions can be roughly summarized as follows: in the YES case there is a subset of half the locations of the proof which contains all the d queries of $\approx 1/2^{d-1}$ fraction of tests of the Inner Verifier, while in the NO case any such subset of the proof locations completely contains $\approx 1/2^d$ fraction of the tests. Taking the locations of the proof on the LHS, the tests of the verifier on the RHS, and connecting a location with a test if it queries the former, this already gives us an instance of Bipartite Expansion with a small hardness factor.

However, the inapproximability obtained above is far too weak for us to amplify the hardness gap using (say) graph powering. For this we modify the construction of [14] to encode the proof of the Inner Verifier using a Hadamard Code over a larger field $\mathbb{F}[q]$ where $q \gg 2$. A similar abstraction of the Inner Verifier as a bipartite graph $G(U, V, E)$ yields a $\eta\delta$ versus η gap in the expansion of similar sized subsets of V, for arbitrarily small $\eta, \delta > 0$. Taking the bipartite kth graph power using OR-product of the edges, where $k \approx C/\eta$, amplifies the above to a $C\delta$ versus $(1 - \exp(-C))$ gap in the expansion. The modified quasi-random PCP also yields a gap in the sizes of the relevant subsets of V which is preserved by the powering operation. This, along with the expansion gap, is sufficient to prove Theorem 1.1.

The construction and the analysis of the modified quasi-random PCP proceed along the same lines as in [14]. The parameters of the construction are set appropriately so that the subsequent OR-product graph powering amplifies the gap as desired.

Organization of the paper. The next section formally defines Bipartite Expansion as a decision problem, and restates our hardness result as Theorem 2.2. Section 3 provides the description of the Homogeneous Algebraic CSP and the quasi-random PCP of Khot [14], along with the statement (Theorem 3.4) of the aforementioned modified quasi-random PCP. Section 4 is devoted to proving Theorem 2.2 starting from Theorem 3.4 and the construction of the modified quasi-random PCP, i.e. the proof of Theorem 3.4, is given in Section 5.

2 Our Results

Bipartite Expansion is defined as the following decision problem.

▶ **Definition 2.1.** For parameters $\tau, \gamma, \beta > 0$, the BipartiteExpansion (τ, γ, β) problem is: given a bipartite graph $G(U, V, E \subseteq U \times V)$ distinguish between the following cases.
- (YES Case) There is a subset $S^* \subseteq V$, $|S^*| \geq \beta|V|$ s.t. $|N(S^*)| \leq \gamma|U|$.
- (NO Case) For any subset $S \subseteq V$ s.t. $|S| \geq \tau\beta|V|$, $|N(S)| \geq (1-\gamma)|U|$.

We prove the following hardness of BipartiteExpansion which implies Theorem 1.1.

▶ **Theorem 2.2.** *For any choice of constants $\varepsilon, \tau, \gamma > 0$, there exists $\beta > 0$, such that there is a* DTIME $(2^{n^{\varepsilon}})$ *reduction from* SAT *to* BipartiteExpansion (τ, γ, β).

3 Preliminaries

3.1 The HomAlgCSP Problem

▶ **Definition 3.1.** Let an HomAlgCSP instance $\mathcal{A}(k, d, m, \mathbb{F}, \mathcal{C})$ be the following problem:
1. $\mathcal{C}$ is a system of constraints on functions $f : \mathbb{F}^m \mapsto \mathbb{F}$ where every constraint is on values of f on k different points and is given by a conjunction of homogeneous linear constraints on those k values. A constraint $C \in \mathcal{C}$ on $f(\overline{p}_1), \dots, f(\overline{p}_k)$ is is given as

$$\sum_{i=1}^{k} \gamma_{ij} f(\overline{p}_i) = 0 \quad \text{for } j = 1, 2, \dots \qquad \text{where} \overline{p}_i \in \mathbb{F}^m \text{ and } \gamma_{ij} \in \mathbb{F}.$$

   We denote a constraint C by the set of points $\{\overline{p}_i\}_{i=1}^{k}$, while the γ_{ij}'s will be implicit.
2. $\mathcal{C}$ has $|\mathbb{F}|^{O(m)}$ constraints.

The goal is to find a m-variate polynomial f of total degree at most d, not identically zero, so as to maximize the fraction of constraints satisfied.

The following inapproximability of HomAlgCSP was shown in [14] following from Theorems 1.5 and 3.4 proved therein.

▶ **Theorem 3.2.** *There is a universal constant $\Delta > 0$, such that for any constant $K > 0$ and any constant $\tilde{d} > 0$ possibly depending on K, there is a reduction from a* SAT *formula of size n to an instance $\mathcal{A}(k, d^* = 10\tilde{d}, m = O(\tilde{m}^3\tilde{d}), \mathbb{F}, \mathcal{C})$ of* HomAlgCSP *with $k = 21$, $N \leq |\mathbb{F}| \leq N^2$ where $N = n^{\Delta K}$, and any choice of $\tilde{m}$ satisfying $\binom{\tilde{m}}{\tilde{d}} \geq N$. The size of the instance $\mathcal{A}$ is $|\mathbb{F}|^{O(m)}$, where the field $\mathbb{F}$ is any suitably sized extension of $\mathbb{F}[2]$. The reduction is a* DTIME $(|\mathbb{F}|^{O(m)})$ *procedure*[1] *such that,*

[1] The work of Khot [14] was based on a randomized hardness reduction for *Minimum Distance of Codeword* [9], which can be made deterministic using subsequent results of Cheng and Wan [8] and Austrin and Khot [6].

1. (YES Case) *If the* SAT *formula is satisfiable then there is a degree d^* multivariate polynomial f, not identically zero, which satisfies $1 - 1/2^K$ fraction of constraints of $\mathcal{A}$.*
2. (NO Case) *If the* SAT *formula is unsatisfiable then no degree $1000d^*$ multivariate polynomial, which is not identically zero, satisfies more than $1/2^{2^K}$ fraction of constraints of $\mathcal{A}$.*

3.2 Quasi-Random PCP of Khot [14]

The following is the statement of Khot's quasi-random PCP.

▶ **Theorem 3.3** (Theorem 1.9 of [14]). *For every $\varepsilon > 0$, there exists an integer $d = O(1/\varepsilon \log(1/\varepsilon))$ such that the following holds : there is a PCP verifier for a* SAT *instance of size n satisfying:*
1. *The proof for the verifier is of size $2^{O(n^\varepsilon)}$.*
2. *The verifier uses $O(n^\varepsilon)$ random bits, runs in time $2^{O(n^\varepsilon)}$, and reads d locations from the proof. Let Q be the d locations queried by the verifier in a random test.*
3. *Every query location is uniformly distributed over the proof, though different query locations within Q are correlated.*
4. (YES Case) *Suppose that the* SAT *instance is satisfiable. Then there exists a subset Π^* of half the locations of the proof such that,*

$$\Pr_Q [Q \subseteq \Pi^*] \geq \frac{1}{2^{d-1}} \left(1 - O\left(\frac{1}{d}\right)\right),$$

where the probability is taken over a random test of the verifier.

5. (NO Case) *Suppose that the* SAT *instance is unsatisfiable, and let Π' be any set of half the locations in the proof. Then,*

$$\left| \Pr_Q [Q \subseteq \Pi'] - \frac{1}{2^d} \right| \leq \frac{1}{2^{20d}}.$$

3.3 Modified Quasi-Random PCP

As discussed in Section 1.2, for our hardness result we construct the quasi-random PCP with an Inner Verifier encoding over a large field $\mathbb{F}[q]$. While the details of the construction and its analysis are given in Section 5, here we abstract out the bounds on the distribution of the PCP queries required for our purposes.

▶ **Theorem 3.4.** *For every positive integer (power of two) $R > 2$, and arbitrarily small $\varepsilon > 0$, there exists an integer $d = \Theta((1/\varepsilon)\log((\log R)/\varepsilon))$ along with the setting $q := R^{4d}$, such that the following holds : there is a PCP verifier for a* SAT *instance of size n satisfying properties (1)-(3) of Theorem 3.3 along with,*

4. (YES Case) *Suppose that the* SAT *instance is satisfiable. Then there exists a subset Π^* of $1/q$ fraction of the locations of the proof, such that*

$$\Pr_Q [Q \subseteq \Pi^*] \geq \frac{1}{q^{d-1}} \left(1 - O\left(\frac{1}{d^2}\right)\right), \tag{1}$$

where the probability is taken over a random test of the verifier.

5. (NO Case) *Suppose that the* SAT *instance is unsatisfiable, and let Π' be any set of $\zeta \in [0, 1]$ fraction of the locations of the proof. Then,*

$$\left| \Pr_Q [Q \subseteq \Pi'] - \zeta^d \right| \leq \frac{1}{q^{2d^2}}. \tag{2}$$

4 Reducing the Modified Quasi-Random PCP to BipartiteExpansion

For convenience we first abstract out the Modified Quasi-Random PCP as a bipartite graph and translate its YES and NO cases into a gap in expansion of small subsets on one side of the bipartition. This gap in expansion is then strengthened using an appropriate powering of the initial bipartite graph to yield the desired hardness for BipartiteExpansion. We assume for the rest of this section that that the parameters R, d and q in Theorem 3.4 are large enough constants.

4.1 Modified Quasi-Random PCP as a Bipartite Graph

Starting from an instance of the Modified Quasi-Random PCP in Theorem 3.4 define the bipartite graph $G(U, V, E \subseteq U \times V)$ where U is the set of proof locations, V is the set of d-query tests of the verifier and $(u, v) \in E$ iff the test v contains the query location u. Restating the YES and NO cases in terms of expansion of subsets of V we have the following lemmas.

▶ **Lemma 4.1.** *If G is a* YES *instance then there is a subset $S^* \subseteq V$ of size at least $0.99|V|/q^{d-1}$ such that $|N(S^*)| \leq |U|/q$.*

Proof. Take S^* to be the set of tests completely contained in the $1/q$ fraction of the proof locations given by the YES case. The lemma follows from (1) and large enough d. ◀

▶ **Lemma 4.2.** *If G is a* NO *instance then for any subset $S \subseteq V$ s.t. $|S| = a|V|$ where $a \in [1/q^{2d^2}, 1]$,*

$$\frac{|N(S)|}{|U|} \geq a^{1/d} - \frac{1}{aq^{2d^2}}.$$

Proof. In the NO case we let $\zeta = \frac{|N(S)|}{|U|}$, and thus from (2) we have

$$a \leq \zeta^d + \frac{1}{q^{2d^2}} \quad \Rightarrow \quad \zeta^d \geq a - \frac{1}{q^{2d^2}} \geq 0,$$

since $a \geq 1/q^{2d^2}$. This implies that,

$$\zeta \geq \left(a - \frac{1}{q^{2d^2}}\right)^{\frac{1}{d}} \geq a^{1/d}\left(1 - \frac{1}{aq^{2d^2}}\right)^{\frac{1}{d}} \geq a^{1/d}\left(1 - \frac{1}{aq^{2d^2}}\right) \geq a^{1/d} - \frac{1}{aq^{2d^2}},$$

since $a^{1/d} \leq 1$. ◀

4.2 Graph powering using OR-product

Fix a parameter $k := R^{4d-1}$. From $G(U, V, E)$ obtained above we construct the bipartite graph $\overline{G}(\overline{U}, \overline{V}, \overline{E} \subseteq \overline{U} \times \overline{V})$ as follows.

- $\overline{U} = U^k$ and $\overline{V} = V^k$. For any $\overline{u} \in \overline{U}$, $j \in [k]$, $\overline{u}_j \in U$ denotes the jth coordinate of $\overline{u}$. Similarly for $\overline{v} \in \overline{V}$.
- $(\overline{u}, \overline{v}) \in \overline{E}$ iff $\exists j \in [k]$ s.t. $(\overline{u}_j, \overline{v}_j) \in E$.

The rest of this section is devoted to proving the desired YES and NO cases completing the proof of Theorem 2.2.

4.2.1 YES Case

We prove the following lemma.

▶ **Lemma 4.3.** *If G is a* YES *instance then there exists a subset $T^* \subseteq \overline{V}$ such that,*

$$|T^*| \geq \left(\frac{0.99}{q^{d-1}}\right)^k |\overline{V}|,$$

and,

$$|N(T^*)| \leq |\overline{U}|/R.$$

Proof. Let $T^* = (S^*)^k$ where S^* is as given in Lemma 4.1. The first condition above is directly satisfied by the bound on the size of S^* in Lemma 4.1. Further, by union bound over all the k coordinates,

$$\frac{|N(T^*)|}{|\overline{U}|} \leq k \cdot \frac{|N(S^*)|}{|U|} \leq \frac{k}{q} = \frac{R^{4d-1}}{R^{4d}} = R^{-1},$$

where $|N(S^*)|/|U| \leq 1/q$ as given in Lemma 4.1. ◀

4.2.2 NO Case

For convenience let $h := 1/q^{d-1/2}$. The NO case is given by the following lemma.

▶ **Lemma 4.4.** *If G is a* NO *instance then for any subset $T \subseteq \overline{V}$ s.t. $|T| \geq h^k|\overline{V}|$,*

$$|N(T)| \geq \left(1 - e^{-R/2}\right)|\overline{U}|.$$

Proof. Let us first define the projections $T_1, \ldots, T_k \subseteq V$ of T as: $T_j = \{v \in V \mid \exists \overline{v} \in T \text{ s.t. } \overline{v}_j = v\}$. By construction, $|T| \leq \prod_{j=1}^k |T_j|$. Let $a_j := |T_j|/|V|$. Thus, we have,

$$\prod_{j=1}^k (a_j|V|) \geq h^k|\overline{V}| = h^k|V|^k,$$

which implies

$$\prod_{j=1}^k a_j \geq h^k. \tag{3}$$

By the AM-GM inequality we have,

$$\mathbb{E}_{j\in[k]}\left[a_j^{1/d}\right] \geq \left(\prod_{j=1}^k a_j^{1/d}\right)^{1/k} = \left(\prod_{j=1}^k a_j\right)^{1/kd} \geq h^{1/d}. \tag{4}$$

We also have the following simple lemma.

▶ **Lemma 4.5.** *For at most k/d values $j \in \{1, \ldots, k\}$, $a_j < h^d$.*

Proof. Assuming that for $t > k/d$ values $j \in \{1, \ldots, k\}$ $a_j < h^d$, we obtain that $\prod_{j=1}^k a_j \leq h^{td} < h^k$ (since $h < 1$), which contradicts (3). ◀

Let us define $b_j := |N(T_j)|/|U|$ for $j \in [k]$. Since, by our setting, $h^d \geq q^{-d^2} \geq q^{-2d^2}$, Lemma 4.2 yields

$$\{a_j \geq h^d\} \Rightarrow \left\{b_j \geq a_j^{1/d} - \frac{1}{a_j q^{2d^2}}\right\} \Rightarrow \left\{b_j \geq a_j^{1/d} - \frac{1}{q^{d^2}}\right\}. \tag{5}$$

Therefore,

$$\begin{aligned}\sum_{j=1}^{k} b_j \geq \sum_{\substack{j\in[k]\\ a_j \geq h^d}} \left(a_j^{1/d} - \frac{1}{q^{d^2}}\right) &= \sum_{j=1}^{k}\left(a_j^{1/d} - \frac{1}{q^{d^2}}\right) - \sum_{\substack{j\in[k]\\ a_j < h^d}} \left(a_j^{1/d} - \frac{1}{q^{d^2}}\right) \\ &\geq \sum_{j=1}^{k}\left(a_j^{1/d} - \frac{1}{q^{d^2}}\right) - \sum_{\substack{j\in[k]\\ a_j < h^d}} a_j^{1/d} \\ &\geq \sum_{j=1}^{k}\left(a_j^{1/d} - \frac{1}{q^{d^2}}\right) - \sum_{\substack{j\in[k]\\ a_j < h^d}} h \\ &\geq \sum_{j=1}^{k}\left(a_j^{1/d} - \frac{1}{q^{d^2}}\right) - \left(\frac{k}{d}\right) h,\end{aligned}$$

where the last inequality uses Lemma 4.5. Taking an expectation we obtain,

$$\mathbb{E}_{j\in[k]}\left[b_j\right] \;\geq\; \mathbb{E}_{j\in[k]}\left[a_j^{1/d}\right] - \frac{1}{q^{d^2}} - \frac{h}{d} \;\geq\; h^{1/d} - \frac{1}{q^{d^2}} - \frac{h}{d} \;\geq\; h^{1/d}/2, \tag{6}$$

where second last inequality follows from (4) and the last inequality is due to the large enough setting of the parameters. Observe that,

$$h^{1/d} = \left(q^{-(d-1/2)}\right)^{1/d} = \left(R^{-4d(d-1/2)}\right)^{1/d} = R^{-4d+2}.$$

Using the above along with the construction of $\overline{G}$ we have,

$$\begin{aligned}1 - \frac{|N(T)|}{|\overline{U}|} &= \prod_{j=1}^{k}(1-b_j) \\ &\leq \left(\mathbb{E}_{j\in[k]}[1-b_j]\right)^k && \text{(By the AM-GM inequality)} \\ &\leq \left(1 - h^{1/d}/2\right)^k && \text{(Using (6))} \\ &\leq \left(1 - \frac{1}{2R^{4d-2}}\right)^{R^{4d-1}} \quad \leq\; e^{-R/2},\end{aligned}$$

which completes the proof of Lemma 4.4. ◀

4.2.3 Gap in the domain subset sizes

In the YES case there is a subset of $\overline{V}$ of fractional size at least

$$\beta := \left(\frac{0.99}{q^{d-1}}\right)^k \tag{7}$$

with neighborhood size at most $R^{-1}|\overline{U}|$, while in the NO case every subset of $\overline{V}$ of fractional size at least

$$h^k = \left(\frac{1}{q^{d-1/2}}\right)^k \tag{8}$$

has neighborhood of size at least $(1 - e^{-R/2})|\overline{U}|$. The subset size threshold in the NO case is much smaller than in the YES case with the gap being,

$$\left(\frac{\beta}{h^k}\right) = \left(\frac{0.99q^{d-1/2}}{q^{d-1}}\right)^k \geq q^{k/3} \geq e^R,$$

for large enough R, q, d which we may assume.

4.2.4 Setting the parameters and proof of Theorem 2.2

Given ε, τ and $\gamma > 0$, choose R large enough so that $\gamma \geq \max\{R^{-1}, e^{-R/2}\}$, and $\tau \geq e^{-R}$. Setting $d = \Theta((1/\varepsilon)\log((\log R)/\varepsilon))$ as per Theorem 3.4 along with Lemmas 4.3, Lemma 4.4, and Section 4.2.3 yields the proof of Theorem 2.2 with β given by (7).

5 Construction of the Quasi-Random PCP

The PCP given in Theorem 3.4 is a composition of an Outer Verifier which is an algebraic test on an instance of HomAlgCSP, with a Hadamard code based encoding (Inner Verifier). This is almost the same as the construction of [14], except that the Inner Verifier's encoding is over a larger field rather than $\mathbb{F}[2]$. We refer the reader to [14] for motivation behind this construction and its nuances, and instead give a concise description of the PCP and its analysis.

Let the HomAlgCSP instance be $\mathcal{A}(k = 21, d^*, m, \mathbb{F}, \mathcal{C})$. The Outer Verifier is given the polynomial f as a table of values at each point in $\mathbb{F}^m$, and it samples a constraint from $\mathcal{C}$ uniformly at random and attempts to verify whether it is satisfied by f, and whether the table of f is a polynomial of degree $\approx d^*$. We need the following definition of a *curve*.

▶ **Definition 5.1.** A *curve* L in $\mathbb{F}^m$ is a function $L : \mathbb{F} \mapsto \mathbb{F}^m$, where $L(t) = (a_1(t), \ldots, a_m(t))$. It is of degree d if each of the coordinate functions a_i is degree d (univariate) polynomial. A *line* is a curve of degree 1.

Let $t_1, t_2, \ldots, t_{k+3}$ be distinct field elements in $\mathbb{F}$ which we fix for the rest of the construction. Suppose the verifier chooses the constraint $C(\{\overline{p}_i\}_{i=1}^k) \in \mathcal{C}$ uniformly at random. For $\overline{a}, \overline{b}, \overline{c} \in \mathbb{F}^m$, define $L = L_{\overline{a},\overline{b},\overline{c}}$ be the unique degree $(k+2)$ curve that satisfies

$$L(t_i) = \overline{p}_i, \quad 1 \leq i \leq k, \quad L(t_{k+1}) = \overline{a}, \quad L(t_{k+2}) = \overline{b}, \quad L(t_{k+3}) = \overline{c}.$$

If f is a degree d^* multivariate polynomial over the vector space $\mathbb{F}^m$ then its restriction to the curve $L(t) = L_{\overline{a},\overline{b},\overline{c}}(t)$, denoted by $f|_L$, is a degree $d - 1 := (k+2)d^*$ univariate polynomial in t. This polynomial can be interpolated from any d values of f on the curve, which is then used to test its consistency at an additional random point. Similarly, given a line ℓ, the restriction of f, denoted by $f|_\ell$ is a degree d^* univariate polynomial. Allowing it degree up to $(d-2)$ it is interpolated using the values of f at $(d-1)$ random points on ℓ, which is used to run the Low Degree test. The following is the description of the Outer Verifier.

5.1 Outer Verifier

Steps of the Outer Verifier

1. Pick a constraint $C = \{\overline{p}_i\}_{i=1}^{k} \in \mathcal{C}$ at random.
2. Pick a random line ℓ in $\mathbb{F}^m$ and pick random points $\overline{v}_1, \ldots, \overline{v}_{d-1}, \overline{v}_d$ on the line.
3. Pick $t \in \mathbb{F} \setminus \{t_1, \ldots, t_{k+3}\}$ at random, points $\overline{a}, \overline{b}$ at random from $\mathbb{F}^m$ and let L be the unique degree $k+2$ curve $L = L_{\overline{a},\overline{b},\overline{c}}$ such that,
$$L(t_i) = \overline{p}_i, \quad 1 \le i \le k, \quad L(t_{k+1}) = \overline{a}, \quad L(t_{k+2}) = \overline{b}, \quad L(t) = \overline{v}_d,$$
and $\overline{c}$ is implicitly defined as $L(t_{k+3})$.
4. Pick random points $\overline{v}_{d+1}, \ldots, \overline{v}_{2d}$ on L.
5. Let $f|_\ell$ be the unique degree $d-2$ polynomial interpolated using the values $\{f(\overline{v}_i)\}_{i=1}^{d-1}$.
6. Let $f|_L$ be the unique degree $d-1$ polynomial interpolated using the values $\{f(\overline{v}_i)\}_{i=d+1}^{2d}$.
7. Check if,
$$f|_L(\overline{v}_d) = f(\overline{v}_d) = f|_\ell(\overline{v}_d).$$
8. Check if the values of $f|_L$ at points $\{\overline{p}_i\}_{i=1}^{k}$ satisfy the constraint C.
9. Check that the values $f(\overline{v}_i)$, $1 \le i \le 2d$ are not all zero.

As in [14], the Outer Verifier can be replaced by the following *Modified Outer Verifier* which reads more values from the proof and makes additional tests, and additionally abstracts out: (i) interpolation into multiplication by an invertible matrix, and (ii) checking the homogeneous constraints of the Outer Verifier into checking orthogonality with a certain subspace. Our construction is the same, except that instead of $\mathbb{F}[2]$ we shall use an extension field $\mathbb{F}[q]$ as the underlying field of representation, where q is as given in Theorem 3.4 and $\mathbb{F}$ in Theorem 3.2 is chosen to be an extension of $\mathbb{F}[q]$.

5.2 Modified Outer Verifier

Since $\mathbb{F}$ is an extension of $\mathbb{F}[q]$ the elements of $\mathbb{F}$ are represented as $\mathbb{F}[q]$-vectors of a length $l = (\log|\mathbb{F}|)/(\log q)$. Moreover, the representation can be chosen such that addition over $\mathbb{F}$ and multiplication by a constant in $\mathbb{F}$ are homogeneous linear operations on these vectors. The Modified Outer Verifier is given a table of values $f(\overline{v})$ (in the form of l length $\mathbb{F}[q]$-vectors) for every point $\overline{v} \in \mathbb{F}^m$ and it executes the following steps:

Steps of the Modified Outer Verifier

1. Pick a constraint $C = \{\overline{p}_i\}_{i=1}^{k} \in \mathcal{C}$ at random.
2. Pick a random line ℓ in $\mathbb{F}^m$ and pick random points $\overline{v}_1, \ldots, \overline{v}_{d-1}, \overline{v}_d$ on the line.
3. Pick $t \in \mathbb{F} \setminus \{t_1, \ldots, t_{k+3}\}$ at random, points $\overline{a}, \overline{b}$ at random from $\mathbb{F}^m$ and let L be the unique degree $k+2$ curve $L = L_{\overline{a},\overline{b},\overline{c}}$ such that,
$$L(t_i) = \overline{p}_i, \quad 1 \le i \le k, \quad L(t_{k+1}) = \overline{a}, \quad L(t_{k+2}) = \overline{b}, \quad L(t) = \overline{v}_d.$$
and $\overline{c}$ is implicitly defined to be $L(t_{k+3})$.
4. Pick random points $\overline{v}_{d+1}, \ldots, \overline{v}_{2d}$ on the curve L.
5. Pick additional random points $\overline{u}_1 \ldots \overline{u}_d$ on the line ℓ and $\overline{u}_{d+1}, \ldots, \overline{u}_{2d}$ from the curve L.
6. Let $T_{2ld \times 2ld}$ be an appropriate invertible matrix over $\mathbb{F}[q]$ and H be an appropriate subspace of $\mathbb{F}[q]^{2ld}$. Both depend only on the choice of the points $\{\overline{v}_i\}_{i=1}^{2d}$ and $\{\overline{u}_j\}_{j=1}^{2d}$. Remark 5.2 explains how T and H are chosen.

7. Read the values of the function f from the table at the points $\overline{v}_1, \dots, \overline{v}_{2d}$ and $\overline{u}_1, \dots, \overline{u}_{2d}$. Since the values are represented by length l $\mathbb{F}[q]$-vectors, let
$$x = f(\overline{v}_1) \circ f(\overline{v}_2) \circ \cdots \circ f(\overline{v}_{2d}) \tag{9}$$
$$y = f(\overline{u}_1) \circ f(\overline{u}_2) \circ \cdots \circ f(\overline{u}_{2d}) \tag{10}$$
where $\circ$ represents concatenation of vectors.
8. Accept iff,
$$x \neq 0, \; x = Ty \qquad \text{and} \qquad h \cdot x = 0 \quad \forall\, h \in H \quad (\text{i.e. } x \perp H). \tag{11}$$

▶ Remark 5.2. The choice of H is such that $h \cdot x = 0 \;\forall\; h \in H$ abstracts out the conditions: (i) the values at the field elements $\{t_i\}_{i=1}^k$ of the degree $d-1$ univariate polynomial interpolated from $f(\overline{v}_{d+1}) \dots f(\overline{v}_{2d})$ satisfy the homogeneous linear constraints of C, and (ii) the polynomial interpolated from the values $f(\overline{v}_1) \dots f(\overline{v}_{d-1})$ agrees with the degree $d-1$ polynomial interpolated from $f(\overline{v}_{d+1}) \dots f(\overline{v}_{2d})$ at the point $\overline{v}_d$, where both evaluate to $f(\overline{v}_d)$.

The invertible matrix T is chosen such that the constraint $x = Ty$ abstracts out the conditions: (i) the degree $d-1$ polynomial interpolated from the values $f(\overline{v}_1) \dots f(\overline{v}_d)$ is the same as the polynomial interpolated from the values $f(\overline{u}_1) \dots f(\overline{u}_d)$, and (ii) the degree $d-1$ polynomial interpolated from $f(\overline{v}_{d+1}) \dots f(\overline{v}_{2d})$ is the same as the polynomial interpolated from the values $f(\overline{u}_{d+1}) \dots f(\overline{u}_{2d})$.

The condition $x \neq 0$ essentially ensures that f is not a zero polynomial.

The following theorem, regarding the acceptance probability of the Outer Verifier, was proved in [14].

▶ **Theorem 5.3.** *There are constants c_1, c_2 such that the following holds. If, after picking a constraint $C(\{\overline{p}_i\}_{i=1}^k) \in \mathcal{C}$, the Outer Verifier (or the Modified Outer Verifier) accepts with probability δ, then for $1 \leq t \leq 2c_2/(\delta/2)^{c_1}$, $P_1, P_2, \dots, P_t$ are all the degree d polynomials that have agreement at least $(\delta/2)^{c_1}/c_2$ with f and for some $1 \leq j \leq t$, P_j is a non-zero polynomial whose values at the points $\{\overline{p}_i\}_{i=1}^k$ satisfies the constraint C.*

5.3 Inner Verifier

The Inner Verifier expects, for every point $\overline{v} \in \mathbb{F}^m$, the Hadamard Code of $f(\overline{v}) \in \mathbb{F}[q]^l$. (See Section 7 for a description of the Hadamard Code).

Steps of the Inner Verifier

1. Pick a constraint $C \in \mathcal{C}$ and the points $\overline{v}_1, \dots, \overline{v}_{2d}$ and $\overline{u}_1, \dots, \overline{u}_{2d}$ as in steps $1-5$ of the Modified Outer Verifier.
2. Let $T_{2ld \times 2ld}$ and H be the matrix and subspace respectively chosen as in step 7 of the Modified Outer Verifier.
3. Pick a random string $z \in (\mathbb{F}[q]^l)^{2d}$ and a random $h \in H$. Write,
$$z = z_1 \circ z_2 \circ \cdots \circ z_{2d}$$
$$h = h_1 \circ h_2 \circ \cdots \circ h_{2d}$$
$$zT = w_1 \circ w_2 \circ \cdots \circ w_{2d}.$$
4. Let $A_1, \dots, A_{2d}$ and $B_1, \dots, B_{2d}$ be the tables giving the supposed Hadamard Codes of $f(\overline{v}_1), \dots, f(\overline{v}_{2d})$ and $f(\overline{u}_1), \dots, f(\overline{u}_{2d})$ respectively.
5. Accept iff $\sum_{i=1}^{2d} A_i(z_i + h_i) + \sum_{j=1}^{2d} B_j(w_j) = 0$.

5.4 Analysis

Let Π_* be a subset of locations of the proof Π given to the Inner Verifier. Setting the locations of Π_* to be 1 and the rest of the locations to zero, we obtain the tables A_i and B_i ($1 \leq j \leq 2d$) which are queried in the description of the Inner Verifier. We wish to analyze the probability over a random test Q of the Inner Verifier that the locations queried by it are contained inside $\Pi*$, i.e.

$$\Pr_Q \left[Q \subseteq \Pi_* \right].$$

This is first arithmetized to,

$$\mathbb{E}_Q \left[\prod_{i=1}^{2d} A_i(z_i + h_i) \prod_{j=1}^{2d} B_j(w_j) \right]. \tag{12}$$

Here Q depends on the choice of the constraint C, the line ℓ and curve L, the points $\overline{v}_1, \ldots, \overline{v}_{2d}, \overline{u}_1, \ldots, \overline{u}_{2d}$, and the choice of z and h. Plugging in the Fourier expansion (see Section 6) of the A_i and B_i we obtain,

$$\begin{aligned} & \mathbb{E}_Q \left[\sum_{\alpha_1,\ldots,\alpha_{2d},\beta_1,\ldots,\beta_{2d}} \left[\prod_{i=1}^{2d} \widehat{A}_{i,\alpha_i} \prod_{j=1}^{2d} \widehat{B}_{j,\beta_j} \prod_{i=1}^{2d} \chi_{\alpha_i}(z_i + h_i) \prod_{j=1}^{2d} \chi_{\beta_j}(w_j) \right] \right] \\ = \quad & \mathbb{E}_Q \left[\sum_{\substack{\alpha=\alpha_1\circ\cdots\circ\alpha_{2d}, \\ \beta=\beta_1\circ\cdots\circ\beta_{2d}}} \left[\prod_{i=1}^{2d} \widehat{A}_{i,\alpha_i} \prod_{j=1}^{2d} \widehat{B}_{j,\beta_j} \cdot \phi(\alpha \cdot z + \beta \cdot w) \cdot \phi(\alpha \cdot h) \right] \right], \end{aligned} \tag{13}$$

where $\phi : \mathbb{F}[q] \to \{-1, 1\}$ is defined in Section 6. Now, since h is randomly chosen from H, the above expectation is zero unless $\alpha \perp H$. Also,

$$\begin{aligned} z \cdot \alpha + w \cdot \beta &= z \cdot \alpha + zT \cdot \beta \\ &= z \cdot (\alpha + T\beta) \end{aligned}$$

which implies that the expectation in (13) is zero unless $\alpha = T\beta$, since z is chosen randomly from $\mathbb{F}[q]^{2ld}$. Therefore we obtain the following expression,

$$\mathbb{E}_{\substack{C,\ell,L,\overline{v}_1,\ldots,\overline{v}_{2d} \\ \overline{u}_1,\ldots,\overline{u}_{2d}}} \left[\sum_{\substack{\alpha=\alpha_1\circ\cdots\circ\alpha_{2d}, \\ \beta=\alpha_1\circ\cdots\circ\beta_{2d}, \\ \alpha\perp H,\ \beta=T^{-1}\alpha}} \left[\prod_{i=1}^{2d} \widehat{A}_{i,\alpha_i} \prod_{j=1}^{2d} \widehat{B}_{j,\beta_j} \right] \right]. \tag{14}$$

5.4.1 YES Case

We prove the following lemma.

▶ **Lemma 5.4.** *If the instance $\mathcal{A}$ of* HomAlgCSP *is a* YES *instance then there exists a subset Π_* of $1/q$ fraction of the proof locations such that*

$$\Pr_Q \left[Q \subseteq \Pi_* \right] \geq \frac{1}{q^{4d-1}} \left(1 - \frac{1}{2^K} \right).$$

Proof. Let f be the polynomial given by the YES case. Since f is of degree at most d^*, it is nonzero at all points except for a negligible fraction ($O(d^*/|\mathbb{F}|)$) which we ignore. Construct the proof Π as follows: For each point $\overline{v} \in \mathbb{F}^m$ let the corresponding table $A_{\overline{v}}$ be defined as:

$$A_{\overline{v}}(x) = \begin{cases} 1 & \text{if the Hadamard Code of } f(\overline{v}) \text{ at location } x \text{ is } 0 \in \mathbb{F}[q], \\ 0 & \text{otherwise.} \end{cases} \tag{15}$$

Now, let Π_* be the subset of locations where Π is 1. Since we are dealing with Hadamard Codes of nonzero values, Π_* is exactly $(1/q)$ fraction of the locations. Also, from Lemma 7.1,

$$A_{\overline{v}} = (1/q) \sum_{t \in \mathbb{F}[q]} \chi_{tf(\overline{v})},$$

where $f(\overline{v})$ in the is represented as an element of $\mathbb{F}[q]^l$. Using this, we see that when the constraint C is satisfied by f, for each $t \in \mathbb{F}[q]$ setting $\alpha_i = tf(\overline{v}_i)$ and $\beta_i = tf(\overline{u}_i)$ $(1 \leq i \leq 2d$ where the f values are represented as elements of $\mathbb{F}[q]^l$) contributes $1/q^{4d}$ in (14). Since $1 - 1/2^K$ fraction of the constraints are satisfied by f in the YES case of Theorem 3.2 the lemma follows. ◀

5.4.2 NO Case

The NO case soundness is given by the following lemma.

▶ **Lemma 5.5.** *Let Π_* be any subset of $\zeta \in [0, 1]$ fraction of locations of the proof Π. Then, if $\mathcal{A}$ is a* NO *instance then,*

$$\left| \Pr_Q [Q \subseteq \Pi_*] - \zeta^{4d} \right| \leq \frac{C_0}{2^{2^K/C_1}}, \tag{16}$$

for some universal constants $C_0, C_1 > 0$.

Proof. Suppose that,

$$\left| \Pr_Q [Q \subseteq \Pi_*] - \zeta^{4d} \right| = \delta. \tag{17}$$

Let the proof Π evaluate to 1 at the locations in Π^* and zero otherwise. Thus,

$$\mathbb{E}_{\overline{v}} \left[\widehat{A}_{\overline{v}} \right] = \zeta.$$

Using the mixing property of curves and lines (refer to Appendix A.4 of [14]) we obtain that for the random line ℓ and curve L chosen by the Outer Verifier, except with probability $O(1/|\mathbb{F}|^{1/3})$ (which we shall ignore),

$$\forall\ i = 1, \ldots, 2d \quad \mathbb{E}_{\overline{v}_i \in \ell}[\widehat{A}_{i,0}] \approx \zeta \quad , \quad \mathbb{E}_{\overline{u}_j \in \ell}[\widehat{B}_{j,0}] \approx \zeta$$

where again the error in the above approximations is bounded by $O(1/|\mathbb{F}|^{1/3})$ which we shall ignore. Thus, the contribution of $\alpha = 0$ in (14) is (up to negligible error) ζ^{4d}. From (17), an analysis identical to that in Section 10.5 of [14] yields a table of values f such that the Outer Verifier accepts with probability δ^2 over a randomly chosen constraint $C \in \mathcal{C}$ of the HomAlgCSP instance $\mathcal{A}$. Thus, for at least $\delta^2/2$ of the constraints C, the Outer Verifier accepts with probability $\delta^2/2$. Using Theorem 5.3, this implies that there is a degree $d \leq (21+3)d^* \leq 100d^*$ polynomial which satisfies at least $(\delta/C_0)^{C_1}$ fraction of the constraints of $\mathcal{A}$ for some universal constants $C_0, C_1 > 0$. This contradicts the NO case of Theorem 3.2 unless δ is at most the RHS of (16), thus completing the proof of the lemma. We omit further details and refer the reader to [14]. ◀

5.4.3 Setting the parameters

The various parameters of the PCP reduction from HomAlgCSP are set so that Lemmas 5.4 and5.5 along with Theorem 3.2 yield Theorem 3.4. First we change $4d$ to d in Lemmas 5.4 and 5.5. As in Theorem 3.4, $q := R^{4d}$. We set $d := \Theta(2^{K/3}/(\log R))$ so that,

$$\frac{1}{2^K} = O\left(\frac{1}{d^3}\right) \quad \text{and} \quad \frac{C_0}{2^{2^K/C_1}} \leq \frac{1}{q^{2d^2}},$$

appropriately bounding the errors in Lemmas 5.4 and 5.5. Note that $d \leq (21+3)\cdot 10\cdot 4\cdot \tilde{d} \leq 1000\tilde{d}$ (where $\tilde{d}$ is as in Theorem 3.2). Thus, choosing $m = n^{1000\Delta K/d}$ yields $\binom{m}{\tilde{d}} \geq N$ as required, and that the entire reduction runs in time 2^{n^ε} where $\varepsilon = \Theta(K/d)$ can be made arbitrarily small by choosing K large enough. Rearranging, $d = \Theta((1/\varepsilon)\log((\log R)/\varepsilon))$.

6 Fourier Analysis

We will be working over the field $\mathbb{F}[q] := \mathbb{F}[2^r]$ for $r > 0$, which is a field extension of $\mathbb{F}[2]$. Let φ be the isomorphism from the additive group $(\mathbb{F}[2^r], +)$ to $(\mathbb{F}[2]^r, +)$. Define the following homomorphism ϕ from $(\mathbb{F}[2^r], +)$ to the multiplicative group $(\{-1, 1\}, .)$.

$$\phi(a) = \begin{cases} 1 & \text{if } \varphi(a) \text{ contains even number of 1s} \\ -1 & \text{otherwise} \end{cases}$$

for any $a \in \mathbb{F}[2^r]$. Note that $\phi(a+b) = \phi(a)\phi(b)$, $\forall a, b \in \mathbb{F}[2^r]$. We now define the 'characters' $\psi_a : \mathbb{F}[2^r] \mapsto \{-1, 1\}$ for $a \in \mathbb{F}[2^r]$ as follows.

$$\psi_a(b) := \phi(ab)$$

The characters ψ_a satisfy the following properties.

$$\begin{aligned} \psi_0(b) &= 1 & \forall b \in \mathbb{F}[2^r] \\ \psi_a(0) &= 1 & \forall a \in \mathbb{F}[2^r] \\ \psi_{a+b}(c) &= \psi_a(c)\psi_b(c) \end{aligned}$$

and,

$$\sum_{a \in \mathbb{F}[2^r]} \psi_a(b) = \begin{cases} |\mathbb{F}[2^r]| & \text{if } b = 0 \\ 0 & \text{otherwise} \end{cases}$$

We note that the 'character' functions form an orthonormal basis for the space $L^2(\mathbb{F}[2^r])$. We have that,

$$\langle \psi_a, \psi_b \rangle = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{otherwise} \end{cases}$$

where,

$$\langle \psi_a, \psi_b \rangle := \mathbb{E}_{c \in \mathbb{F}[2^r]}\left[\psi_a(c)\psi_b(c)\right].$$

We now consider the vector space $\mathbb{F}[2^r]^m$ for some positive integer m. We define the 'characters' $\chi_\alpha : \mathbb{F}[2^r]^m \mapsto \{-1, 1\}$ for every $\alpha \in \mathbb{F}[2^r]^m$ as,

$$\chi_\alpha(f) := \phi(\alpha \cdot f), \qquad f \in \mathbb{F}[2^r]^m$$

where '$\cdot$' is the inner product in the vector space $\mathbb{F}[2^r]^m$. From the way we defined the characters ψ_a, we have,

$$\chi_\alpha(f) = \prod_{i=1}^{m} \psi_{\alpha_i}(f_i),$$

where α_i and f_i are the i^{th} coordinates of α and f respectively. The characters χ_α satisfy the following properties,

$$\begin{aligned} \chi_0(f) &= 1 && \forall f \in \mathbb{F}[2^r]^m \\ \chi_\alpha(0) &= 1 && \forall \alpha \in \mathbb{F}[2^r]^m \\ \chi_{\alpha+\beta}(f) &= \chi_\alpha(f)\chi_\beta(f) \\ \chi_\alpha(f+g) &= \chi_\alpha(f)\chi_\alpha(g) \end{aligned}$$

and,

$$\mathbb{E}_{f\in\mathbb{F}[2^r]^m}\left[\chi_\alpha(f)\right] = \begin{cases} 1 & \text{if } \alpha = 0 \\ 0 & \text{otherwise} \end{cases}$$

The characters χ_α form an orthonormal basis for $L^2(\mathbb{F}[2^r]^m)$. We have,

$$\langle \chi_\alpha, \chi_\beta \rangle = \begin{cases} 1 & \text{if } \alpha = \beta \\ 0 & \text{otherwise} \end{cases}$$

where,

$$\langle \chi_\alpha, \chi_\beta \rangle := \mathbb{E}_{f\in\mathbb{F}[2^r]^m}\left[\chi_\alpha(f)\chi_\beta(f)\right].$$

Let $A : \mathbb{F}[2^r]^m \mapsto \mathbb{R}$ be any real valued function. Then the Fourier expansion of A is given by,

$$A(x) = \sum_{\alpha\in\mathbb{F}[2^r]^m} \widehat{A}_\alpha \chi_\alpha(x),$$

where,

$$\widehat{A}_\alpha = \mathbb{E}_{x\in\mathbb{F}[2^r]^m}[A(x)\chi_\alpha(x)].$$

A useful equality is:

$$\widehat{A}_0 = \mathbb{E}_{x\in\mathbb{F}[2^r]^m}[A(x)].$$

7 Hadamard Codes

Let l be a positive integer and $\mathbb{F}[q]$ be an extension of $\mathbb{F}[2]$. Then, for any $a \in \mathbb{F}[q]^l$, its Hadamard Code $H_a : \mathbb{F}[q]^l \to \mathbb{F}[q]$ is given by $H_a(x) = a \cdot x = \sum_{i=1}^{l} a_i x_i$. We have the following simple lemma.

▶ **Lemma 7.1.** *For any $a \in \mathbb{F}[q]^l$, let $A : \mathbb{F}[q]^l \to \{0,1\}$ be defined as $A(x) := \mathbb{1}\{H_a(x) = 0\}$. Then,*

$$A = (1/q) \sum_{t\in\mathbb{F}[q]} \chi_{ta}.$$

Proof. If $a = 0$, then A is identically 1, and thus $A = \chi_0 = (1/q)\sum_{t\in\mathbb{F}[q]} \chi_{ta}$. If $a \neq 0$, then $\mathbb{E}_x[A(x)] = \Pr_x[a \cdot x = 0] = 1/q$. Further,

$$\{A(x) = 1\} \Leftrightarrow \{a \cdot x = 0\} \Leftrightarrow \{ta \cdot x = 0,\ \forall t \in \mathbb{F}[q]\} \Rightarrow \{\chi_{ta}(x) = 1,\ \forall t \in \mathbb{F}[q]\}.$$

Thus, $\widehat{A}_{ta} = \mathbb{E}_x[A(x)\chi_{ta}(x)] = \mathbb{E}_x[A(x)] = 1/q$, for all $t \in \mathbb{F}[q]$. By Parseval's identity these are the only non-zero Fourier coefficients. ◀

References

1 A. Agarwal, M. Charikar, K. Makarychev, and Y. Makarychev. O(sqrt(log n)) approximation algorithms for min UnCut, min 2CNF deletion, and directed cut problems. In *Proceedings of the ACM Symposium on the Theory of Computing*, pages 573–581, 2005.

2 N. Alon. Eigenvalues, geometric expanders, sorting in rounds, and Ramsey theory. *Combinatorica*, 6(3):207–219, 1986.

3 C. Ambühl, M. Mastrolilli, and O. Svensson. Inapproximability results for maximum edge biclique, minimum linear arrangement, and sparsest cut. *SIAM Journal of Computing*, 40(2):567–596, 2011.

4 B. Applebaum, B. Barak, and A. Wigderson. Public-key cryptography from different assumptions. In *Proceedings of the ACM Symposium on the Theory of Computing*, pages 171–180, 2010.

5 S. Arora, S. Rao, and U. V. Vazirani. Expander flows, geometric embeddings and graph partitioning. *Journal of the ACM*, 56(2):1–37, 2009.

6 P. Austrin and S. Khot. A simple deterministic reduction for the gap minimum distance of code problem. In *Proceedings of ICALP*, pages 474–485, 2011.

7 S. Chawla, R. Krauthgamer, R. Kumar, Y. Rabani, and D. Sivakumar. On the hardness of approximating multicut and sparsest-cut. *Computational Complexity*, 15(2):94–114, 2006.

8 Q. Cheng and D. Wan. A deterministic reduction for the gap minimum distance problem: [extended abstract]. In *Proceedings of the ACM Symposium on the Theory of Computing*, pages 33–38, 2009.

9 I. Dumer, D. Micciancio, and M. Sudan. Hardness of approximating the minimum distance of a linear code. *IEEE Trans. Information Theory*, 49(1):22–37, 2003.

10 U. Feige. Relations between average case complexity and approximation complexity. In *Proceedings of the ACM Symposium on the Theory of Computing*, pages 534–543, 2002.

11 U. Feige, M. Hajiaghayi, and J. R. Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM Journal of Computing*, 38(2):629–657, 2008.

12 V. Guruswami, C. Umans, and S. P. Vadhan. Unbalanced expanders and randomness extractors from Parvaresh–Vardy codes. *Journal of the ACM*, 56(4), 2009.

13 S. Khot. On the power of unique 2-prover 1-round games. In *Proceedings of the ACM Symposium on the Theory of Computing*, pages 767–775, 2002.

14 S. Khot. Ruling out PTAS for graph min-bisection, dense k-subgraph, and bipartite clique. *SIAM Journal of Computing*, 36(4):1025–1071, 2006.

15 S. Khot and N. K. Vishnoi. The unique games conjecture, integrality gap for cut problems and embeddability of negative type metrics into ℓ_1. In *Proceedings of the Annual Symposium on Foundations of Computer Science*, pages 53–62, 2005.

16 F. T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46(6):787–832, 1999.

17 N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, 1995.

18 R. J. Lipton and R. E. Tarjan. Applications of a planar separator theorem. *SIAM Journal of Computing*, 9(3):615–627, 1980.

19 A. Louis, P. Raghavendra, and S. Vempala. The complexity of approximating vertex expansion. In *Proceedings of the Annual Symposium on Foundations of Computer Science*, pages 360–369, 2013.

20 A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–227, 1988.

21 R. Müller and D. Wagner. α-vertex separator is NP-hard even for 3-regular graphs. *Computing*, 46:343–353, 1991.

22 N. Pippenger. Sorting and selecting in rounds. *SIAM Journal of Computing*, 16(6):1032–1038, 1987.

23 P. Raghavendra and D. Steurer. Graph expansion and the unique games conjecture. In *Proceedings of the ACM Symposium on the Theory of Computing*, pages 755–764, 2010.

24 P. Raghavendra, D. Steurer, and M. Tulsiani. Reductions between expansion problems. In *Proceedings of the Annual IEEE Conference on Computational Complexity*, pages 64–73, 2012.

25 A. Rao. *Randomness Extractors for Independent Sources and Applications.* PhD thesis, University of Texas at Austin, 2007.

26 M. Sipser and D. A. Spielman. Expander codes. *IEEE Transactions on Information Theory*, 42(6):1710–1722, 1996.

27 Z. Svitkina and L. Fleischer. Submodular approximation: Sampling-based algorithms and lower bounds. *SIAM Journal of Computing*, 40(6):1715–1737, 2011.

A Streaming Algorithm for the Undirected Longest Path Problem*

Lasse Kliemann[1], Christian Schielke[2], and Anand Srivastav[3]

1 Kiel University, Faculty of Engineering, Department of Computer Science, Kiel, Germany
lki@informatik.uni-kiel.de

2 Kiel University, Faculty of Engineering, Department of Computer Science, Kiel, Germany
csch@informatik.uni-kiel.de

3 Kiel University, Faculty of Engineering, Department of Computer Science, Kiel, Germany
asr@informatik.uni-kiel.de

Abstract

We present the first streaming algorithm for the longest path problem in undirected graphs. The input graph is given as a stream of edges and RAM is limited to only a linear number of edges at a time (linear in the number of vertices n). We prove a per-edge processing time of $\mathcal{O}(n)$, where a naive solution would have required $\Omega(n^2)$. Moreover, we give a concrete linear upper bound on the number of bits of RAM that are required.

On a set of graphs with various structure, we experimentally compare our algorithm with three leading RAM algorithms: Warnsdorf (1823), Pohl-Warnsdorf (1967), and Pongrácz (2012). Although conducting only a small constant number of passes over the input, our algorithm delivers competitive results: with the exception of preferential attachment graphs, we deliver at least 71% of the solution of the best RAM algorithm. The same minimum relative performance of 71% is observed over *all* graph classes after removing the 10% worst cases. This comparison has strong meaning, since for each instance class there is one algorithm that on average delivers at least 84% of a Hamilton path. In some cases we deliver even better results than any of the RAM algorithms.

1998 ACM Subject Classification G.2.2 [Graph Theory] Graph Algorithms

Keywords and phrases Streaming Algorithms, Undirected Longest Path Problem, Graph Algorithms, Combinatorial Optimization

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.56

1 Introduction

Let $G = (V, E)$ be an undirected, simple, and finite graph (all our graphs are of this type). A *path* of length k in G is a sequence of k distinct vertices $(v_1, \ldots, v_k)$ such that $v_i v_{i+1} \in E$ for each $1 \leq i < k$. (We write uv or vu for the undirected edge $\{u, v\}$ between vertices u and v.) In the *longest path problem* (LPP), we ask for a path in G that has maximum length among all the paths in G. This problem is NP-hard as seen easily by a reduction from the Hamilton cycle problem. A long line of research has investigated its approximability,

* Supported by German Research Foundation (DFG), Grant SR7/15-1 *Algorithmic Foundations for Genome Assembly.*

and several polynomial-time heuristics and algorithms with proven worst-case guarantee are known. With increasing size of the input graph, not only the time complexity of an algorithm becomes important but also its space complexity. For large graphs G, e. g., graphs that occur in genome assembly, it is not realistic anymore to assume that G can be fully stored in (fast) random-access memory (RAM). Instead we should assume that it can only be accessed efficiently in a *sequential* manner. The *graph streaming model* formalizes this. Here, the graph is given as a sequence $e_1, \dots, e_m$ of its edges, and access is only provided in the form of *passes*: a *pass* means each edge in the sequence is presented to the algorithm once. RAM is restricted to $\mathcal{O}(n \cdot \operatorname{poly}\log(n))$ bits, so essentially we can store a number of edges linear in the number n of vertices. Besides the solution quality, the most important property of a streaming algorithm is the number of passes that it has to conduct in order to obtain a good solution. To be practical, this number should be a small constant. This model does not only make sense when the input is stored on a disk or remote server, but also in the context of memory hierarchies, where cache RAM is several magnitudes faster than main RAM.

The standard graph search techniques breadth-first search (BFS) and depth-first search (DFS) cannot be done in the streaming model within a constant number of passes [13]. However, all relevant existing algorithms for the LPP use some form of graph search and/or require super-linear data structures (as in the dynamic programming part for color coding [1]). We therefore take a different approach that will never have to do a graph search on a graph with more than a linear number of edges and moreover does not require more than a linear amount of RAM for additional data structures.

1.1 Our Contribution

We give a streaming algorithm for the longest path problem in undirected graphs with a proven per-edge processing time of $\mathcal{O}(n)$. Our algorithm works in two phases, which we outline here briefly and explain in detail in Section 3. In the first phase, global information on the graph is gathered in form of a constant number of spanning trees[1] $T_1, \dots, T_\tau$. This is possible in the streaming model since roughly speaking, for a spanning tree we can 'take edges as they come'. A spanning tree can be constructed in just one pass – we however use multiple passes and limit the maximum degree during the first passes in order to favor path-like structures and avoid clusters of edges. Experiments clearly indicate that this degree-limiting is essential for solution quality. The spanning trees fit into RAM, since we consider τ as constant (we will in fact have $\tau = 1$ or $\tau = 2$ in the experiments). After construction of the τ trees, they are merged into one graph U by taking the union of their edges. Then we use standard algorithms to determine a long path P in U, isolate P, and finally add enough edges around P to obtain a tree T.

Then, in the second phase, we conduct further passes during which we test if the exchange of single edges of T can improve the longest path in it. (A longest path in a tree can be found by conducting DFS two times [10]; the length of a longest path in a tree is its diameter.) The main challenge in the second phase is to quickly determine which edges should be exchanged. We show that this decision can be made in linear time, hence yielding a per-edge processing time of $\mathcal{O}(n)$.

An experimental study is conducted on randomly generated instances with different structure, including ones created with the recently published generator for hyperbolic geometric

[1] For simplicity, we always assume that our input graphs are connected, but it would be easy to adapt our algorithm to the general case.

random graphs [30]. Different variants of our streaming algorithm are compared with four RAM algorithms: Warnsdorf and Pohl-Warnsdorf (two related classical heuristics [24, 25]), Pongrácz (a recently published heuristic [26]), and a simple randomized DFS. Experiments show that although we never do more than 11 passes, results delivered by our algorithm are competitive. We deliver at least 71% of the best result delivered by any of the tested RAM algorithms, with the exception of preferential attachment graphs. By considering low percentiles, we observe a similar quality without any restriction on the graph class. This is a good result also in absolute terms, since we observe that for each graph class and set of parameters, there is one algorithm that on average gives a path of length $0.84 \cdot n$, i. e., 84% of a Hamilton path. On some graph classes, we outperform any of the tested RAM algorithms, which makes our algorithm interesting even outside of the streaming setting. A detailed discussion of results is given in Section 7.

1.2 Previous and Related Work

Algorithms for the LPP have been studied extensively in the RAM model. We start listing algorithms with proven guarantees. Bodlaender [9] and Monien [22] gave algorithms that find a path of length k (if it exists) in $\mathcal{O}(2^k k!\, n)$ and $\mathcal{O}(k!\, nm)$ time, respectively. Alon et al. [1] introduced the method of *color coding* and based on that gave an algorithm running in expected time $2^{\mathcal{O}(k)} n$. There is a recent randomized algorithm by Björklund et al. [7] that given k, finds a path of length k (if it exists) in $\mathcal{O}(1.66^k \cdot \operatorname{poly} n)$ time (see also [19, 32, 5]). Those works show that the problem is fixed-parameter tractable: a path of length k can be found (if it exists) in polynomial time, for fixed k. The particular dependence of the running time on k (factorial or exponential) determines up to which k we stay polynomial and thus determines the length guarantee for a polynomial-time approximation algorithm.

In Hamiltonian graphs, a path of length $\Omega\Big(\big(\frac{\log(n)}{\log\log(n)}\big)^2\Big)$ can be found with the algorithm by Vishwanathan [29]; and Feder et al. gave further results for sparse Hamiltonian graphs [11]. Björklund and Husfeldt [6] gave an algorithm that finds a path of length $\Omega\big(\frac{(\log(\mathrm{opt}))^2}{\log\log(\mathrm{opt})}\big)$, where opt is the length of a longest path. It works by a decomposition of the graph into paths and cycles. Their technique subsequently was extended by Gabow [14] and Gabow and Nie [15] yielding guarantees for the length of the path of $\exp\big(\Omega\big(\sqrt{\frac{\log(\mathrm{opt})}{\log\log(\mathrm{opt})}}\big)\big)$ and $\exp(\Omega(\sqrt{\log(\mathrm{opt})}))$, respectively. Apart from that, the field is dominated by heuristics, such as (Pohl-)Warnsdorf [24, 25] and Pongrácz [26].

The Björklund-Husfeldt algorithm uses color coding as an important subroutine. We implemented and tested a simple algorithm based on color coding, which gave inferior results and more importantly took very long time to complete, substantially longer than (Pohl-)Warnsdorf, Pongrácz, or our algorithm. Details will be given in the full version. Thus we refrained from further implementing the Björklund-Husfeldt algorithm. (The original description in [6] uses Bodlaender's algorithm [9], which has an even higher running time than color coding.) The Gabow-Nie algorithm [15] does not use color coding, but at the time of writing was only available as a short conference version, making it difficult to implement.

Several non-approximability results have been shown by Karger et al. [17]: a constant-factor approximation is NP-hard; and for any $\varepsilon > 0$, the LPP cannot be approximated with a ratio of $2^{\mathcal{O}(\log^{1-\varepsilon}(n))}$, unless NP $\subseteq$ DTIME$(2^{\mathcal{O}(\log^{1/\varepsilon}(n))})$, that is, such an approximation is quasi-NP-hard. Bazgan et al. showed that the same holds even when restricting to cubic Hamiltonian graphs [4].

The LPP is also interesting in directed graphs. For any $\varepsilon > 0$, it is NP-hard to approximate in directed graphs within $n^{1-\varepsilon}$ [8]. The best approximation guarantee in the directed case (unless restricting to special classes of graphs) is still the color coding algorithm that also works in the undirected case [1]. For special graph classes, there exist exact polynomial-time algorithms, e. g., for (undirected) trees (given by Dijkstra around 1960, see [10] for a proof), for directed acyclic graphs [27, pp. 661-666], for grid graphs [18], and for cactus graphs [28, 21].

The study of graph problems in streaming models started around the beginning of the 21st century, see [2, 12] for early works. The idea of using a linear amount of memory is due to Muthukrishnan [23]. Up to then, the ‘streaming’ term was associated with sub-linear memory, which is not enough for many graph problems [12]. To emphasize the difference, the streaming model with linear RAM (that we use) is also referred to as the *semi-streaming model* in the literature.

Since then, many kinds of graph problems have been addressed, such as shortest paths, spanning trees, connectivity, cuts, matching, and vertex cover. Several lower bounds are known. Most importantly for us, Feigenbaum et al. [13] proved that any BFS algorithm computing the first k layers with probability at least $2/3$, requires more than $k/2$ passes if staying within $\mathcal{O}(n \cdot \operatorname{poly}\log(n))$ memory (see Guruswami and Onak [16] for improved lower bounds). This constitutes a substantial hurdle when transferring existing algorithms into the streaming model. To the best of our knowledge, longest paths have not been addressed before in a streaming model.

It must be emphasized that streaming techniques also make sense when the graph is of size $c \cdot n \cdot \log(n)$ if a streaming algorithm can guarantee to stay within $c' \cdot n \cdot \log(n)$ for $c' < c$. Therefore, we give a memory guarantee for our algorithm using concrete constants.

1.3 Ongoing and Future Work

A major practical motivation for this work is the genome assembly problem. There, a graph is built based on the output of a sequencing machine, and long paths in this graph correspond to large sub-sequences of the genome. However, there the graphs are usually directed. Therefore, our next step will be the extension of our algorithm to directed graphs and the integration of it into existing genome assembly software.

In a separate line of research, we will try to give a streaming algorithm for the LPP (undirected or directed) with a proven worst-case guarantee on the length of the path and number of passes. At this time, it is unclear if any of the established theoretical techniques, such as color coding and Björklund-Husfeldt-type decompositions, are feasible in the streaming model.

Outline. We briefly describe three known RAM algorithms in Section 2. In Section 3, all the details of our algorithm are explained. In Section 4 we analyze its theoretical properties. The experimental studies takes place in Section 5 to Section 8.

2 Previous Algorithms

Trees. An algorithm for longest paths in trees was presented by Dijkstra around 1960; a proof of correctness can be found in [10]. It consists of two invocations of DFS, the first starting at an arbitrarily chosen vertex (e. g., chosen uniformly at random), and the second starting at a vertex that is in the final layer constructed by the first DFS.

Warnsdorf and Pohl-Warnsdorf. Warnsdorf’s rule was originally presented in 1823 and is a DFS that always picks a neighbor with a minimum number of unvisited neighbors. In case there are multiple such neighbors to choose from, Pohl gave a refinement [24, 25]: we restrict to those neighbors which themselves have a minimum-degree neighbor. Each vertex is used once as the starting point of the DFS, and the best path found is returned. This gives a total runtime of $\mathcal{O}(nm)$.

Pongrácz. This algorithm was announced in 2012 [26] and to the best of our knowledge has not been thoroughly studied since. We give a technically slightly modified description here. Given a start vertex r, using BFS we compute for each vertex v its distance to r. Then, starting at a randomly chosen v, we conduct a DFS that always picks an unvisited neighbor with maximum distance to r. Each vertex is used once as the start r, and the longest path found is returned. In the original version, for each r, also *each* v is tried (and not just one chosen randomly). In order to stay within $\mathcal{O}(nm)$, we decided to enumerate only one of the two possibilities: either r or v. In preliminary experiments, we found the choice given here (enumerate all r, pick one v randomly) to be superior. We leave a thorough study of the different variants of Pongrácz's algorithm for future work.

3 Description of Our Streaming Algorithm

Our algorithm works in two phases: (1) spanning tree construction, (2) spanning tree diameter improvement. Phase (1) is characterized by a parameter $\tau \in \mathbb{N}$ and a sequence $D = (D_1, \ldots, D_{q_1})$ of degree limits, where $q_1 \geq 2$ and $D_{q_1} = \infty$. For each $i \in [\tau] = \{1, \ldots, \tau\}$, a tree T_i is constructed. We start with the empty graph $T_i = (V, \emptyset)$ and then add edges to T_i over a number of q_1 passes. In each pass $p \in [q_1]$, we add an edge to T_i iff that does not create any cycle and it does not increase the maximum degree in T_i beyond D_p. Since $D_{q_1} = \infty$, we arrive at a spanning tree eventually (recall that we assume all our input graphs to be connected). The motivation for the degree limit is to favor path-like structures over clusters of edges. As an extreme example, consider a complete graph. Without degree restriction, it is possible that a spanning tree is constructed that is a star; whereas with a degree restriction of 2, we find a Hamilton path during the first pass.

In order to not just create the same tree τ times, in the first pass, we pick a number $r \in [m]$ uniformly at random (where $e_1, \ldots, e_m$ is the stream of edges) and ignore any edges with an index smaller than r. Due to this offset for the first pass, it makes sense (but is not necessary) to use the same degree limit for the second pass. We will test $D = (2, \infty)$ and $D = (2, 2, 3, \infty)$ in experiments. By standard techniques (keeping track of the connected components), this algorithm can be implemented with a per-edge processing time of $\mathcal{O}(n)$: we can decide in $\mathcal{O}(1)$ if the current edge is to be inserted and if so, it takes $\mathcal{O}(n)$ to update connectivity information.

When all trees $T_1, \ldots, T_\tau$ have been constructed, we unite them into a graph $U := (V, \bigcup_{i=1}^{\tau} E(T_i))$. This graph will in general contain cycles, but it has no more than τn edges. Since we construct U from trees, it is guaranteed to be connected and to span all the vertices of the input graph. In U, a long path P is constructed with a RAM algorithm; we use the Warnsdorf algorithm for this task. The final step of the first phase is to isolate P and then to build a spanning tree T around it using the same technique as for the trees $T_1, \ldots, T_\tau$. Since we may assume that the constructions of $T_1, \ldots, T_\tau$ are fed from the same passes, we thus have $2q_1$ passes for the first phase. We summarize phase (1) in Algorithm 1, which uses procedure `SpanningTree`, also given below. For a set X, we write $x :=_{\mathrm{unif}} X$ to express that x is drawn uniformly at random from X.

When phase (1) is concluded, we determine a longest path P in the spanning tree T using the Dijkstra algorithm (Section 2). In phase (2), we try to modify this tree in order

```
Algorithm 1: Streaming Phase (1): Spanning Tree Construction
  Input: connected graph $G = (V, E)$ as a stream of edges, parameter $\tau$,
         degree limit sequence $D = (D_1, \dots, D_{q_1})$
  Output: spanning tree of $G$
1 foreach $i = 1, \dots, \tau$ do
2     $T_i \coloneqq (V, \emptyset)$;
3     SpanningTree ($T_i$);
4 $U \coloneqq (V, \bigcup_{i=1}^{\tau} E(T_i))$;
5 find a long path $P$ in $U$ using Warnsdorf's algorithm;
6 $T \coloneqq (V, E(P))$;
7 SpanningTree ($T$);
8 return $T$;
```

```
Procedure SpanningTree(T)
  Input: forest $T$ on $V$, possibly empty
  Output: spanning tree on $V$
1 $r \coloneqq_{\text{unif}} [m]$;
2 fast-forward the stream to position $r$;
3 for $p = 1, \dots, q_1$ do
4     while not at the end of the stream do
5         get next edge $vw$ from the stream;
6         if $T + vw$ is cycle-free and $\max\{\deg_T(v), \deg_T(w)\} < D_p$ then $T \coloneqq T + vw$;
7         if $|T| = n - 1$ then break;
8     rewind the stream to its beginning;
```

that it admits longer paths than P. A number of additional passes is conducted. In order to save time, we developed a criterion based on which we only consider a fraction of the edges during those passes. We explored the two options: (i) consider each edge independently with probability $\frac{n}{m+1}$ (resulting in only $\mathcal{O}(n)$ edges being considered); or (ii) skip an edge if both endpoints are on the so-far longest path P. After preliminary experiments, we decided for option (ii) due to better solution quality at a moderate runtime expense. A detailed comparison of (i) and (ii) is planned for the full version of this work; in our tables in Section 8, however we already give results for one variant of our algorithm using option (i).

For each edge e that is considered and that is not in T, we temporarily add e to T, creating a fundamental cycle C in $T' \coloneqq T + e$. We want to go back to a tree. To this end, we have to remove an edge from C. This edge is chosen so that among all possibilities, the resulting tree has maximum diameter.

It should not be assumed that an edge with both endpoints on P could not yield an improvement. Intuitively, relative to P it acts like a shortcut, but examples can be found where adding such an edge (and subsequently removing one edge from the fundamental cycle) improves the diameter of the tree. Still, criterion (ii) has shown to be effective in practice.

Phase (2) terminates after a preset number of passes q_2. We summarize phase (2) in Algorithm 2, where for any graph H, we denote $\ell(H)$ the length of a longest path in H.

Algorithm 2: Streaming Phase (2): Improvement

```
   Input: connected graph $G$ as a stream of edges, spanning tree $T$, pass limit $q_2$
   Output: a (long) path in $G$
1  compute longest path $P$ in $T$ with Dijkstra algorithm;
2  for $q_2$ times do
3      rewind the stream to its beginning;
4      while not at the end of the stream do
5          get next edge $e = vw$ from stream;
6          if $v \in V(P)$ and $w \in V(P)$ then discard and continue with next iteration;
7          $T' := T + e$;
8          compute fundamental cycle $C$ in $T'$;
9          $\ell^* := \max_{f \in E(C) \setminus \{e\}} \ell(T' - f)$;
10         if $\ell^* > |P|$ then
11             pick any $e'$ from the set $\{f \in E(C) \setminus \{e\} ;\ \ell(T' - f) = \ell^*\}$;
12             $T := T' - e'$;
13             update $P$ with longest path in $T$;
14 return $P$;
```

4 Properties of Our Streaming Algorithm

If the cycle C is of length $\Omega(n)$, then a naive implementation requires $\Omega(n^2)$ to find an edge e' to remove (temporarily remove each edge on the cycle and invoke the Dijkstra algorithm). However, we have:

▶ **Theorem 1.** *Phase (2) can be implemented with per-edge processing time $\mathcal{O}(n)$.*

Proof. An $\mathcal{O}(n)$ bound is clear for all lines of Algorithm 2, except line 9 and line 11. Denote

$$\ell' := \max_{f \in E(C) \setminus \{e\}} \max \{|P| \,;\ P \text{ is path in } T' - f \text{ and } e \in E(P)\}$$

and let $R' \subseteq E(C) \setminus \{e\}$ be the set of edges where this maximum is attained. Then the following implications hold: $\ell' \leq |P| \implies \ell^* \leq |P|$ and $\ell' > |P| \implies \ell' = \ell^*$. This is because if a longest path in $T' - f$ is supposed to be longer than P, it must use e (since otherwise it would be a path in T). Hence it suffices to determine ℓ', and if $\ell' > |P|$, to find an element of R'.

Denote $C = (v_1, \dots, v_k)$ the fundamental cycle for some $k \in \mathbb{N}$ written so that $e = v_1 v_k$. When computing ℓ', we can restrict to paths in T' of the form

$$(\dots, v_s, v_{s-1}, \dots, v_1, v_k, v_{k-1}, \dots, v_t, \dots) \tag{1}$$

for $1 \leq s < t \leq k$, where v_s is the first and v_t is the last common vertex, respectively, of the path and C. For each i, let T_i be the connected component of v_i in $T - E(C)$, i. e., T_i is the part of T that is reachable from v_i without using the edges of C. Denote $\ell(T_i)$ the length of a longest path in T_i that starts at v_i and denote $c_i := \ell(T_i) + i - 1$ and $a_i := \ell(T_i) + k - i$. Then a longest path entering C at v_s and leaving it at v_t, as in (1), has length exactly $c_s + a_t$. Hence we have to determine a pair (s, t) such that $c_s + a_t$ is maximum (this maximum value is ℓ'); we call such a pair an *optimal pair*. If the so determined value ℓ' is not greater than $|P|$, then nothing further has to be done (the edge e cannot give an improvement). Otherwise, having constructed our optimal pair (s, t), we pick an arbitrary edge (e. g., uniformly at random) from $\{v_i v_{i+1} \; ; \; s \leq i < t\}$, which are the edges between v_s and v_t on C. We show that the following algorithm computes the value ℓ' and an optimal pair in $\mathcal{O}(n)$.

```
compute $c_1, \ldots, c_{k-1}$ and $a_2, \ldots, a_k$ using DFS;
$M := 0$; $L := 0$;
for $i = 1, \ldots, k-1$ do
    if $c_i > M$ then
        $M := c_i$;
        $s := i$;
    if $M + a_{i+1} > L$ then
        $L := M + a_{i+1}$;
        $t := i + 1$;
return $(s, t)$;
```

The total of computations in line 1 can be done by DFS in $\mathcal{O}(n)$, and the loop in $\mathcal{O}(k) \leq \mathcal{O}(n)$. We prove that the final (s, t) is optimal. For fixed t, the best possible length $c_s + c_t$ is obtained if t is combined with an $s < t$ where $c_s \geq c_j$ for all $j < t$. In the algorithm, for each t (when $t = i + 1$ in the loop) we combine a_t with the maximum $\max_{j<t} c_j$ (stored in the variable M). Thus, when the algorithm terminates, $L = \ell'$ and $c_s + c_t = \ell'$. ◀

▶ **Corollary 2.** *Our streaming algorithm (with the two phases as in Algorithm 1 and Algorithm 2) can be implemented with a per-edge processing time of $\mathcal{O}(n)$.*

We turn to the memory requirement. Denote b the amount of RAM required to store one vertex or one pointer (e. g., $b = 32$ bit or $b = 64$ bit) and call $n \cdot b$ one *unit*.

▶ **Theorem 3.** *Our streaming algorithm (with the two phases as in Algorithm 1 and Algorithm 2) conducts at most $2q_1 + q_2$ passes. Moreover, the algorithm can be implemented such that the RAM requirement is at most $(\max\{4\tau, 2\tau + 4\} \cdot n + c) \cdot b$ with a constant c.*

Proof. The construction of each of the initial trees $T_1, \ldots, T_\tau$ can be fed from the same passes, so we obtain those τ trees within at most q_1 passes. After isolating the path P, we need at most q_1 more passes to get back to a spanning tree. A bound of q_2 for phase (2) is obvious. We turn to the memory requirement.

Phase (1). All the adjacency lists of one tree together require 2 units, plus 1 unit for an array of pointers to each of the lists. We need 1 additional unit per tree to store connectivity information during tree construction. This amounts to 4τ units for the main data structures at any time so far, plus a few extra bits required for bookkeeping (loop variables, etc) that are covered by the constant c. The graph U can be stored in $2\tau + 1$ units (adjacency lists plus pointer array). For the Warnsdorf algorithm, we need 1 unit to store DFS information (e. g., store the DFS tree as a predecessor relation) and 1 unit for the best path so far. To determine the next vertex to visit (according to Warnsdorf's rule), we need $\Delta b \leq nb$ bits, where Δ is the maximum degree in the graph. This amounts to $2\tau + 4$ units for the main data structures for the Warnsdorf algorithm on U. The rest of phase (1) is clearly covered by this as well.

Phase (2). We need 3 units to store the tree, 1 unit to store the longest path so far, 1 unit for the fundamental cycle, and 1 unit for DFS. This amounts to 6 units during this phase plus bookkeeping, which is covered by the stated bound. ◀

▶ Remark. On a graph with average degree d, the Warnsdorf, Pohl-Warnsdorf, and Pongrácz algorithms each requires at least $(d+3)\cdot nb$ bits of memory.

Proof. Since those algorithms perform special variants of DFS (and Pongrácz also BFS), we cannot restrict them to sequential access and thus we have to load the instance into RAM as adjacency lists.[2] Hence, $d+1$ units are required to store the graph. Two more units must be allotted to store DFS information and the longest path found so far, in the case of Pongrácz need one more unit for the distance information. ◀

▶ **Corollary 4.** *Not counting the additive constant c from Theorem 3, the RAM algorithms require $\frac{d+3}{\max\{4\tau,2\tau+4\}}$ times more RAM than our streaming algorithm, on a graph with average degree d. For $\tau=2$, this ratio is $\frac{d+3}{8}$.*

5 Test Instances

Connected Random. We denote this model by $\mathcal{G}^*(n,p)$. A graph is constructed by starting with a random tree on n vertices (via a randomly chosen Prüfer sequence) and then adding further edges as in $\mathcal{G}(n,p)$. The average degree in such a graph is slightly larger than np due to the $n-1$ initial tree edges.

Chains. Parameters for a chain graph are n,p, and k, with n being a multiple of k. We create k graphs $G_1,\dots,G_k$, the *clusters*, according to $\mathcal{G}^*(n,p)$, each on n/k vertices. Then we insert an edge v_iw_i with randomly chosen $v_i\in V(G_i)$ and $w_i\in V(G_{i+1})$ for each $1\le i<k$, making sure that $w_i\ne v_{i+1}$. Such graphs pose a particular challenge to DFS-based LPP algorithms, since if the DFS visits the connecting point to the next cluster (w_i or v_i) too early, it will eventually miss out on a large number of vertices in the current cluster.

Preferential Attachment and Small World. Preferential attachment graphs are created as per the Barabási-Albert model [3]: parameters are $n,n_0,d\in\mathbb{N}$, where n is total the number of vertices, n_0 is the size of the initial tree, and in each step the new vertex is connected by d new edges. This model guarantees connectedness. Small world graphs are created as per the Watts-Strogatz model [31], with a small modification. Parameters are $n,d\in\mathbb{N}$, with d even, and $0\le\beta\le1$. We start with a ring lattice where each vertex is connected to each $d/2$ vertices on either side, then each edge vw with v and w not being next to each other on the ring is replaced with a random edge vu with probability β (the *rewiring probability*). Our modification (not to rewire certain edges) guarantees that the result is Hamiltonian (and in particular connected).

These two models were chosen since they yield very different degree distributions: for preferential attachment, we have a power-law and there exist a few *hubs*, i. e., vertices with high degree. In the small world model on the other hand, vertices tend to have similar degree.

[2] This is unless we invoke external-memory techniques, which is unexplored for the LPP at this time.

Hyperbolic Geometric. Hyperbolic geometric graphs are a very interesting new class of graphs, for which efficient generators were recently given by von Looz, Staudt, Meyerhenke, and Prutkin [30]. They are constructed in hyperbolic space of constant negative curvature. Vertices correspond to points that are randomly inserted into this space, and an edge between two vertices is inserted if the corresponding points are within a certain distance from each other. This model has been shown to exhibit many features of complex real-world networks. We refer to [20, 30] for details. Parameters are number of vertices n, average degree d, and the exponent γ of the power-law degree distribution. We use the generator implementation from [30]. Connectedness is ensured by initializing the graph with a random tree.

6 Experimental Setup

Each algorithm was implemented in C++14. Each graph stream is realized as a `std::vector` of pairs of 32 bit integers. We keep those vectors in RAM for the sake of faster running times and hence more experiments conducted – but it is guaranteed that we access those vectors only sequentially and all other data structures are $\mathcal{O}(n)$. Our implementation also allows to process graphs stored in a file on disk, without copying the contents of the file into RAM (it is accessed via a `std::ifstream`). Using the *Valgrind* tool *Massif*,[3] we verified that RAM consumption of our algorithm is indeed independent of the number of edges. For each instance, the stream of edges is randomized once and the order does not change between passes or between the invocations of the algorithms. Each implementation concludes immediately when a Hamilton path is found.

For each random graph model under consideration, we test three settings: $n = 16{,}000$ and nominal average degree $d = 14$ (*sparse*); $n = 16{,}000$ and nominal average degree $d = \sqrt[3]{n}$ (*dense*); and $n = 100{,}000$ and nominal average degree $d = 10$ (*large*). (Note that chain and hyperbolic graphs will have a slightly larger average degree than the given d due to the additional tree that is used to guarantee connectedness.) The *dense* graphs have $\Omega(n^{4/3})$ edges and are thus beyond the theoretical RAM capacity of the semi-streaming model. More on the practical side, note that by Corollary 4 (not counting the small additive constant), even for average degree $d = 14$, the RAM algorithms require more than two times more memory than ours when configured with $\tau \leq 2$. Due to lack of space we skip the details for *sparse* and *dense* small world graphs, and we only use a selection of algorithms for the *large* graphs. The study of larger and more instances is deferred to the full version, due to time constraints.

We run Warnsdorf, Pohl-Warnsdorf, Pongrácz, the simple randomized DFS, and different variants of our algorithm on 100 randomly generated instances for each parameter set (only 50 instances for *large* graphs in order to save time) and record the length of the path that is found and the running time. Variants of our algorithm are denoted in the form $\tau/q_1/q_2$, where τ is the number of trees in the beginning, q_1 is the maximum number of passes used to construct a spanning tree using degree limiting, and q_2 is the number of improvement passes. In order to save time, for fixed τ and q_1, we obtain results for $\tau/q_1/0$ up to $\tau/q_1/q_2$ by running $\tau/q_1/q_2$ and recording intermediate results.

Solution quality is analyzed in terms of *relative solution quality.* For an instance I and algorithm A, denote $\ell(A, I)$ the length of the path delivered by A on I. Then we define $\rho(A, I) := \frac{\ell(A,I)}{\max_{A'} \ell(A',I)} \in [0, 1]$, where A' runs over all algorithms under investigation. That

[3] `http://valgrind.org/docs/manual/ms-manual.html`

is the result of A divided by the best result on any of the algorithms. Clearly, one algorithm per instance will always have relative solution quality 100%.

7 Data and Discussion

Tables with detailed experimental data can be found in section 8. The column labeled 'ℓ' gives statistics (mean value μ and standard deviation σ) for the lengths of the paths found and is intended as a general orientation in which range our solutions are located. The column labeled 'wins' counts how many times this algorithm delivered the best solution, i. e., how many times it achieved relative solution quality $\rho = 100\%$. Detailed statistics are given for the relative performance in the following columns: mean value, standard deviation, minimum, 5th and 10th percentile, and median. We use percentile notation everywhere: P_0 for the minimum, P_5 and P_{10} for the 5th and 10th percentile, and P_{50} for the median. In the final two columns, we give the running time in seconds. The algorithm marked with a star (2/4/3*) uses the randomized criterion for skipping edges in the improvement phase, whereas all other variants of our algorithm use the path criterion as stated in Algorithm 2. In the following, we distill the data from the tables into several observations and conclusions.

- The fact that the simple randomized DFS algorithm (denoted 'DFS' in the tables) delivers clearly inferior results in many cases is an indication that at least those instances are not 'too easy'.
- Warnsdorf and Pohl-Warnsdorf are generally the best, except for chain graphs. For many of the instances, they find a Hamilton path, and then they are very fast, sometimes below one second. Note that this advantage could easily be removed by making the graphs non-Hamiltonian, e. g., by connecting two additional vertices as leafs to the same vertex.
- Warnsdorf and Pohl-Warnsdorf are close to each other in terms of solution quality, but unsurprisingly the former is faster.
- In terms of the average path length $\mu(\ell)$, for each set of parameters there is one algorithm that delivers at least $0.84 \cdot n$, i. e., 84% of a Hamilton path. It follows that a good relative performance also means a good absolute performance.
- Our strongest variant, 2/4/3, with the exception of preferential attachment graphs, always delivers a relative solution quality of at least 71%. For preferential attachment, we record a minimum of 49% in Table 3. In terms of the 5th percentile, i. e., after removing the 5% worst cases, and omitting preferential attachment graphs, our minimum relative solution quality is 83%. In terms of the 10th percentile and including preferential attachment graphs, we still have at least 71%. In terms of mean and median, we have at least 83%.
- Regarding running time, we compare our variant 2/4/3 with Warnsdorf, which is the fastest RAM algorithm, not counting the simple randomized DFS. Clearly, we cannot compete in cases where Warnsdorf finds a Hamilton path within a second, but as remarked before, this advantage of Warnsdorf could easily be removed by making the graph non-Hamiltonian. Apart from those cases, in the *sparse* and *dense* sets, the biggest difference is for *sparse* hyperbolic graphs, where Warnsdorf only needs about 56% of our running time on average. For *dense* chains, we are faster than Warnsdorf. For the *large* set, our variant 2/4/1 has similar running times as Warnsdorf, while delivering at least 71% in terms of P_{10}, and when excluding preferential attachment graphs it delivers 82% in terms of P_5. More than one improvement pass here only gives incremental gain, so in order to save time on large graphs, the variant 2/4/1 is recommended over 2/4/2 or 2/4/3.
- Using $\tau = 2$ has a clear advantage over $\tau = 1$, in particular compare 1/2/0 with 2/2/0 in terms of ℓ in Table 1 and Table 2.

- The degree-limiting technique yields substantial improvements. For $q_1 = 2$ (i.e., for variants of the form $\tau/2/q_2$), we use the sequence $D = (2, \infty)$, i.e., in the first pass we limit the degree to 2 and in the second pass we have no limit. In the configuration with $q_1 = 4$ we use $D = (2, 2, 3, \infty)$. Comparing for example 1/2/0 with 1/4/0 with respect to ℓ in Table 2 for preferential attachment and hyperbolic graphs, we see that 1/2/0 delivers roughly $50 - 60\%$ length on average compared to 1/4/0. Comparing 2/2/3 with 2/4/3 in particular with respect to P_0, P_5, and P_{10} for preferential attachment graphs in Table 1, we see that $q_1 = 4$ brings an improvement even on top of the improvement gained by using $\tau = 2$ and by the improvement phase.
- The improvement phase (phase (2)) can bring further improvements, in particular with respect to P_0. This is seen for example by comparing 2/4/0 with 2/4/3 for preferential attachment and hyperbolic graphs in Table 1.
- Comparing the runtimes of 2/2/3 and 2/4/3 over all tables, we find that consistently the former is slower, while delivering inferior solutions. The same goes for 1/4/3 and 2/4/3; here the difference in running time is very high for preferential attachment graphs. This shows that a lack of effort in phase (1) can make phase (2) substantially slower. An explanation is that more improvement steps have to be carried out.
- Comparing 2/4/3 with 2/4/3*, we find the former being consistently better in terms of solution quality, but requiring up to roughly 30% more time.
- Our biggest advantage (using 2/4/3) over the other algorithms is for chain graphs.

In particular, we conclude from those observations that none of the three features (namely using multiple trees in the beginning, degree-limiting, and improvement) should be missed. The combination of all those features makes our algorithm competitive.

8 Tables of Experimental Data

On the following pages please find tables of results for the experiments as discussed in Section 7. By μ we denote the mean value and by σ the standard deviation. By P_i we denote the ith percentile, in particular P_0 is the minimum and P_{50} is the median. By ℓ we denote the path length and by ρ the relative performance. Running times t (last two columns) are in seconds. For further explanations, please see Section 7.

Acknowledgments. We thank the German Research Foundation (DFG) for financial support through Grant SR7/15-1 *Algorithmic Foundations for Genome Assembly* within Priority Programme 1736 *Algorithms for Big Data*. We thank Ole Kliemann for substantially improving our C++ implementation and helpful discussions.

Table 1 *Sparse Set:* $n = 16{,}000$ and $d = 14$.

		ℓ			ρ in %						t	
graph class	algo	μ	σ	wins	μ	σ	P_0	P_5	P_{10}	P_{50}	μ	σ
chain $k = 125$ $l = 128$ $p = 0.11$ $n = 16{,}000$ $\lvert E\rvert \approx 127{,}044$	1/2/0	3,032	645	0	20	4	11	12	13	21	26	2
	1/2/3	13,435	606	0	89	4	79	80	82	91	185	34
	1/4/0	3,947	222	0	26	1	22	24	24	26	23	0
	1/4/3	14,150	55	0	94	1	93	93	93	94	141	4
	2/2/0	10,985	1,336	0	73	9	17	58	64	75	46	4
	2/2/3	14,522	373	7	96	2	91	92	93	97	129	18
	2/4/0	11,683	617	0	77	4	63	68	73	78	48	4
	2/4/3	15,062	122	93	100	0	98	100	100	100	103	3
	2/4/3*	14,346	283	0	95	2	86	90	94	96	92	4
	Pon	14,518	52	0	96	1	95	95	95	96	110	4
	War	10,598	304	0	70	2	66	67	68	70	86	2
	PW	10,539	306	0	70	2	65	67	68	70	131	3
	DFS	9,255	165	0	61	1	59	60	60	61	38	1
pref. attach. $n_0 = 7$ $d = 14$ $n = 16{,}000$ $\lvert E\rvert = 111{,}957$	1/2/0	464	137	0	3	1	1	2	2	3	35	6
	1/2/3	6,653	1,212	0	42	8	27	27	28	43	437	23
	1/4/0	748	105	0	5	1	3	4	4	5	29	0
	1/4/3	8,281	122	0	52	1	50	50	51	52	394	8
	2/2/0	12,712	2,350	0	79	15	15	43	58	85	47	3
	2/2/3	13,000	1,859	0	81	12	36	53	65	86	169	71
	2/4/0	13,743	1,825	0	86	11	18	66	76	90	46	4
	2/4/3	14,060	1,100	0	88	7	55	73	79	91	133	44
	2/4/3*	13,817	1,265	0	86	8	51	69	75	90	104	3
	Pon	13,565	28	0	85	0	84	84	85	85	113	4
	War	16,000	0	100	100	0	100	100	100	100	0	0
	PW	16,000	0	100	100	0	100	100	100	100	0	0
	DFS	12,385	27	0	77	0	77	77	77	77	41	1
hyperbolic $d = 14$ $\gamma = 3$ $n = 16{,}000$ $\lvert E\rvert \approx 128{,}185$	1/2/0	586	185	0	4	1	1	2	2	4	33	22
	1/2/3	9,791	826	0	61	5	49	50	54	62	337	38
	1/4/0	968	144	0	6	1	4	5	5	6	25	0
	1/4/3	10,987	145	0	69	1	67	67	67	69	290	8
	2/2/0	12,822	1,808	0	80	11	37	47	62	84	46	4
	2/2/3	14,099	1,013	0	88	6	67	71	77	90	130	41
	2/4/0	13,732	941	0	86	6	56	72	78	88	46	5
	2/4/3	14,646	509	0	92	3	77	84	86	93	108	19
	2/4/3*	14,303	587	0	89	4	69	82	85	91	97	3
	Pon	14,373	106	0	90	1	87	89	89	90	117	5
	War	15,997	5	92	100	0	100	100	100	100	61	38
	PW	15,998	4	94	100	0	100	100	100	100	83	52
	DFS	12,908	22	0	81	0	80	80	81	81	40	1

Table 2 *Dense Set:* $n = 16{,}000$ and $d = \sqrt[3]{n}$.

		ℓ			ρ in %						t	
graph class	algo	μ	σ	wins	μ	σ	P_0	P_5	P_{10}	P_{50}	μ	σ
	1/2/0	3,749	860	0	24	6	11	13	15	26	25	2
	1/2/3	14,633	438	0	94	3	86	87	89	95	200	52
	1/4/0	4,666	314	0	30	2	24	27	27	30	23	1
	1/4/3	15,079	34	0	97	0	97	97	97	97	144	4
chain	2/2/0	11,646	755	0	75	5	57	62	70	77	49	5
$k = 125$	2/2/3	15,248	251	12	98	2	92	95	96	99	139	26
$l = 128$	2/4/0	11,864	867	0	77	5	40	67	69	78	50	6
$p = 0.21$	2/4/3	15,489	67	88	100	0	99	100	100	100	112	5
$n = 16{,}000$	2/4/3*	14,715	135	0	95	1	91	93	94	95	99	5
$\|E\| \approx 222{,}351$	Pon	15,034	39	0	97	0	96	96	97	97	165	10
	War	10,420	313	0	67	2	63	64	65	67	126	5
	PW	10,380	315	0	67	2	62	64	65	67	208	6
	DFS	9,348	142	0	60	1	58	59	59	60	52	3
	1/2/0	639	176	0	4	1	2	2	2	4	32	5
	1/2/3	8,783	1,238	0	55	8	34	39	43	56	721	67
	1/4/0	1,037	151	0	6	1	4	5	5	6	27	1
	1/4/3	10,576	113	0	66	1	64	65	65	66	604	16
	2/2/0	14,248	1,587	0	89	10	26	72	83	92	50	3
pref. attach.	2/2/3	14,534	969	0	91	6	59	80	85	93	182	80
$n_0 = 13$	2/4/0	14,878	720	0	93	5	65	84	91	94	51	4
$d = 26$	2/4/3	15,102	422	0	94	3	80	88	93	95	139	33
$n = 16{,}000$	2/4/3*	15,004	450	0	94	3	73	88	91	95	111	4
$\|E\| = 207{,}843$	Pon	14,754	18	0	92	0	92	92	92	92	165	13
	War	16,000	0	100	100	0	100	100	100	100	0	0
	PW	16,000	0	100	100	0	100	100	100	100	0	0
	DFS	13,923	14	0	87	0	87	87	87	87	55	4
	1/2/0	737	242	0	5	2	1	2	3	5	29	5
	1/2/3	11,818	852	0	74	5	60	62	67	75	458	59
	1/4/0	1,304	188	0	8	1	6	6	7	8	25	1
	1/4/3	12,885	122	0	81	1	78	79	80	81	369	13
	2/2/0	13,867	1,090	0	87	7	48	69	80	89	49	4
hyperbolic	2/2/3	14,998	480	0	94	3	81	87	90	95	139	38
$d = 26$	2/4/0	14,299	554	0	89	3	64	84	89	90	50	4
$\gamma = 3$	2/4/3	15,237	210	0	95	1	88	93	95	96	119	16
$n = 16{,}000$	2/4/3*	14,727	423	0	92	3	77	87	89	93	101	4
$\|E\| \approx 224{,}369$	Pon	14,971	77	0	94	0	91	93	93	94	169	10
	War	16,000	0	100	100	0	100	100	100	100	0	0
	PW	16,000	0	100	100	0	100	100	100	100	0	0
	DFS	13,769	19	0	86	0	86	86	86	86	55	3

Table 3 *Large Set:* $n = 100{,}000$ and $d = 10$.

		ℓ			ρ in %						t	
graph class	algo	μ	σ	wins	μ	σ	P_0	P_5	P_{10}	P_{50}	μ	σ
chain $k = 1{,}000$ $l = 100$ $p = 0.01$ $n = 100{,}000$ $\|E\| \approx 599{,}468$	2/4/0	69,738	8,809	0	82	8	35	69	72	84	2,345	387
	2/4/1	84,335	3,661	0	99	0	98	99	99	99	3,997	395
	2/4/2	84,884	3,511	0	100	0	100	100	100	100	5,334	603
	2/4/3	84,909	3,500	50	100	0	100	100	100	100	6,591	833
	War	68,212	2,071	0	80	4	76	77	77	79	3,587	192
pref. attach. $n_0 = 5$ $d = 10$ $n = 100{,}000$ $\|E\| = 499{,}979$	2/4/0	80,190	8,064	0	82	8	49	58	71	86	2,612	778
	2/4/1	80,380	7,681	0	82	8	51	59	71	86	4,385	1,308
	2/4/2	80,510	7,459	0	82	8	53	60	71	86	5,974	1,957
	2/4/3	80,606	7,322	0	83	7	54	60	71	86	7,465	2,660
	War	97,685	52	50	100	0	100	100	100	100	4,110	588
hyperbolic $d = 10$ $\gamma = 3$ $n = 100{,}000$ $\|E\| \approx 599{,}680$	2/4/0	84,202	4,582	0	85	5	60	76	82	87	2,641	358
	2/4/1	88,147	3,772	0	89	4	68	82	86	91	3,978	591
	2/4/2	88,426	3,519	0	90	4	70	82	87	91	5,015	858
	2/4/3	88,494	3,400	0	90	3	71	83	87	91	5,963	1,076
	War	98,710	84	50	100	0	100	100	100	100	3,910	294
small world $d = 10$ $\beta = 0.3$ $n = 100{,}000$ $\|E\| = 500{,}000$	2/4/0	86,928	4,924	0	89	5	68	80	83	91	2,441	401
	2/4/1	90,810	4,038	0	93	4	76	86	89	95	3,457	531
	2/4/2	91,171	3,758	0	94	4	78	86	89	95	4,275	837
	2/4/3	91,253	3,590	0	94	4	79	87	89	95	5,072	1,148
	War	97,212	44	50	100	0	100	100	100	100	3,357	223

References

1 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, 1995. doi:10.1145/210332.210337.

2 Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, USA, January 2002 (SODA 2002)*, pages 623–632, 2002. URL: http://dl.acm.org/citation.cfm?id=545464.

3 Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999. doi:10.1126/science.286.5439.509.

4 Cristina Bazgan, Miklos Santha, and Zsolt Tuza. On the approximation of finding a(nother) Hamiltonian cycle in cubic Hamiltonian graphs. *Journal of Algorithms*, 31(1):249–268, 1999. Conference version at STACS 1998. doi:10.1006/jagm.1998.0998.

5 Andreas Björklund. Determinant sums for undirected Hamiltonicity. *SIAM Journal on Computing*, 43(1):280–299, 2014. Conference version at FOCS 2010. doi:10.1137/110839229.

6 Andreas Björklund and Thore Husfeldt. Finding a path of superlogarithmic length. *SIAM Journal on Computing*, 32(6):1395–1402, 2003. doi:10.1137/S0097539702416761.

7 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Narrow sieves for parameterized paths and packings, 2010. URL: http://arxiv.org/abs/1007.1161.

8 Andreas Björklund, Thore Husfeldt, and Sanjeev Khanna. Approximating longest directed paths and cycles. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming, Turku, Finland, July 2004 (ICALP 2004)*, pages 222–233, 2004. doi:10.1007/978-3-540-27836-8_21.

9 Hans L. Bodlaender. On linear time minor tests with depth-first search. *Journal of Algorithms*, 14:1–23, 1993. Conference version at WADS 1989. doi:10.1006/jagm.1993.1001.

10 R.W. Bulterman, F.W. van der Sommen, G. Zwaan, T. Verhoeff, A.J.M. van Gasteren, and W.H.J. Feijen. On computing a longest path in a tree. *Information Processing Letters*, 81(2):93–96, 2002. doi:10.1016/S0020-0190(01)00198-3.

11 Tomás Feder, Rajeev Motwani, and Carlos Subi. Approximating the longest cycle problem in sparse graphs. *SIAM Journal on Computing*, 31(5):1596–1607, 2002. doi:10.1137/S0097539701395486.

12 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharath Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348:207–216, 2005. Conference version at ICALP 2004. doi:10.1016/j.tcs.2005.09.013.

13 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharath Suri, and Jian Zhang. Graph distances in the data-stream model. *SIAM Journal on Computing*, 38:1709–1727, 2008. doi:10.1137/070683155.

14 Harold N. Gabow. Finding paths and cycles of superpolylogarithmic length. *SIAM Journal on Computing*, 36(6):1648–1671, 2007. doi:10.1137/S0097539704445366.

15 Harold N. Gabow and Shuxin Nie. Finding long paths, cycles and circuits. In *Proceedings of the 19th International Symposium on Algorithms and Computation, Gold Coast, Australia, December 2008 (ISAAC 2008)*, pages 752–753, 2008. doi:10.1007/978-3-540-92182-0_66.

16 Venkatesan Guruswami and Krzysztof Onak. Superlinear lower bounds for multipass graph processing. *Electronic Colloquium on Computational Complexity*, 2014. Conference version at CCC 2013. URL: http://eccc.hpi-web.de/report/2013/002/.

17 David Karger, Rajeev Motwani, and G.D.S. Ramkumar. On approximating the longest path in a graph. *Algorithmica*, 18:82–98, 1997. doi:10.1007/BF02523689.

18 Fatemeh Keshavarz-Kohjerdia, Alireza Bagherib, and Asghar Asgharian-Sardroudb. A linear-time algorithm for the longest path problem in rectangular grid graphs. *Discrete Applied Mathematics*, 160(3):210–217, 2012. doi:10.1016/j.dam.2011.08.010.

19 Ioannis Koutis. Faster algebraic algorithms for path and packing problems. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming, Reykjavik, Iceland, July 2008 (ICALP 2008)*, pages 575–586, 2008. doi:10.1007/978-3-540-70575-8_47.

20 Dmitri Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguñá. Hyperbolic geometry of complex networks. *Physical Review E*, 82, 2010. doi:10.1103/PhysRevE.82.036106.

21 Minko Markov, Mugurel Ionuţ Andreica, Krassimir Manev, and Nicolae Ţăpuş. A linear time algorithm for computing longest paths in cactus graphs. *Serdica Journal of Computing*, 6(3), 2012. URL: http://serdica-comp.math.bas.bg/index.php/serdicajcomputing/article/view/158.

22 Burkhard Monien. How to find long paths efficiently. *Annals of Discrete Mathematics*, 25:239–254, 1985. URL: https://digital.ub.uni-paderborn.de/hs/content/titleinfo/42079.

23 Muthu Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2):67 pages, 2005. URL: http://algo.research.googlepages.com/eight.ps.

24 Ira Pohl. A method for finding Hamilton paths and knight's tours. *Communications of the ACM*, 10(7):446–449, 1967. doi:10.1145/363427.363463.

25 Ira Pohl and Larry Stockmeyer. Pohl-Warnsdorf – revisited. In *Proceedings of the International Conference on Intelligent Systems and Control, Honolulu, Hawaii, USA, August 2004 (ISC 2004)*, 2004. URL: https://users.soe.ucsc.edu/~pohl/Papers/Pohl_Stockmeyer_full.pdf.

26 Lajos L. Pongrácz. A greedy approximation algorithm for the longest path problem in undirected graphs, 2012. URL: http://arxiv.org/abs/1209.2503v2.

27 Robert Sedgewick and Kevin Wayne. *Algorithms*. Addison-Wesley Professional, 2011.

28 Ryuhei Uehara and Yushi Uno. On computing longest paths in small graph classes. *International Journal of Foundations of Computer Science*, 18(5), 2007. doi:10.1142/S0129054107005054.

29 Sundar Vishwanathan. An approximation algorithm for finding long paths in Hamiltonian graphs. *Journal of Algorithms*, 50(2):246–256, 2004. Conference version at SODA 2000. doi:10.1016/S0196-6774(03)00093-2.

30 Moritz von Looz, Christian L. Staudt, Henning Meyerhenke, and Roman Prutkin. Fast generation of complex networks with underlying hyperbolic geometry, 2015. URL: http://arxiv.org/abs/1501.03545.

31 Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998. doi:10.1038/30918.

32 Ryan Williams. Finding paths of length k in $O^*(2^k)$ time. *Information Processing Letters*, 109:315–318, 2009. doi:10.1016/j.ipl.2008.11.004.

A Note On Spectral Clustering[*][†]

Pavel Kolev[1] and Kurt Mehlhorn[2]

1 **Max-Planck-Institut für Informatik, Saarbrücken, Germany**
pkolev@mpi-inf.mpg.de

2 **Max-Planck-Institut für Informatik, Saarbrücken, Germany**
mehlhorn@mpi-inf.mpg.de

Abstract

Spectral clustering is a popular and successful approach for partitioning the nodes of a graph into clusters for which the ratio of outside connections compared to the volume (sum of degrees) is small. In order to partition into k clusters, one first computes an approximation of the bottom k eigenvectors of the (normalized) Laplacian of G, uses it to embed the vertices of G into k-dimensional Euclidean space $\mathbb{R}^k$, and then partitions the resulting points via a k-means clustering algorithm. It is an important task for theory to explain the success of spectral clustering.

Peng et al. (COLT, 2015) made an important step in this direction. They showed that spectral clustering provably works if the gap between the $(k+1)$-th and the k-th eigenvalue of the normalized Laplacian is sufficiently large. They proved a structural and an algorithmic result. The algorithmic result needs a considerably stronger gap assumption and does not analyze the standard spectral clustering paradigm; it replaces spectral embedding by heat kernel embedding and k-means clustering by locality sensitive hashing.

We extend their work in two directions. Structurally, we improve the quality guarantee for spectral clustering by a factor of k and simultaneously weaken the gap assumption. Algorithmically, we show that the standard paradigm for spectral clustering works. Moreover, it even works with the same gap assumption as required for the structural result.

1998 ACM Subject Classification G.2.2 [Discrete Mathematics] Graph Theory, Graph Algorithms, G.3 [Probability and Statistics] Probabilistic Algorithms (including Monte Carlo)

Keywords and phrases spectral embedding, k-means clustering, power method, gap assumption

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.57

1 Introduction

A *cluster* in an undirected graph $G = (V, E)$ is a set S of nodes whose volume is large compared to the number of outside connections. Formally, we define the *conductance* of S by $\phi(S) = \left|E(S, \overline{S})\right| / \mu(S)$, where $\mu(S) = \sum_{v \in S} \deg(v)$ is the *volume* of S. The k-way partitioning problem for graphs asks to partition the vertices of a graph such that the conductance of each block of the partition is small (formal definition below). This problem arises in many applications, e.g., image segmentation and exploratory data analysis. We refer to the survey [10] for additional information. A popular and very successful approach to clustering [4, 8, 10] is *spectral clustering*. One first computes an approximation of the bottom k eigenvectors of the (normalized) Laplacian of G, uses it to embed the vertices of G into k-dimensional Euclidean space $\mathbb{R}^k$, and then partitions the resulting points via a

[*] The full version of the paper is available at http://arxiv.org/abs/1509.09188.

[†] This work has been funded by the Cluster of Excellence "Multimodal Computing and Interaction" within the Excellence Initiative of the German Federal Government.

k-means clustering algorithm. It is an important task for theory to explain the success of spectral clustering. Recently, Peng et al. [7] made an important step in this direction. They showed that spectral clustering provably works if the $(k+1)$-th and the k-th eigenvalue of the normalized Laplacian differ sufficiently. In order to explain their result, we need some notation.

Let $\mathcal{L}_G = I - D^{-1/2}AD^{-1/2}$ be the normalized Laplacian matrix of G, where D is the diagonal degree matrix and A is the adjacency matrix, and let $f_j \in \mathbb{R}^V$ be the eigenvector corresponding to the j-th smallest eigenvalue λ_j of $\mathcal{L}_G$. The *spectral embedding map* $F : V \to \mathbb{R}^k$ is defined by

$$F(u) = \frac{1}{\sqrt{d_u}} \left(f_1(u), \ldots, f_k(u)\right)^{\mathrm{T}}, \quad \text{for all vertices } u \in V. \tag{1}$$

Peng et al. [7] construct a k-means instance $\mathcal{X}_V$ by inserting d_u many copies of the vector $F(u)$ into $\mathcal{X}_V$, for every vertex $u \in V$.

Let $\mathcal{X}$ be a set of vectors of the same dimension. Then

$$\triangle_k(\mathcal{X}) \triangleq \min_{\text{partition } (X_1,\ldots,X_k) \text{ of } \mathcal{X}} \sum_{i=1}^{k} \sum_{x \in X_i} \|x - c_i\|^2, \quad \text{where} \quad c_i = \frac{1}{|X_i|} \sum_{x \in X_i} x,$$

is the optimal cost of clustering $\mathcal{X}$ into k sets. An α-approximate clustering algorithm returns a k-way partition $(A_1, \ldots, A_k)$ and centers $c_1, \ldots, c_k$ such that

$$\mathrm{Cost}(\{A_i, c_i\}_{i=1}^k) \triangleq \sum_{i=1}^{k} \sum_{x \in A_i} \|x - c_i\|^2 \leqslant \alpha \cdot \triangle_k(\mathcal{X}). \tag{2}$$

The *order k conductance constant* $\rho(k)$ is a well studied worst case guarantee for k-way partitioning that is defined by

$$\rho(k) = \min_{\text{disjoint nonempty } Z_1,\ldots,Z_k} \Phi(Z_1, \ldots, Z_k), \quad \text{where} \quad \Phi(Z_1, \ldots, Z_k) = \max_{i \in [1:k]} \phi(Z_i). \tag{3}$$

Lee et al. [3] connected $\rho(k)$ and the k-th smallest eigenvalue of the normalized Laplacian matrix $\mathcal{L}_G$ through the relation, also known as higher order Cheeger inequality,

$$\lambda_k/2 \leqslant \rho(k) \leqslant O(k^2)\sqrt{\lambda_k}. \tag{4}$$

In this work, we focus attention on the *order k partition constant* $\widehat{\rho}(k)$ of G, defined by

$$\widehat{\rho}(k) \triangleq \min_{\text{partition } (P_1,\ldots,P_k) \text{ of } V} \Phi(P_1, \ldots P_k), \quad \text{where} \quad \Phi(Z_1, \ldots, Z_k) = \max_{i \in [1:k]} \phi(Z_i).$$

In a consecutive work, inspired by partitioning graphs into expanders, Oveis Gharan and Trevisan [6] proved the following relation

$$\rho(k) \leqslant \widehat{\rho}(k) \leqslant k\rho(k). \tag{5}$$

We are now ready to state the main structural result by Peng et al [7].

▶ **Theorem 1.1** ([7, Theorem 1.2]). *Let $k \geqslant 3$ and $(P_1, \ldots, P_k)$ be a k-way partition of V with $\Phi(P_1, \ldots, P_k) = \widehat{\rho}(k)$. Let G be a graph that satisfies the gap assumption*[1]

$$\delta = \frac{2 \cdot 10^5 \cdot k^3}{\Upsilon} \in (0, 1/2], \quad \textit{where} \quad \Upsilon \triangleq \frac{\lambda_{k+1}}{\widehat{\rho}(k)}. \tag{6}$$

[1] Note that $\lambda_k/2 \leqslant \widehat{\rho}(k)$, see (4,5). Thus the assumption implies $\lambda_k/2 \leqslant \widehat{\rho}(k) = \delta\lambda_{k+1}/(2 \cdot 10^5 \cdot k^3)$, i.e., there is a substantial gap between the k-th and the $(k+1)$-th eigenvalue.

Let $(A_1, \ldots, A_k)$ be the k-way partition[2] *of V returned by an α-approximate k-means algorithm applied to $\mathcal{X}_V$. Then the following statements hold (after suitable renumbering of one of the partitions):*

1. $\mu(A_i \triangle P_i) \leqslant \alpha\delta \cdot \mu(P_i)$*, and*
2. $\phi(A_i) \leqslant (1 + 2\alpha\delta) \cdot \phi(P_i) + 2\alpha\delta$*,*

where the symmetric difference $A_i \triangle P_i = (A_i \backslash P_i) \cup (P_i \backslash A_i)$.

Under the stronger gap assumption $\delta = 2 \cdot 10^5 \cdot k^5 / \Upsilon \in (0, 1/2]$, they showed how to obtain a partition in time $O(m \cdot \operatorname{poly}\log(n))$ with essentially the guarantees stated in Theorem 1.1, where $m = |E|$ is the number of edges in G and $n = |V|$ is the number of nodes.

However, their algorithmic result does not analyze the standard spectral clustering paradigm, since it replaces spectral embedding by heat kernel embedding and k-means clustering by locality sensitive hashing. Therefore, their algorithmic result does not explain the success of the standard spectral clustering paradigm.

Our Results

We strengthen the approximation guarantees in Theorem 1.1 by a factor of k and simultaneously weaken the gap assumption. As a consequence, the variant of Lloyd's k-means algorithm analyzed by Ostrovsky et al. [5] applied to[3] $\widetilde{\mathcal{X}_V}$ achieves the improved approximation guarantees in time $O(m(k^2 + \frac{\ln n}{\lambda_{k+1}}))$ with constant probability. Table 1 summarizes these results.

Let $\mathcal{O}$ be the set of all k-way partitions $(P_1, \ldots, P_k)$ with $\Phi(P_1, \ldots, P_k) = \widehat{\rho}(k)$, i.e., the set of all partitions that achieve the order k partition constant. Let

$$\widehat{\rho}_{\text{avr}}(k) \triangleq \min_{(P_1,\ldots,P_k)\in\mathcal{O}} \frac{1}{k}\sum_{i=1}^{k} \phi(P_i)$$

be the *minimal average conductance* over all k-way partitions in $\mathcal{O}$. The minimal average conductance can be considerably smaller than the order k partition constant. Consider a graph consisting of one clique of size $S_k = f(n) = o(n/k^{3/2})$, $k-1$ cliques of size $(n - f(n))/(k-1)$ each, and k additional edges that connect the cliques in the form of a ring. Then $\phi(S_i) \approx k^2/n^2$ for $1 \leqslant i \leqslant k-1$ and $\phi(S_k) \approx 1/f(n)^2$. Thus $\widehat{\rho}(k) = \max_i \phi(S_i) = \phi(S_k) \approx 1/f(n)^2$ and $\widehat{\rho}_{\text{avr}}(k) = (1/k)\sum_{1 \leqslant i \leqslant k} \phi(S_k) \approx k^2/n^2 + (1/k) \cdot (1/f(n)^2) \approx \widehat{\rho}(k)/k$.

For the remainder of this paper we denote by $(P_1, \ldots, P_k)$ a k-way partition of V that achieves $\widehat{\rho}_{\text{avr}}(k)$. In the full version of the paper, we give an analogous relation to (5) for $\widehat{\rho}_{\text{avr}}(k)$. We state now our main result.

▶ **Theorem 1.2** (Main Theorem).

(a) (Existence of a Good Clustering) *Let $k \geqslant 3$. Let G be a graph satisfying the gap assumption*

$$\delta = \frac{20^4 \cdot k^3}{\Psi} \in (0, 1/2], \quad \textit{where} \quad \Psi \triangleq \frac{\lambda_{k+1}}{\widehat{\rho}_{\text{avr}}(k)}. \tag{7}$$

Let $(A_1, \ldots, A_k)$ be the k-way partition returned by an α-approximate clustering algorithm applied to the spectral embedding $\mathcal{X}_V$. Then for every $i \in [1 : k]$ the following two statements hold (after suitable renumbering of one of the partitions):

[2] The k-means algorithm returns a partition of $\mathcal{X}_V$. One may assume w.l.o.g. that all copies of $F(u)$ are put into the same cluster of $\mathcal{X}_V$. Thus the algorithm also partitions V.

[3] $\widetilde{\mathcal{X}_V}$ is defined as $\mathcal{X}_V$ but in terms of approximate eigenvectors, see Subsection 2.3.

Table 1 A comparison of the results in Peng et al. [7] and our results. The parameter $\delta \in (0, 1/2]$ relates the approximation guarantees with the gap assumption.

	Gap Assumption	Partition Quality	Running Time
Peng et al. [7]	$\delta = 2 \cdot 10^5 \cdot k^3/\Upsilon$	$\mu(A_i \triangle P_i) \leqslant \alpha\delta \cdot \mu(P_i)$ $\phi(A_i) \leqslant (1 + 2\alpha\delta)\,\phi(P_i) + 2\alpha\delta$	Existential result
This paper	$\delta = 20^4 \cdot k^3/\Psi$	$\mu(A_i \triangle P_i) \leqslant \frac{\alpha\delta}{10^3 k} \cdot \mu(P_i)$ $\phi(A_i) \leqslant \left(1 + \frac{2\alpha\delta}{10^3 k}\right) \phi(P_i) + \frac{2\alpha\delta}{10^3 k}$	Existential result
Peng et al. [7]	$\delta = 2 \cdot 10^5 \cdot k^5/\Upsilon$	$\mu(A_i \triangle P_i) \leqslant \frac{\delta \log^2 k}{k^2} \cdot \mu(P_i)$ $\phi(A_i) \leqslant \left(1 + \frac{2\delta \log^2 k}{k^2}\right) \phi(P_i) + \frac{2\delta \log^2 k}{k^2}$	$O\,(m \cdot \operatorname{poly}\log(n))$
This paper	$\delta = 20^4 \cdot k^3/\Psi$ $\delta \leqslant k/10^9$ $\triangle_k(\mathcal{X}_V) \geqslant n^{-O(1)}$	$\mu(A_i \triangle P_i) \leqslant \frac{2\delta}{10^3 k} \cdot \mu(P_i)$ $\phi(A_i) \leqslant \left(1 + \frac{4\delta}{10^3 k}\right) \phi(P_i) + \frac{4\delta}{10^3 k}$	$O\left(m\left(k^2 + \frac{\ln n}{\lambda_{k+1}}\right)\right)$

1. $\mu(A_i \triangle P_i) \leqslant \frac{\alpha\delta}{10^3 k} \cdot \mu(P_i)$, *and*
2. $\phi(A_i) \leqslant \left(1 + \frac{2\alpha\delta}{10^3 k}\right) \cdot \phi(P_i) + \frac{2\alpha\delta}{10^3 k}$.

(b) (An Efficient Algorithm) *If in addition* $\delta \leqslant k/10^9$ *and*[4] $\triangle_k(\mathcal{X}_V) \geqslant n^{-O(1)}$, *then the variant of Lloyd's algorithm analyzed by Ostrovsky et al. [5] applied to* $\widetilde{\mathcal{X}_V}$ *returns in time* $O(m(k^2 + \frac{\ln n}{\lambda_{k+1}}))$ *with constant probability a partition* $(A_1, \ldots, A_k)$ *such that for every* $i \in [1 : k]$ *the following two statements hold (after suitable renumbering of one of the partitions):*

3. $\mu(A_i \triangle P_i) \leqslant \frac{2\delta}{10^3 k} \cdot \mu(P_i)$, *and*
4. $\phi(A_i) \leqslant \left(1 + \frac{4\delta}{10^3 k}\right) \cdot \phi(P_i) + \frac{4\delta}{10^3 k}$.

Part (b) of Theorem 1.2 gives a theoretical support for the practical success of spectral clustering based on approximate spectral embedding followed by k-means clustering. Moreover, if $k \leqslant \operatorname{poly}(\log n)$ and $\lambda_{k+1} \geqslant \operatorname{poly}(\log n)$, our algorithm works in nearly linear time. Previous papers [3, 7, 9] replaced k-means clustering by other techniques for their algorithmic results.

The k-means algorithm in [5] is efficient only for inputs $\mathcal{X}$ for which some partition into k clusters is much better than any partition into $k-1$ clusters. The authors proved that the algorithm is efficient for inputs $\mathcal{X}$ satisfying $\triangle_k(\mathcal{X}) \leqslant \varepsilon^2 \triangle_{k-1}(\mathcal{X})$ for some $\varepsilon \in (0, \varepsilon_0]$, where $\varepsilon_0 = 6/10^7$, stated that the result should also hold for a larger ε_0, and mentioned that they did not attempt to maximize ε_0. For the proof of part (b) of Theorem 1.2, we show in Section 5 that $\widetilde{\mathcal{X}_V}$ satisfies this assumption. In this proof, we need $\delta \leqslant k \cdot \varepsilon_0/600 = k/10^9$.

One of the reviewers suggested to include a numerical example. Consider a graph consisting of k cliques of size n/k each plus k additional edges that connect the cliques in the form of a ring. Such a graph is about the easiest input for a clustering algorithm. Then $\widehat{\rho}_{\text{avr}}(k) = \widehat{\rho}(k) \approx (k/n)^2$. For the gap assumption to hold we need $\lambda_{k+1} \geqslant 2 \cdot 20^4 \cdot k^3 \cdot \widehat{\rho}_{\text{avr}}(k)$. Since $\lambda_{k+1} \leqslant 2$, this implies $n \geqslant 400 \cdot k^{2.5}$. For small k, this is a modest requirement on the size of the graph.

[4] The case $\triangle_k(\mathcal{X}_V) \leqslant n^{-O(1)}$ constitutes a trivial clustering problem. For technical reasons, we have to exclude too easy inputs.

For the algorithmic result, we need in addition $\delta \leqslant k \cdot \varepsilon_0/600$. For the gap condition to hold, we need $2 \geqslant \lambda_{k+1} \geqslant (600/\varepsilon_0 k) \cdot 20^4 \cdot k^3 \cdot (k^2/n^2)$ or $n \geqslant 4\sqrt{3} \cdot 10^3 \cdot k^2/\sqrt{\varepsilon_0}$. For $\varepsilon_0 = 6/10^7$, this amounts to $n \geqslant 4\sqrt{5} \cdot 10^6 \cdot k^2$, a quite large lower bound on n.

Our statement above that Part (b) of Theorem 1.2 gives a theoretical support for the practical success of spectral clustering based on approximate spectral embedding followed by k-means clustering therefore has to be taken with a grain of salt. It is only an asymptotic statement and does not explain the good behavior on small graphs.

2 Highlights of Our Technical Contribution

2.1 Exact Spectral Embedding – Notation

We use the notation adopted by Peng et al. [7]. Let $f_j \in \mathbb{R}^V$ be the eigenvector corresponding to the j-th smallest eigenvalue λ_j of $\mathcal{L}_G$, and let $\overline{g_i} = \frac{D^{1/2}\chi_{P_i}}{\|D^{1/2}\chi_{P_i}\|}$ be the normalized indicator vector associated with the i-th optimal cluster $P_i \subset V$.

Since the eigenvectors $\{f_i\}_{i=1}^n$ form an orthonormal basis of $\mathbb{R}^n$, each normalized indicator vector $\overline{g_i}$ can be expressed as $\overline{g_i} = \sum_{j=1}^n \alpha_j^{(i)} f_j$, for all $i \in [1:k]$. Its *projection* into the subspace spanned by the bottom k eigenvectors is given by $\widehat{f_i} = \sum_{j=1}^k \alpha_j^{(i)} f_j$. Peng et al. [7] proved that if the gap parameter Υ is large enough then $\mathrm{span}(\{\widehat{f_i}\}_{i=1}^k) = \mathrm{span}(\{f_i\}_{i=1}^k)$ and hence the bottom k eigenvectors can be expressed by $f_i = \sum_{j=1}^k \beta_j^{(i)} \widehat{f_j}$, for all $i \in [1:k]$. We show that similar statements hold with substituted gap parameter Ψ.

A corner stone in the analysis of spectral clustering is to prove the existence of exactly k directions near which all spectrally embedded vectors are closely concentrated. These estimation centers are defined by

$$p^{(i)} = \frac{1}{\sqrt{\mu(P_i)}} \left(\beta_i^{(1)}, \ldots, \beta_i^{(k)}\right)^{\mathrm{T}}. \tag{8}$$

Our analysis crucially relies on the isometric properties of the following square matrix. Let $\mathbf{B} \in \mathbb{R}^{k \times k}$ be a matrix defined by $\mathbf{B}_{j,i} = \beta_j^{(i)}$, for every $i, j \in [1:k]$.

2.2 Exact Spectral Embedding – Structural Results

The proof of Theorem 1.2 (a) follows the proof-structure of [7, Theorem 1.2] in Peng et al., but improves upon it in essential ways.

Our key technical insight is that the matrices $\mathbf{B}\mathbf{B}^{\mathrm{T}}$ and $\mathbf{B}^{\mathrm{T}}\mathbf{B}$ are close to the identity matrix. The proof of Theorem 2.1 appears in the full version of the paper.

▶ **Theorem 2.1** (Matrix $\mathbf{B}\mathbf{B}^{\mathrm{T}}$ is Close to Identity Matrix). *If $\Psi \geqslant 10^4 \cdot k^3/\varepsilon^2$ and $\varepsilon \in (0,1)$ then for all distinct $i, j \in [1:k]$ it holds*

$$1 - \varepsilon \leqslant \langle \mathbf{B}_{i,:}, \mathbf{B}_{i,:} \rangle \leqslant 1 + \varepsilon \quad \textit{and} \quad |\langle \mathbf{B}_{i,:}, \mathbf{B}_{j,:} \rangle| \leqslant \sqrt{\varepsilon}.$$

Informally, Theorem 2.1 implies that each normalized indicator vector $\overline{g_i}$ is close to the corresponding eigenvector f_i. This gives a simple and intuitive explanation for the success of spectral clustering.

To see this, let $\mathbf{F}_k, \widehat{\mathbf{F}}_k \in \mathbb{R}^{k \times k}$ be matrices whose i-th column is f_i and $\widehat{f_i}$, respectively. The projection matrix $\mathbf{P}_k$ into the k-th principle subspace of $\mathcal{L}_G$ is given by $\mathbf{P}_k = \mathbf{F}_k \mathbf{F}_k^{\mathrm{T}}$ and since $\widehat{\mathbf{F}}_k \mathbf{B} = \mathbf{F}_k$, by Theorem 2.1 it follows that $\widehat{\mathbf{F}}_k \widehat{\mathbf{F}}_k^{\mathrm{T}} \approx \mathbf{F}_k \mathbf{F}_k^{\mathrm{T}}$. Therefore, each projected indicator vector satisfies $\widehat{f_i} \approx f_i$. This implies $\alpha^{(i)} \approx \boldsymbol{\chi}_i$ and hence we have $\overline{g_i} \approx f_i$.

Formally, Theorem 2.1 allows us to improve the separation guarantee between any pair of estimation centers by a factor of k over [7, Lemma 4.3], measured in terms of the Euclidean distance.

▶ **Lemma 2.2.** *If $\delta = 20^4 \cdot k^3/\Psi \in (0,1]$ then for every $i \in [1:k]$ it holds that $\left\|p^{(i)}\right\|^2 \in \left[1 \pm \sqrt{\delta}/4\right] \frac{1}{\mu(P_i)}$.*

Proof. By definition $p^{(i)} = \frac{1}{\sqrt{\mu(P_i)}} \cdot \mathbf{B}_{i,:}$ and Theorem 2.1 yields $\|\mathbf{B}_{i,:}\|^2 \in [1 \pm \sqrt{\delta}/4]$. ◀

▶ **Lemma 2.3** (Larger Distance Between Estimation Centers)**.** *If $\delta = 20^4 \cdot k^3/\Psi \in (0,1/2]$ then for any distinct $i,j \in [1:k]$ it holds that $\left\|p^{(i)} - p^{(j)}\right\|^2 \geqslant \left[2 \cdot \min\left\{\mu(P_i), \mu(P_j)\right\}\right]^{-1}$.*

Proof. Since $p^{(i)}$ is a row of matrix B, Theorem 2.1 with $\varepsilon = \sqrt{\delta}/4$ yields

$$\left\langle \frac{p^{(i)}}{\left\|p^{(i)}\right\|}, \frac{p^{(j)}}{\left\|p^{(j)}\right\|} \right\rangle = \frac{\langle \mathbf{B}_{i,:}, \mathbf{B}_{j,:}\rangle}{\|\mathbf{B}_{i,:}\| \, \|\mathbf{B}_{j,:}\|} \leqslant \frac{\sqrt{\varepsilon}}{1-\varepsilon} = \frac{2\delta^{1/4}}{3}.$$

W.l.o.g. assume that $\left\|p^{(i)}\right\|^2 \geqslant \left\|p^{(j)}\right\|^2$, say $\left\|p^{(j)}\right\| = \alpha \cdot \left\|p^{(i)}\right\|$ for some $\alpha \in (0,1]$. Then by Lemma 2.2 we have $\left\|p^{(i)}\right\|^2 \geqslant (1 - \sqrt{\delta}/4) \cdot \left[\min\left\{\mu(P_i), \mu(P_j)\right\}\right]^{-1}$, and hence

$$\begin{aligned}\left\|p^{(i)} - p^{(j)}\right\|^2 &= \left\|p^{(i)}\right\|^2 + \left\|p^{(j)}\right\|^2 - 2\left\langle \frac{p^{(i)}}{\left\|p^{(i)}\right\|}, \frac{p^{(j)}}{\left\|p^{(j)}\right\|} \right\rangle \left\|p^{(i)}\right\| \left\|p^{(j)}\right\| \\ &\geqslant \left(\alpha^2 - \frac{4\delta^{1/4}}{3} \cdot \alpha + 1\right) \left\|p^{(i)}\right\|^2 \geqslant \left[2 \cdot \min\left\{\mu(P_i), \mu(P_j)\right\}\right]^{-1}.\end{aligned}$$ ◀

The observation that Υ can be replaced by Ψ in all statements in [7] is technically easy. However, this is crucial for Theorem 1.2 (b), since it yields an improved version of [7, Lemma 4.5] showing that a weaker by a factor of k assumption is sufficient. Due to space limitation, we defer the proof of Lemma 2.4 to the full version of the paper.

▶ **Lemma 2.4.** *Let $(P_1, \ldots, P_k)$ and $(A_1, \ldots, A_k)$ are partitions of the vector set. Suppose for every permutation $\pi : [1:k] \to [1:k]$ there is an index $i \in [1:k]$ such that*

$$\mu(A_i \triangle P_{\pi(i)}) \geqslant \frac{2\varepsilon}{k} \cdot \mu(P_{\pi(i)}), \tag{9}$$

where $\varepsilon \in (0,1)$ is a parameter. If $\delta = 20^4 \cdot k^3/\Psi \in (0,1/2]$ and $\varepsilon \geqslant 64\alpha \cdot k^3/\Psi$ then

$$\mathrm{Cost}(\{A_i, c_i\}_{i=1}^k) > \frac{2k^2}{\Psi}\alpha.$$

With the above Lemmas in place, the proof of Theorem 1.2 (a) is then completed as in [7]. We give more details in Section 3.

Before we turn to Theorem 1.2 (b), we consider the variant of Lloyd's algorithm analyzed by Ostrovsky et al. [5] applied to $\mathcal{X}_V$. This algorithm is efficient for inputs $\mathcal{X}$ satisfying: some partition into k clusters is much better than any partition into $k-1$ clusters.

▶ **Theorem 2.5.** *[5, Theorem 4.15] Assuming that $\triangle_k(\mathcal{X}) \leqslant \varepsilon^2 \triangle_{k-1}(\mathcal{X})$ for $\varepsilon \in (0, 6/10^7]$, there is an algorithm that returns a solution of cost at most $[(1-\varepsilon^2)/(1-37\varepsilon^2)]\triangle_k(\mathcal{X})$ with probability at least $1 - O(\sqrt{\varepsilon})$ in time $O(nkd + k^3 d)$.*

In Section 4, we establish the assumption of Ostrovsky et al. [5] for $\mathcal{X}_V$.

▶ **Theorem 2.6** (Normalized Spectral Embedding is ε-separated). *Let G be a graph that satisfies the gap assumption $\delta = 20^4 \cdot k^3/\Psi \in (0, 1/2]$ and $\delta \leqslant k \cdot \varepsilon/600$, where $\varepsilon = 6/10^7$ is the Ostrovsky et al.'s constant. Then it holds*

$$\triangle_k(\mathcal{X}_V) \leqslant \varepsilon^2 \triangle_{k-1}(\mathcal{X}_V). \tag{10}$$

However, Theorem 2.6 is insufficient for Theorem 1.2 (b), since we need a similar result for the set $\widetilde{\mathcal{X}_V}$ formed by approximate eigenvectors. To overcome this issue we build upon the recent work by Boutsidis et al. [1] which shows that running an approximate k-means clustering algorithm on approximate eigenvectors obtained via the power method, yields an additive approximation to solving the k-means clustering problem on exact eigenvectors.

In order to state the connection, we need to introduce some of their notation.

2.3 Approximate Spectral Embedding – Notation

Let $Z \in \mathbb{R}^{n\times k}$ be a matrix whose rows represent n vectors that are to be partitioned into k clusters. For every k-way partition we associate an indicator matrix $X \in \mathbb{R}^{n\times k}$ that satisfies $X_{ij} = 1/\sqrt{|C_j|}$ if the i-th row $Z_{i,:}$ belongs to the j-th cluster C_j, and $X_{ij} = 0$ otherwise. We denote the optimal indicator matrix X_{opt} by

$$X_{\text{opt}} = \arg \min_{X\in\mathbb{R}^{n\times k}} \left\|Z - XX^{\mathrm{T}}Z\right\|_F^2 = \arg \min_{X\in\mathbb{R}^{n\times k}} \sum_{j=1}^{k} \sum_{u\in X_j} \|Z_{u,:} - c_j\|_2^2, \tag{11}$$

where $c_j = (1/|X_j|) \sum_{u\in X_j} Z_{u,:}$ is the center point of cluster C_j.

The normalized Laplacian matrix $\mathcal{L}_G \in \mathbb{R}^{n\times n}$ of a graph G is define by $\mathcal{L}_G = I - \mathcal{A}$, where $\mathcal{A} = D^{-1/2}AD^{-1/2}$ is the normalized adjacency matrix. Let $U_k \in \mathbb{R}^{n\times k}$ be a matrix composed of the bottom k orthonormal eigenvectors of $\mathcal{L}_G$ corresponding to the smallest eigenvalues $\lambda_1, \ldots, \lambda_k$. We define by $Y \triangleq U_k$ the canonical spectral embedding.

Our approximate spectral embedding is computed by the so called "**Power method**". Let $S \in \mathbb{R}^{n\times k}$ be a matrix whose entries are i.i.d. samples from the standard Gaussian distribution $N(0, 1)$ and p be a positive integer. Then the approximate spectral embedding $\widetilde{Y}$ is defined by the following process:

$$1)\ \mathcal{B} \triangleq I + \mathcal{A}; \quad 2)\ \text{Let } \widetilde{U}\widetilde{\Sigma}\widetilde{V}^{\mathrm{T}} \text{ be the SVD of } \mathcal{B}^p S; \quad \text{and} \quad 3)\ \widetilde{Y} \triangleq \widetilde{U} \in \mathbb{R}^{n\times k}. \tag{12}$$

We proceed by defining the normalized (approximate) spectral embedding. We construct a matrix $Y' \in \mathbb{R}^{m\times k}$ such that for every vertex $u \in V$ we add $\deg(u)$ many copies of the normalized row $U_k(u, :)/\sqrt{\deg(u)}$ to Y'. Formally, the normalized (approximate) spectral embedding Y' ($\widetilde{Y'}$) is defined by

$$Y' = \begin{pmatrix} \mathbf{1}_{\deg(1)} \frac{U_k(1,:)}{\sqrt{\deg(1)}} \\ \cdots \\ \mathbf{1}_{\deg(n)} \frac{U_k(n,:)}{\sqrt{\deg(n)}} \end{pmatrix}_{m\times k} \quad \text{and} \quad \widetilde{Y'} = \begin{pmatrix} \mathbf{1}_{\deg(1)} \frac{\widetilde{U}(1,:)}{\sqrt{\deg(1)}} \\ \cdots \\ \mathbf{1}_{\deg(n)} \frac{\widetilde{U}(n,:)}{\sqrt{\deg(n)}} \end{pmatrix}_{m\times k}, \tag{13}$$

where $\mathbf{1}_{\deg(i)}$ is all-one column vector with dimension $\deg(i)$.

Similarly to (11) we associate to Y' ($\widetilde{Y'}$) an indicator matrix X' ($\widetilde{X'}$) that satisfies $X'_{ij} = 1/\sqrt{\mu(C_j)}$ if the i-th row $Y'_{i,:}$ belongs to the j-th cluster C_j, and $X'_{ij} = 0$ otherwise. We may assume w.l.o.g. that a k-means algorithm outputs an indicator matrix X' such that all copies of row $U_k(v, :)/\sqrt{\deg(v)}$ belong to the same cluster, for every vertex $v \in V$.

We associate to matrices Y' and $\widetilde{Y'}$ the sets of points $\mathcal{X}_V$ and $\widetilde{\mathcal{X}_V}$ respectively. We present now a key connection between the spectral embedding map $F(\cdot)$, the optimal k-means cost $\triangle_k(\mathcal{X}_V)$ and matrices Y', X'_{opt}:

$$\left\|Y' - X'_{\text{opt}}\left(X'_{\text{opt}}\right)^{\mathrm{T}} Y'\right\|_F^2 = \sum_{j=1}^{k} \sum_{v \in C_j^\star} \deg(v) \left\|F(v) - c_j^\star\right\|_F^2 = \triangle_k(\mathcal{X}_V), \tag{14}$$

where each center satisfies $c_j^\star = \mu(C_j^\star)^{-1} \cdot \sum_{v \in C_j^\star} \deg(v) F(v)$ and $F(v) = Y_{v,:}/\sqrt{\deg(v)}$.

2.4 Approximate Spectral Embedding – Algorithmic Results

Our analysis relies on the proof techniques developed in [1, 2]. By adjusting these techniques (c.f. [1, Lemma 5] and [2, Lemma 7]) to our setting, we prove (in the full version of the paper) the following result for the symmetric positive semi-definite matrix $\mathcal{B}$ whose largest k singular values (eigenvalues) correspond to the eigenvectors $u_1, \ldots, u_k$ of $\mathcal{L}_G$.

▶ **Lemma 2.7.** *Let $\widetilde{U}\widetilde{\Sigma}\widetilde{V}^{\mathrm{T}}$ be the SVD of $\mathcal{B}^p S \in \mathbb{R}^{n \times k}$, where $p \geqslant 1$ and S is an $n \times k$ matrix of i.i.d. standard Gaussians. Let $\gamma_k = \frac{2-\lambda_{k+1}}{2-\lambda_k} < 1$ and fix $\delta, \epsilon \in (0,1)$. Then for any $p \geqslant \ln(8nk/\epsilon\delta) \big/ \ln(1/\gamma_k)$ with probability at least $1 - 2e^{-2n} - 3\delta$ it holds*

$$\left\|U_k U_k^{\mathrm{T}} - \widetilde{U}\widetilde{U}^{\mathrm{T}}\right\|_F \leqslant \epsilon.$$

We establish several technical Lemmas that combined with Lemma 2.7 allow us to apply the proof techniques in [1, Theorem 6]. More precisely, we prove in Subsection 5.1 that running an approximate k-means algorithm on a normalized approximate spectral embedding $\widetilde{Y'}$ computed by the power method, yields an approximate clustering of the normalized spectral embedding Y'.

▶ **Theorem 2.8.** *Compute matrix $\widetilde{Y'}$ via the power method with $p \geqslant \ln(8nk/\epsilon\delta) \big/ \ln(1/\gamma_k)$, where $\gamma_k = (2 - \lambda_{k+1})/(2 - \lambda_k) < 1$. Run on the rows of $\widetilde{Y'}$ an α-approximate k-means algorithm with failure probability δ_α. Let the outcome be a clustering indicator matrix $\widetilde{X'_\alpha} \in \mathbb{R}^{n \times k}$. Then with probability at least $1 - 2e^{-2n} - 3\delta_p - \delta_\alpha$ it holds*

$$\left\|Y' - \widetilde{X'_\alpha}\left(\widetilde{X'_\alpha}\right)^{\mathrm{T}} Y'\right\|_F^2 \leqslant (1 + 4\varepsilon) \cdot \alpha \cdot \left\|Y' - X'_{\text{opt}}\left(X'_{\text{opt}}\right)^{\mathrm{T}} Y'\right\|_F^2 + 4\varepsilon^2.$$

Our main technical contribution is to prove, in Subsection 5.2, that $\widetilde{\mathcal{X}_V}$ satisfies the assumption of Ostrovsky et al. [5]. Our analysis builds upon Theorem 2.6 and Theorem 2.8.

▶ **Theorem 2.9** (Approximate Normalized Spectral Embedding is ε-separated)**.** *Let G be a graph that satisfies the gap assumption $\delta = 20^4 \cdot k^3/\Psi \in (0, 1/2]$ and $\delta \leqslant k \cdot \varepsilon/600$, where $\varepsilon = 6/10^7$ is the Ostrovsky et al.'s constant. If the optimum cost*[5] *$\left\|Y' - X'_{\text{opt}}(X'_{\text{opt}})^{\mathrm{T}} Y'\right\|_F \geqslant n^{-O(1)}$ and the matrix $\widetilde{Y'}$ is constructed via the power method with $p \geqslant \Omega(\frac{\ln n}{\lambda_{k+1}})$, then w.h.p it holds $\triangle_k\left(\widetilde{\mathcal{X}_V}\right) < 5\varepsilon^2 \cdot \triangle_{k-1}\left(\widetilde{\mathcal{X}_V}\right)$.*

Based on the preceding results, we prove Theorem 1.2 (b) in Subsection 5.3.

[5] $\left\|Y' - X'_{\text{opt}}(X'_{\text{opt}})^{\mathrm{T}} Y'\right\|_F \geqslant n^{-O(1)}$ asserts a multiplicative approximation guarantee in Theorem 2.8.

3 The Proof of Part (a) of Theorem 1.2

The proof of part $(a.1)$ builds upon the following Lemmas. Recall that $\mathcal{X}_V$ contains d_u copies of $F(u)$ for each $u \in V$. W.l.o.g. we may restrict attention to clusterings of $\mathcal{X}_V$ that put all copies of $F(u)$ into the same cluster and hence induce a clustering of V. Let $(A_1, \dots, A_k)$ with cluster centers c_1 to c_k be a clustering of V. Its k-means cost is

$$\mathrm{Cost}(\{A_i, c_i\}_{i=1}^k) = \sum_{i=1}^{k} \sum_{u \in A_i} d_u \left\| F(u) - c_i \right\|^2 .$$

The proofs of Lemma 3.1 and Lemma 3.2 appear in the full version of the paper.

▶ **Lemma 3.1** ($(P_1, \dots, P_k)$ is a good k-means partition)**.** *If $\Psi > 4 \cdot k^{3/2}$ then there are vectors $\{p^{(i)}\}_{i=1}^k$ such that* $\mathrm{Cost}(\{P_i, p^{(i)}\}_{i=1}^k) \leqslant (1 + \frac{3k}{\Psi}) \cdot \frac{k^2}{\Psi}$.

▶ **Lemma 3.2** (Only partitions close to $(P_1, \dots, P_k)$ are good)**.** *Under the hypothesis of Theorem 1.2, the following holds. If for every permutation $\sigma : [1:k] \to [1:k]$ there exists an index $i \in [1:k]$ such that*

$$\mu(A_i \triangle P_{\sigma(i)}) \geqslant \frac{8\alpha\delta}{10^4 k} \cdot \mu(P_{\sigma(i)}), \quad \textit{then it holds} \quad \mathrm{Cost}(\{A_i, c_i\}_{i=1}^k) > \frac{2\alpha k^2}{\Psi}.$$

We note that Lemma 3.2 follows directly by applying Lemma 2.4 with $\varepsilon = 64\alpha \cdot k^3/\Psi$. Substituting these bounds into (2) yields a contradiction, since

$$\frac{2\alpha k^2}{\Psi} < \mathrm{Cost}(\{A_i, c_i\}_{i=1}^k) \leqslant \alpha \cdot \triangle_k(\mathcal{X}_V) \leqslant \alpha \cdot \mathrm{Cost}(\{P_i, p^{(i)}\}_{i=1}^k) \leqslant \left(1 + \frac{3k}{\Psi}\right) \cdot \frac{\alpha k^2}{\Psi}.$$

Therefore, there exists a permutation π (the identity after suitable renumbering of one of the partitions) such that $\mu(A_i \triangle P_i) < \frac{8\alpha\delta}{10^4 k} \cdot \mu(P_i)$ for all $i \in [1:k]$.

Part $(a.2)$ follows from part $(a.1)$. Indeed, for $\delta' = 8\delta/10^4$ we have

$$\mu(A_i) \geqslant \mu(P_i \cap A_i) = \mu(P_i) - \mu(P_i \setminus A_i) \geqslant \mu(P_i) - \mu(A_i \triangle P_i) \geqslant \left(1 - \frac{\alpha\delta'}{k}\right) \cdot \mu(P_i)$$

and $|E(A_i, \overline{A_i})| \leqslant |E(P_i, \overline{P_i})| + \mu(A_i \Delta P_i)$ since every edge that is counted in $|E(A_i, \overline{A_i})|$ but not in $|E(P_i, \overline{P_i})|$ must have an endpoint in $A_i \Delta P_i$. Thus

$$\Phi(A_i) = \frac{|E(A_i, \overline{A_i})|}{\mu(A_i)} \leqslant \frac{|E(P_i, \overline{P_i})| + \frac{\alpha\delta'}{k} \cdot \mu(P_i)}{(1 - \frac{\alpha \cdot \delta'}{k}) \cdot \mu(P_i)} \leqslant \left(1 + \frac{2\alpha\delta'}{k}\right) \cdot \phi(P_i) + \frac{2\alpha\delta'}{k}.$$

This completes the proof of Theorem 1.2 (a).

4 The Normalized Spectral Embedding is ε-separated

In this section, we prove that the normalized spectral embedding $\mathcal{X}_V$ is ε-separated.

Proof of Theorem 2.6

We establish first a lower bound on $\triangle_{k-1}(\mathcal{X}_V)$.

▶ **Lemma 4.1.** *Let G be a graph that satisfies the gap assumption $\delta = 20^4 \cdot k^3/\Psi \in (0, 1/2]$. Then for $\delta' = 2\delta/20^4$ it holds $\triangle_{k-1}(\mathcal{X}_V) \geqslant 1/12 - \delta'/k$.*

Before we prove Lemma 4.1 we show that it implies (10). By Lemma 3.1 we have $\triangle_k(\mathcal{X}_V) \leqslant 2k^2/\Psi = \delta'/k$. Then, we apply Lemma 4.1 with $\delta \leqslant k \cdot \varepsilon/600$, where $\varepsilon = 6/10^7$ is the Ostrovsky et al.'s constant, yielding

$$\triangle_{k-1}(\mathcal{X}_V) \geqslant \frac{1}{12} - \frac{\delta'}{k} = \frac{1}{12} - \frac{2}{20^4}\cdot\frac{\delta}{k} \geqslant \frac{10^{10}}{9\cdot 2^5}\cdot\frac{\delta}{k} = \frac{1}{\varepsilon^2}\cdot\frac{\delta'}{k} \geqslant \frac{1}{\varepsilon^2}\cdot\triangle_k(\mathcal{X}_V).$$

Proof of Lemma 4.1

Let $(P_1,\ldots,P_k)$ and $(Z_1,\ldots,Z_{k-1})$ be partitions of V. We define a mapping $\sigma : [1:k-1] \mapsto [1:k]$ by

$$\sigma(i) = \arg\max_{j\in[1:k]} \frac{\mu(Z_i\cap P_j)}{\mu(P_j)}, \quad \text{for every } i \in [1:k-1].$$

We lower bound now the clusters overlapping in terms of the volume between any k-way and $(k-1)$-way partitions of V.

▶ **Lemma 4.2.** *Suppose $(P_1,\ldots,P_k)$ and $(Z_1,\ldots,Z_{k-1})$ are partitions of V. Then for any index $\ell \in [1:k]\setminus\{\sigma(1),\ldots,\sigma(k-1)\}$ (there is at least one such ℓ) and for every $i \in [1:k-1]$ it holds*

$$\left\{\mu(Z_i\cap P_{\sigma(i)}), \mu(Z_i\cap P_\ell)\right\} \geqslant \tau_i\cdot\min\left\{\mu(P_\ell), \mu(P_{\sigma(i)})\right\},$$

where $\sum_{i=1}^{k-1}\tau_i = 1$ and $\tau_i \geqslant 0$.

Proof. By pigeonhole principle there is an index $\ell \in [1:k]$ such that $\ell \notin \{\sigma(1),\ldots,\sigma(k-1)\}$. Thus, for every $i \in [1:k-1]$ we have $\sigma(i) \neq \ell$ and

$$\frac{\mu(Z_i\cap P_{\sigma(i)})}{\mu(P_{\sigma(i)})} \geqslant \frac{\mu(Z_i\cap P_\ell)}{\mu(P_\ell)} \triangleq \tau_i,$$

where $\sum_{i=1}^{k-1}\tau_i = 1$ and $\tau_i \geqslant 0$ for all i. Hence, the statement follows. ◀

Proof of Lemma 4.1. Let $(Z_1,\ldots,Z_{k-1})$ be a $(k-1)$-way partition of V with centers $c'_1,\ldots,c'_{k-1}$ that achieves $\triangle_{k-1}(\mathcal{X}_V)$, and $(P_1,\ldots,P_k)$ be a k-way partition of V achieving $\widehat{\rho}_{\mathrm{avr}}(k)$. Our goal now is to lower bound the optimal $(k-1)$-means cost

$$\triangle_{k-1}(\mathcal{X}_V) = \sum_{i=1}^{k-1}\sum_{j=1}^{k}\sum_{u\in Z_i\cap P_j} d_u \left\|F(u)-c'_i\right\|^2. \tag{15}$$

By Lemma 4.2 there is an index $\ell \in [1:k]\setminus\{\sigma(1),\ldots,\sigma(k-1)\}$. For $i \in [1:k-1]$ let

$$p^{\gamma(i)} = \begin{cases} p^\ell & \text{, if } \left\|p^\ell - c'_i\right\| \geqslant \left\|p^{\sigma(i)} - c'_i\right\|; \\ p^{\sigma(i)} & \text{, otherwise.}\end{cases}$$

Then by combining Lemma 2.3 and Lemma 4.2, we have

$$\left\|p^{\gamma(i)} - c'_i\right\|^2 \geqslant \left[8\cdot\min\left\{\mu(P_\ell),\mu(P_{\sigma(i)})\right\}\right]^{-1} \text{ and } \mu(Z_i\cap P_{\gamma(i)}) \geqslant \tau_i\cdot\min\left\{\mu(P_\ell),\mu(P_{\sigma(i)})\right\}, \tag{16}$$

where $\sum_{i=1}^{k-1}\tau_i = 1$. We now lower bound the expression in (15). Since

$$\left\|F(u)-c'_i\right\|^2 \geqslant \frac{1}{2}\left\|p^{\gamma(i)}-c'_i\right\|^2 - \left\|F(u)-p^{\gamma(i)}\right\|^2,$$

it follows for $\delta' = 2\delta/20^4$ that

$$\begin{aligned}\triangle_{k-1}(\mathcal{X}_V) &= \sum_{i=1}^{k-1}\sum_{j=1}^{k}\sum_{u\in Z_i\cap P_j} d_u \left\|F(u)-c_i'\right\|^2 \geqslant \sum_{i=1}^{k-1}\sum_{u\in Z_i\cap P_{\gamma(i)}} d_u \left\|F(u)-c_i'\right\|^2 \\ &\geqslant \frac{1}{2}\sum_{i=1}^{k-1}\sum_{u\in Z_i\cap P_{\gamma(i)}} d_u \left\|p^{\gamma(i)}-c_i'\right\|^2 - \sum_{i=1}^{k-1}\sum_{u\in Z_i\cap P_{\gamma(i)}} d_u \left\|F(u)-p^{\gamma(i)}\right\|^2 \\ &\geqslant \frac{1}{2}\sum_{i=1}^{k-1}\frac{\mu(Z_i\cap P_{\gamma(i)})}{8\cdot\min\left\{\mu(P_{\gamma(i)}),\mu(P_{\sigma(i)})\right\}} - \sum_{i=1}^{k}\sum_{u\in P_i} d_u \left\|F(u)-p^i\right\|^2 \\ &\geqslant \frac{1}{16}-\frac{\delta'}{k},\end{aligned}$$

where the last inequality holds due to (16) and Lemma 3.1. ◀

5 An Efficient Spectral Clustering Algorithm

Here, we apply the proof techniques developed by Boutsidis et al. [1, 2] to our setting. More precisely, we prove that any α-approximate k-means algorithm that runs on an approximate normalized spectral embedding $\widetilde{Y'}$ computed by the power method, yields an approximate clustering $\widetilde{X'_\alpha}$ of the normalized spectral embedding Y'.

Furthermore, we prove under our gap assumption that $\widetilde{Y'}$ is ε-separated. This allows us to apply the variant of Lloyd's k-means algorithm analyzed by Ostrovsky et al. [5] to efficiently compute $\widetilde{X'_\alpha}$. Then we use Theorem 1.2 (a) to establish the desired statement.

This section is organized as follows. In Subsection 5.1, we prove Theorem 2.8. Then in Subsection 5.2, we present the proof of Theorem 2.9. Based on the results from the preceding two subsections, we prove Theorem 1.2 (b) in Subsection 5.3.

5.1 Proof of Theorem 2.8

Due to space limits, we defer the proofs of the next Lemmas to the full version of the paper.

▶ **Lemma 5.1.** *$X'X'^{\mathrm{T}}$ is a projection matrix.*

▶ **Lemma 5.2.** *It holds that $Y'^{\mathrm{T}}Y' = I_{k\times k} = \widetilde{Y'}^{\mathrm{T}}\widetilde{Y'}$.*

▶ **Lemma 5.3.** *It holds that $\left\|Y'Y'^{\mathrm{T}} - \widetilde{Y'}\widetilde{Y'}^{\mathrm{T}}\right\|_F = \left\|YY^{\mathrm{T}} - \widetilde{Y}\widetilde{Y}^{\mathrm{T}}\right\|_F$.*

▶ **Lemma 5.4.** *For any matrix U with orthonormal columns and every matrix A it holds*

$$\left\|UU^{\mathrm{T}} - AA^{\mathrm{T}}UU^{\mathrm{T}}\right\|_F = \left\|U - AA^{\mathrm{T}}U\right\|_F. \tag{17}$$

Proof Sketch of Theorem 2.8. By combining Lemma 2.7 and Lemma 5.3, with probability at least $1-2e^{-2n}-3\delta_p$ we have $\left\|Y'Y'^{\mathrm{T}} - \widetilde{Y'}\widetilde{Y'}^{\mathrm{T}}\right\|_F = \left\|YY^{\mathrm{T}} - \widetilde{Y}\widetilde{Y}^{\mathrm{T}}\right\|_F \leqslant \varepsilon$.

Let $Y'Y'^{\mathrm{T}} = \widetilde{Y'}\widetilde{Y'}^{\mathrm{T}} + E$ such that $\|E\|_F \leqslant \varepsilon$. Based on Lemma 5.2 and Lemma 5.4 we have that (17) holds for the matrices Y' and $\widetilde{Y'}$. Hence, by Lemma 5.1 we can apply the proof in [1, Theorem 6] to obtain $\left\|Y' - \widetilde{X'_\alpha}\left(\widetilde{X'_\alpha}\right)^{\mathrm{T}}Y'\right\|_F \leqslant \sqrt{\alpha}\cdot\left(\left\|Y' - X'_{\mathrm{opt}}\left(X'_{\mathrm{opt}}\right)^{\mathrm{T}}Y'\right\|_F + 2\varepsilon\right)$. The desired statement follows by simple algebraic manipulations. ◀

5.2 Proof of Theorem 2.9

In this subsection, we show under our gap assumption that the approximate normalized spectral embedding $\widetilde{Y'}$ is ε-separated, i.e. $\triangle_k(\widetilde{\mathcal{X}_V}) < 5\varepsilon^2 \cdot \triangle_{k-1}(\widetilde{\mathcal{X}_V})$. Our analysis builds upon Theorem 2.6, Theorem 2.8 and the proof techniques in [1, Theorem 6].

We use interchangeably X'_{opt} and $X'^{(k)}_{\text{opt}}$ to denote the optimal indicator matrix for the k-means problem on $\mathcal{X}_V$ that is induced by the rows of matrix Y'. Similarly, we denote by $X'^{(k-1)}_{\text{opt}}$ the optimal indicator matrix for the $(k-1)$-means problem on $\mathcal{X}_V$.

Proof Sketch of Theorem 2.9. By Theorem 2.6 we have

$$\left\| Y' - X'^{(k)}_{\text{opt}} \left(X'^{(k)}_{\text{opt}} \right)^{\mathrm{T}} Y' \right\|_F \leqslant \varepsilon \left\| Y' - X'^{(k-1)}_{\text{opt}} \left(X'^{(k-1)}_{\text{opt}} \right)^{\mathrm{T}} Y' \right\|_F . \tag{18}$$

We set the approximation parameter in Theorem 2.8 to

$$\varepsilon' \triangleq \frac{1}{4}\sqrt{\triangle_k(\mathcal{X}_V)} = \frac{1}{4} \left\| Y' - X'^{(k)}_{\text{opt}} \left(X'^{(k)}_{\text{opt}} \right)^{\mathrm{T}} Y' \right\|_F \geqslant n^{-O(1)}, \tag{19}$$

and we note that by Theorem 2.6 it holds $\varepsilon' \leqslant \frac{\varepsilon}{4}\sqrt{\triangle_{k-1}(\mathcal{X}_V)}$.

We construct now matrix $\widetilde{Y}$ via the power method with $p \geqslant \Omega(\frac{\ln n}{\lambda_{k+1}})$. By Lemma 5.3 we have $\left\| Y'Y'^{\mathrm{T}} - \widetilde{Y'}\widetilde{Y'}^{\mathrm{T}} \right\|_F = \left\| YY^{\mathrm{T}} - \widetilde{Y}\widetilde{Y}^{\mathrm{T}} \right\|_F$ and thus by Lemma 2.7 with high probability it holds that $\left\| Y'(Y')^{\mathrm{T}} - \widetilde{Y'}\widetilde{Y'}^{\mathrm{T}} \right\|_F \leqslant \varepsilon'$.

Let $Y'(Y')^{\mathrm{T}} = \widetilde{Y'}\widetilde{Y'}^{\mathrm{T}} + E$ such that $\|E\|_F \leqslant \varepsilon'$. By combining Lemma 5.1, Lemma 5.2, Lemma 5.3 and by applying the proof techniques in [1, Theorem 6] we obtain

$$\sqrt{\triangle_k\left(\widetilde{\mathcal{X}_V}\right)} \leqslant \|E\|_F + \left\| Y' - \widetilde{X'^{(k)}_{\text{opt}}} \left(\widetilde{X'^{(k)}_{\text{opt}}} \right)^{\mathrm{T}} Y' \right\|_F .$$

Furthermore, we can show that $\ln\left(\frac{2-\lambda_k}{2-\lambda_{k+1}}\right) \geqslant \frac{1}{2}\left(1 - \frac{4\delta}{20^4 k^2}\right)\lambda_{k+1}$. Then Theorem 2.8 yields

$$\left\| Y' - \widetilde{X'^{(k)}_{\text{opt}}} \left(\widetilde{X'^{(k)}_{\text{opt}}} \right)^{\mathrm{T}} Y' \right\|_F^2 \leqslant (1+4\varepsilon') \cdot \left\| Y' - X'^{(k)}_{\text{opt}} \left(X'^{(k)}_{\text{opt}} \right)^{\mathrm{T}} Y' \right\|_F^2 + 4\varepsilon'^2.$$

Also, we can show $\left\| Y' - X'^{(k)}_{\text{opt}} \left(X'^{(k)}_{\text{opt}} \right)^{\mathrm{T}} Y' \right\|_F^2 \leqslant \frac{1}{8 \cdot 10^{13}}$ and by the definition of ε' it follows

$$\sqrt{\triangle_k\left(\widetilde{\mathcal{X}_V}\right)} \leqslant 2\sqrt{\triangle_k(\mathcal{X}_V)} \leqslant 2\varepsilon \cdot \sqrt{\triangle_{k-1}(\mathcal{X}_V)}. \tag{20}$$

Moreover, by applying similar arguments as in the proof of [1, Theorem 6] we can prove that

$$\sqrt{\triangle_{k-1}(\mathcal{X}_V)} \leqslant \left(1 + \frac{\varepsilon}{2}\right) \sqrt{\triangle_{k-1}\left(\widetilde{\mathcal{X}_V}\right)}. \tag{21}$$

The statement follows by combining (20) and (21). ◀

5.3 Proof of Part (b) of Theorem 1.2

Let $p = \Theta(\frac{\ln n}{\lambda_{k+1}})$. We can compute the matrix $\mathcal{B}^p S$ in time $O(mkp)$ and its singular value decomposition $\widetilde{U}\widetilde{\Sigma}\widetilde{V}^{\mathrm{T}}$ in time $O(nk^2)$. Based on it, we construct in time $O(mk)$ matrix $\widetilde{Y'}$ (c.f. (13)).

By Theorem 2.9, $\widetilde{\mathcal{X}_V}$ is ε-separated for $\varepsilon = 6/10^7$, i.e. $\triangle_k\left(\widetilde{\mathcal{X}_V}\right) < 5\varepsilon^2 \cdot \triangle_{k-1}\left(\widetilde{\mathcal{X}_V}\right)$. Hence, by Theorem 2.5 there is an algorithm that outputs in time $O(mk^2 + k^4)$ a clustering with indicator matrix $\widetilde{X'_\alpha}$ satisfying

$$\left\|\widetilde{Y'} - \widetilde{X'_\alpha}\left(\widetilde{X'_\alpha}\right)^{\mathrm{T}}\widetilde{Y'}\right\|_F^2 \leqslant \left(1 + \frac{1}{10^{10}}\right) \cdot \left\|\widetilde{Y'} - \widetilde{X'_{\mathrm{opt}}}\left(\widetilde{X'_{\mathrm{opt}}}\right)^{\mathrm{T}}\widetilde{Y'}\right\|_F^2$$

with constant probability (close to 1), where $\alpha = 1 + 1/10^{10}$.

Moreover, by applying Theorem 2.8 with $\varepsilon' = \frac{1}{4 \cdot 10^3}\left\|Y' - X'_{\mathrm{opt}}\left(X'_{\mathrm{opt}}\right)^{\mathrm{T}} Y'\right\|_F$ we can prove that the indicator matrix $\widetilde{X'_\alpha}$ yields a multiplicative approximation of $\mathcal{X}_V$, i.e.

$$\left\|Y' - \widetilde{X'_\alpha}\left(\widetilde{X'_\alpha}\right)^{\mathrm{T}} Y'\right\|_F^2 \leqslant \left(1 + \frac{1}{10^6}\right)\left\|Y' - X'_{\mathrm{opt}}\left(X'_{\mathrm{opt}}\right)^{\mathrm{T}} Y'\right\|_F^2. \tag{22}$$

The statement follows by Theorem 1.2 (a) applied to the partition $(A_1, \ldots, A_k)$ of V that is induced by the indicator matrix $\widetilde{X'_\alpha}$.

Acknowledgement. We would like to thank the anonymous reviewers for their constructive comments.

References

1 Christos Boutsidis, Prabhanjan Kambadur, and Alex Gittens. Spectral clustering via the power method - provably. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 40–48, 2015.

2 Christos Boutsidis and Malik Magdon-Ismail. Faster svd-truncated regularized least-squares. In *2014 IEEE International Symposium on Information Theory, Honolulu, HI, USA, June 29 - July 4, 2014*, pages 1321–1325, 2014.

3 James R. Lee, Shayan Oveis Gharan, and Luca Trevisan. Multi-way spectral partitioning and higher-order Cheeger inequalities. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, STOC'12, pages 1117–1130, New York, NY, USA, 2012. ACM.

4 Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems 14*, pages 849–856. MIT Press, 2002.

5 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, January 2013.

6 Shayan Oveis Gharan and Luca Trevisan. Partitioning into expanders. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1256–1266, 2014.

7 Richard Peng, He Sun, and Luca Zanetti. Partitioning well-clustered graphs: Spectral clustering works! In *Proceedings of The 28th Conference on Learning Theory, COLT 2015, Paris, France, July 3-6, 2015*, pages 1423–1455, 2015.

8 Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

9 Ali Kemal Sinop. How to round subspaces: A new spectral clustering algorithm. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1832–1847, 2016.

10 Ulrike von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, pages 395–416, 2007.

On the Fine-Grained Complexity of Rainbow Coloring[*]

Łukasz Kowalik[1], Juho Lauri[2], and Arkadiusz Socała[3]

1 **University of Warsaw, Warsaw, Poland**
kowalik@mimuw.edu.pl
2 **Tampere University of Technology, Tampere, Finland**
juho.lauri@tut.fi
3 **University of Warsaw, Warsaw, Poland**
a.socala@mimuw.edu.pl

Abstract

The Rainbow k-Coloring problem asks whether the edges of a given graph can be colored in k colors so that every pair of vertices is connected by a rainbow path, i.e., a path with all edges of different colors. Our main result states that for any $k \geq 2$, there is no algorithm for Rainbow k-Coloring running in time $2^{o(n^{3/2})}$, unless ETH fails. Motivated by this negative result we consider two parameterized variants of the problem. In the Subset Rainbow k-Coloring problem, introduced by Chakraborty *et al.* [STACS 2009, J. Comb. Opt. 2009], we are additionally given a set S of pairs of vertices and we ask if there is a coloring in which all the pairs in S are connected by rainbow paths. We show that Subset Rainbow k-Coloring is FPT when parameterized by $|S|$. We also study Maximum Rainbow k-Coloring problem, where we are additionally given an integer q and we ask if there is a coloring in which at least q anti-edges are connected by rainbow paths. We show that the problem is FPT when parameterized by q and has a kernel of size $O(q)$ for every $k \geq 2$, extending the result of Ananth *et al.* [FSTTCS 2011]. We believe that our techniques used for the lower bounds may shed some light on the complexity of the classical Edge Coloring problem, where it is a major open question if a $2^{O(n)}$-time algorithm exists.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases graph coloring, computational complexity, lower bounds, exponential time hypothesis, FPT algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.58

1 Introduction

The Rainbow k-Coloring problem asks whether the edges of a given graph can be colored in k colors so that every pair of vertices is connected by a rainbow path, i.e., a path with all edges of different colors. A minimum such k, called the *rainbow connection number* can be viewed as yet another measure of graph connectivity. The concept of rainbow coloring was introduced by Chartrand, Johns, McKeon, and Zhang [7] in 2008, while also featured in an earlier book of Chartrand and Zhang [8]. Chakraborty, Fischer, Matsliah, and Yuster [3] describe an interesting application of rainbow coloring in telecommunications. The problem

[*] Work partially supported by the National Science Centre of Poland, grant number 2013/09/B/ST6/03136 (Ł.K., A.S.), and by the Emil Aaltonen Foundation (J.L.). This is an extended abstract; for the full version see the technical report at arxiv [17].

is intensively studied from the combinatorial perspective, with over 100 papers published by now (see the survey of Li, Shi, and Sun [20] for an overview). However, the computational complexity of the problem seems less explored. It was conjectured by Caro, Lev, Roditty, Tuza, and Yuster [2] that the Rainbow k-Coloring problem is NP-complete for $k = 2$. This conjecture was confirmed by Chakraborty *et al.* [3]. Ananth, Nasre, and Sarpatwar [1] noticed that the proof of Chakraborty *et al.* in fact proves NP-completeness for every even $k > 1$, and complemented this by showing NP-completeness of the odd cases as well. An alternative hardness proof for every $k > 1$ was provided by Le and Tuza [19]. For complexity results on restricted graph classes, see e.g., [4, 5, 6, 12].

For many NP-complete graph problems there are algorithms running in time $2^{O(n)}$ for an n-vertex graph. This is obviously the case for problems asking for a set of vertices, like Clique or Vertex Cover, or more generally, for problems which admit polynomially (or even subexponentially) checkable $O(n)$-bit certificates. However, there are $2^{O(n)}$-time algorithms also for some problems for which such certificates are not known, including e.g., Hamiltonicity [13] and Vertex Coloring [18]. Unfortunately it seems that the best known worst-case running time bound for Rainbow k-Coloring is $k^m 2^k n^{O(1)}$, where m is the number of edges, which is obtained by checking each of the k^m colorings by a simple $2^k n^{O(1)}$-time dynamic programming algorithm [23]. Even in the simplest variant of just two colors, i.e., $k = 2$, this algorithm takes $2^{O(n^2)}$ time if the input graph is dense. It raises a natural question: is this problem really much harder than, say, Hamiltonicity, or have we just not found the right approach yet? Questions of this kind have received considerable attention recently. For example, the existence of a $2^{O(n)}$-time algorithm for Edge Coloring is a notorious question, appearing in numerous open problem lists. On the other hand, it was shown that unless the Exponential Time Hypothesis fails, there is no algorithm running in time $2^{o(n \log n)}$ for Channel Assignment [21], Subgraph Homomorphism, and Subgraph Isomorphism [9]. Let us recall the precise statement of the Exponential Time Hypothesis (ETH).

▶ **Hypothesis 1** (Exponential Time Hypothesis [14]). *There exists a constant $c > 0$, such that there is no deterministic algorithm solving* 3-SAT *in time $O^*(2^{cn})$.*

Note that some kind of a complexity assumption, like ETH, is hard to avoid when we prove exponential lower bounds, unless one aims at proving P $\neq$ NP.

Main Result. Our main result is the following theorem.

▶ **Theorem 2.** *For any $k \geq 2$,* Rainbow k-Coloring *can be solved neither in $2^{o(n^{3/2})}$ nor $2^{o(m/\log m)}$ time where n and m are the number of vertices and edges respectively, unless ETH fails.*

Hence, this is an NP-complete graph problem which does not admit a $2^{o(n^{1+\epsilon})}$-time algorithm (under reasonable complexity assumptions), for an $\epsilon > 0$. Such lower bounds are fairly rare in the literature. The best known algorithm for Rainbow k-Coloring just verifies all possible colorings and thus it runs in time $2^{O(m)}$ for any fixed k. Our lower bounds mean that one cannot hope for substantial improvements in this running time.

Remaining Lower Bounds. We also study a natural generalized problem, called Subset Rainbow k-Coloring, introduced by Chakraborty *et al.* [3] as a natural intermediate step in reductions from 3-SAT to Rainbow k-Coloring. In Subset Rainbow k-Coloring, we are given a connected graph G, and a set of pairs of vertices $S \subseteq \binom{V(G)}{2}$. Elements of S

are called *requests*. For a given coloring of $E(G)$ we say that a request $\{u, v\}$ is *satisfied* if u and v are connected by a rainbow path. The goal in Subset Rainbow k-Coloring is to determine whether there is a k-coloring of $E(G)$ such that every pair in S is satisfied. Our main result implies that Subset Rainbow k-Coloring admits no algorithm running in time $2^{o(n^{3/2})}$, under ETH. We show also two more lower bounds, as follows.

▶ **Theorem 3.** *For any $k \geq 2$,* Subset Rainbow k-Coloring *can be solved neither in time $2^{o(n^{3/2})}$, nor in time $2^{o(m)}$, nor in time $2^{o(s)}$ where n is the number of vertices, m is the number of edges, and s is the number of requests, unless ETH fails.*

An interesting feature here is that for $k = 2$ the $2^{o(m)}$ and $2^{o(s)}$ bounds are *tight* up to a polynomial factor (a $2^m n^{O(1)}$ algorithm is immediate, and a $2^{|S|}n^{O(1)}$-time algorithm is discussed in the next paragraph).

New Algorithms. In the context of the hardness results mentioned above it is natural to ask for FPT algorithms for Subset Rainbow k-Coloring. We show that for every fixed k, Subset Rainbow k-Coloring parameterized by $|S|$ is FPT:

▶ **Theorem 4.** *For every integer k,* Subset Rainbow k-Coloring *is FPT and it has an algorithm running in time $|S|^{O(|S|)}n^{O(1)}$.*

For the 2 color case we are able to show a different, faster algorithm running in time $2^{|S|}n^{O(1)}$, which is tight up to a polynomial factor.

We also study the Maximum Rainbow k-Coloring problem, introduced by Ananth, Nasre, and Sarpatwar [1]. Intuitively, the idea is to parameterize the problem by the number of pairs to satisfy. However, all pairs of adjacent vertices are trivially satisfied by any edge-coloring. Hence, we parameterize by the number of anti-edges to satisfy. More formally, in Maximum Rainbow k-Coloring we are given a graph $G = (V, E)$, an integer q, and asked whether there is a coloring of E that satisfies at least q anti-edges. First, we show that the maximization version of the problem (find maximum such q) admits a constant factor approximation algorithm for every fixed value of k. Second, we show that Maximum Rainbow k-Coloring is FPT for every $k \geq 2$, which generalizes the result of Ananth *et al.* [1] who showed this claim for the $k = 2$ case. Our algorithm runs in time $2^{q \log q}n^{O(1)}$ for any k, which is faster than the algorithm of Ananth *et al.* for 2 colors. For 2 colors we give an even faster algorithm, running in time $8^q n^{O(1)}$. We also show that the problem admits a kernel size $O(q)$, i.e., that there is a polynomial-time algorithm that returns an equivalent instance with $O(q)$ vertices. (For more background on kernelization see e.g., [10].) Before, this was known only for $k = 2$ (due to Ananth *et al.* [1]). Our main results for Maximum Rainbow k-Coloring are summarized in the following theorem.

▶ **Theorem 5.** Maximum Rainbow k-Coloring *parameterized by the number of anti-edges q is FPT for every $k \geq 2$. Moreover, it admits a kernel of linear size.*

Notation. For standard graph-theoretic notions, we refer the reader to [11]. All graphs we consider in this paper are simple and undirected. We denote $\Delta_1(G) = \max\{\Delta(G), 1\}$.

By $\bar{E}$ we denote the set of anti-edges, i.e., $\bar{E} = \binom{V}{2} \setminus E$. When $G = (V, E)$ is a graph then $\bar{G} = (V, \bar{E})$ is its *complement graph.* By $x^{\underline{k}}$ we denote the falling factorial, i.e., $x^{\underline{k}} = x(x-1)\cdots(x-k+1)$. For an integer k, we denote $[k] = \{1, \ldots, k\}$. For a (partial) function c, by $\mathrm{Dom}(c)$ we denote its domain.

If I and J are instances of decision problems P and R, respectively, then we say that I and J are *equivalent*, when either both I and J are YES-instances or both are NO-instances.

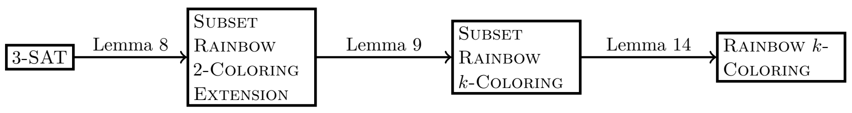

3-SAT
Lemma 8
SUBSET RAINBOW 2-COLORING EXTENSION
Lemma 9
SUBSET RAINBOW k-COLORING
Lemma 14
RAINBOW k-COLORING

Figure 1 A simplified road map of our reductions.

Organization of the paper. In Section 2 we present our hardness results. The main difficulties we encountered are sketched at the beginning of that section. Due to space constraints proofs of the claims marked by ★ are skipped and can be found in the full version [17]. Next, in Section 3 we present our algorithms for SUBSET RAINBOW k-COLORING. Again because of the space limitations, our algorithms for MAXIMUM RAINBOW k-COLORING are skipped in this extended abstract and are available in the full version [17].

2 Hardness of rainbow coloring

2.1 Overview

The main goal of this section is to show that for any $k \geq 2$ RAINBOW k-COLORING does not admit an algorithm running in time $2^{o(n^{3/2})}$, unless the Exponential Time Hypothesis fails. Let us give a high-level overview of our proof. A natural idea would be to begin with a 3-SAT formula ϕ with n variables and then transform it in time $2^{o(n)}$ to an equivalent instance $G = (V, E)$ of RAINBOW k-COLORING with $O(n^{2/3})$ vertices. Then indeed a $2^{o(|V|^{3/2})}$-time algorithm that solves RAINBOW 2-COLORING can be used to decide 3-SAT in time $2^{o(n)}$. Note that in a typical NP-hardness reduction, we observe some polynomial blow-up of the instance size. For example, one can verify that in the reduction of Chakraborty *et al.* [3], the initial 3-SAT formula with n variables and m clauses is transformed into a graph with $\Theta(n^4 + m^4)$ vertices and edges. In our case, instead of a blow-up we aim at *compression*: the number of vertices needs to be much smaller than the number of variables in the input formula ϕ. As usual in reductions, variables and clauses in ϕ are going to correspond to some structures in G, called gadgets. The compression requirement means that our gadgets need to share vertices. To make our lives slightly easier, we apply the following well-known Sparsification Lemma, which allows for assuming that the number of clauses is $O(n)$.

▶ **Lemma 6** (Sparsification Lemma [15]). *For each $\varepsilon > 0$ there exist a constants c_ε, such that any* 3-SAT *formula φ with n variables can be expressed as $\varphi = \vee_{i=1}^{t} \psi_i$, where $t \leq 2^{\varepsilon n}$ and each ψ_i is a* 3-SAT *formula with the same variable set as φ, but contains at most $c_\varepsilon n$ clauses. Moreover, this disjunction can be computed in time $O^*(2^{\varepsilon n})$.*

Note that by using the Sparsification Lemma we tweak our general plan a bit: instead of creating one equivalent instance, we are going to create $2^{\varepsilon n}$ instances (for arbitrarily small ϵ), each with $O(n^{2/3})$ vertices. The following lemma further simplifies the instance.

▶ **Lemma 7** ([22]). *Given a* 3-SAT *formula φ with m clauses one can transform it in polynomial time into a formula φ' with $O(m)$ variables and $O(m)$ clauses, such that φ' is satisfiable iff φ' is satisfiable, and moreover each clause of φ' contains exactly three different variables and each variable occurs in at most 4 clauses of φ'.*

Now our goal is to transform a 3-SAT formula ϕ with n variables such that every variable occurs in at most 4 clauses, to a graph with $O(n^{2/3})$ vertices — an equivalent instance of RAINBOW k-COLORING. We do it in three steps (see Fig 1).

In the first step we transform ϕ to an instance $I = (G, S, c_0)$ of Subset Rainbow 2-Coloring Extension, which is a generalization of Subset Rainbow 2-Coloring, where c_0, called a *precoloring*, is a partial coloring of the edges of G into two colors and the goal is to determine if there is an edge-coloring of $E(G)$ which extends c_0 and such that all pairs of S are satisfied. The first step is crucial, because here the compression takes place: $|V(G)| = O(n^{2/3})$ and $E(G) = O(n)$. The major challenge in the construction is avoiding interference between gadgets that share a vertex: to this end we define various conflict graphs and we show that they can be vertex-colored in a few colors. This reduction is described in Section 2.2.

In the second step (Lemma 9) we reduce Subset Rainbow 2-Coloring Extension to Subset Rainbow k-Coloring, for every $k \geq 2$. In fact, in the full version this is done in two sub-steps, via Subset Rainbow k-Coloring Extension. The number of the vertices in the resulting instance does not increase more than by a constant factor. This step is rather standard, though some technicalities appear because we need to guarantee additional properties of the output instance, which are needed by the reduction in the third step.

The last step (Section 2.3), where we reduce an instance $(G = (V, E), S)$ of Subset Rainbow k-Coloring to an instance G' of Rainbow k-Coloring, is yet another challenge. We would like to get rid of the set of requests somehow. For simplicity, let us focus on the $k = 2$ case now. Here, the natural idea, used actually by Chakraborty *et al.* [3] is to create, for every $\{u, v\} \notin S$, a path (u, x_{uv}, v) through a new vertex x_{uv}. Such a path cannot help any of the requests $\{u', v'\} \in S$ to get satisfied (since if it creates a new path P' between u' and v', then P' has length at least 3), and by coloring it into two different colors we can satisfy $\{u, v\}$. Unfortunately, in our case we cannot afford for creating a new vertex for every such $\{u, v\}$, because that would result in a quadratic blow up in the number of vertices. However, one can observe that for any biclique (a complete bipartite subgraph) in the graph $(V, \binom{V}{2} \setminus S)$ it is sufficient to use just one such vertex x (connected to all the vertices of the biclique). By applying a result of Jukna [16] we can show that in our specific instance of Subset Rainbow 2-Coloring which results from a 3-SAT formula, the number of bicliques needed to cover all the pairs in $\binom{V}{2} \setminus S$ is small enough. We show a $2^{|V(G)|}|V(G)|^{O(1)}$-time algorithm to find such a cover. Although this algorithm does not seem fast, in our case $|V(G)| = O(n^{2/3})$, so this complexity is *subexponential* in the number of variables of the input formula, which is enough for our goal. The case of $k \geq 3$ is similar, i.e., we also use the biclique cover. However, the details are much more technical because for each biclique we need to introduce a much more complex gadget.

2.2 From 3-SAT to Subset Rainbow k-Coloring

Let Subset Rainbow k-Coloring Extension be a generalization of Subset Rainbow k-Coloring, where c_0 is a partial k-coloring of the edges of G and the goal is to determine if there is an edge-coloring of $E(G)$ which extends c_0 and such that all pairs of S are satisfied. In this section we show a reduction (Lemma 8) from 3-SAT to Subset Rainbow 2-Coloring Extension.

For an instance $I = (G, S, c_0)$ of Subset Rainbow k-Coloring Extension (for any $k \geq 2$), let us define a *precoloring conflict graph* CG_I. Its vertex set is the set of colored edges, i.e., $V(CG_I) = \mathrm{Dom}(c_0)$. Two different colored edges e_1 and e_2 (treated as vertices of CG_I) are adjacent in CG_I when they are incident in G or there is a pair of endpoints $u \in e_1$ and $v \in e_2$ such that $uv \in E(G) \cup S$.

In what follows the reduction in Lemma 8 is going to be pipelined with further reductions going through Subset Rainbow k-Coloring Extension and Subset Rainbow k-

Coloring to Rainbow k-Coloring. In these three reductions we need to keep the instance small. To this end, the instance of Subset Rainbow 2-Coloring Extension resulting in Lemma 8 has to satisfy some additional properties, which are formulated in the claim of Lemma 8. Their role will become clearer later on.

▶ **Lemma 8.** *Given a* 3-SAT *formula φ with n variables such that each clause of φ contains exactly three variables and each variable occurs in at most four clauses, one can construct in polynomial time an equivalent instance (G, S, c_0) of* Subset Rainbow 2-Coloring Extension *such that G has $O(n^{2/3})$ vertices and $O(n)$ edges. Moreover, $\Delta(G) = O(n^{1/3})$, $\Delta(V(G), S) = O(n^{1/3})$, $|\mathrm{Dom}(c_0)| = O(n^{2/3})$ and along with the instance $I = (G, S, c_0)$ the algorithm constructs a proper vertex 4-coloring of $(V(G), E \cup S)$ (so also of $(V(G), S)$) and a proper vertex $O(n^{1/3})$-coloring of the precoloring conflict graph CG_I.*

Proof. Let m denote the number of clauses in φ. Observe that $m \leq \frac{4}{3}n$. Let Var and Cl denote the sets of variables and clauses of φ. For more clarity, the two colors of the partial coloring c_0 will be called T and F. Let us describe the graph G along with a set of anti-edges S. Graph G consists of two disjoint vertex subsets: the variable part and the clause part. The intuition is that in any 2-edge coloring of G that extends c_0 and satisfies all pairs in S

- edge colors in the variable part represent an assignment of the variables of φ,
- edge colors in the clause part represent a choice of literals that satisfy all the clauses, and
- edge colors between the two parts make the values of the literals from the clause part consistent with the assignment represented by the variable part.

The variable part. The vertices of the variable part consist of the *middle set* M and $\lceil n^{1/3} \rceil$ layers $L_1 \cup L_2 \cdots \cup L_{\lceil n^{1/3} \rceil}$. The middle set M consists of vertices m_i for each $i = 1, \ldots, \lceil n^{2/3} \rceil + 9$. For every $i = 1, \ldots, \lceil n^{1/3} \rceil$ the layer L_i consists of two parts: upper $L_i^{\uparrow} = \{u_{i,j} \ : \ j = 1, \ldots, \lceil n^{1/3} \rceil + 3\}$ and lower $L_i^{\downarrow} = \{l_{i,j} \ : \ j = 1, \ldots, \lceil n^{1/3} \rceil + 3\}$.

We are going to define four functions: mid : Var $\to M$, lay, up, low : Var $\to [\lceil n^{1/3} \rceil]$. Then, for every variable $x \in$ Var we add two edges $u_{\mathrm{lay}(x),\mathrm{up}(x)}\mathrm{mid}(x)$ and $\mathrm{mid}(x)l_{\mathrm{lay}(x),\mathrm{low}(x)}$. Moreover, we add the pair $p_x = \{u_{\mathrm{lay}(x),\mathrm{up}(x)}, l_{\mathrm{lay}(x),\mathrm{low}(x)}\}$ to S. In other words, x corresponds to the 2-path $u_{\mathrm{lay}(x),\mathrm{up}(x)}\mathrm{mid}(x)l_{\mathrm{lay}(x),\mathrm{low}(x)}$. Now we describe a careful construction of the four functions, that guarantee several useful properties (for example edge-disjointness of paths corresponding to different variables).

Let us define the *variable conflict graph* $G_V = (\mathrm{Var}, E_{G_V})$, where for two variables $x, y \in$ Var we have xy are adjacent iff they both occur in the same clause. Since every variable occurs in at most 4 clauses, $\Delta(G_V) \leq 8$. It follows that there is a proper vertex 9-coloring $\alpha : Var \to [9]$ of G_v, and it can be found by a simple linear time algorithm. Next, each of the 9 color classes $\alpha^{-1}(i)$ is partitioned into $\lceil |\alpha^{-1}(i)| / \lceil n^{1/3} \rceil \rceil$ disjoint groups, each of size at most $\lceil n^{1/3} \rceil$. It follows that the total number n_g of groups is at most $\lceil n^{2/3} \rceil + 9$. Let us number the groups arbitrarily from 1 to n_g and for every variable $x \in$ Var, let $g(x)$ be the number of the group that contains x. Then we define $\mathrm{mid}(x) = m_{g(x)}$. Since any group contains only vertices of the same color we can state the following property:

($\mathbf{P_1}$) If variables x and y occur in the same clause then $\mathrm{mid}(x) \neq \mathrm{mid}(y)$.

Now, for every variable x we define its layer, i.e., the value of the function lay(x). Recall that for every $i = 1, \ldots, \lceil n^{2/3} \rceil + 9$ the i-th group $\mathrm{mid}^{-1}(m_i)$ contains at most $\lceil n^{1/3} \rceil$ variables. Inside each group, number the variables arbitrarily and let lay(x) be the number of variable x in its group, $\mathrm{lay}(x) \in [n^{1/3}]$. This implies another important property.

($\mathbf{P_2}$) If variables x and y belong to the same layer then $\mathrm{mid}(x) \neq \mathrm{mid}(y)$.

Observe that every layer gets assigned at most $\lceil n^{2/3} \rceil + 9$ variables. For every layer L_i pick any injective function $h_i : \mathrm{lay}^{-1}(i) \to [\lceil n^{1/3} \rceil + 3]^2$. Then, for every variable $x \in \mathrm{Var}$ we put $(\mathrm{up}(x), \mathrm{low}(x)) = h_{\mathrm{lay}(x)}(x)$. Note that by (P$_2$) we have the following.

(P$_3$) For every variable x there is exactly one 2-path in G connecting p_x, namely $(u_{\mathrm{lay}(x),\mathrm{up}(x)}, \mathrm{mid}(x), l_{\mathrm{lay}(x),\mathrm{low}(x)})$.

(P$_4$) For every pair of variables x, y the two unique paths connecting p_x and p_y are edge-disjoint.

Although we are going to add more edges and vertices to G, none of these edges has any endpoint in $\bigcup_i L_i$, so P_3 will stay satisfied.

The clause part. The vertices of the clause part are partitioned into $O(m^{1/3})$ clusters. Similarly as in the case of variables, each clause is going to correspond to a pair of vertices in the same cluster. Again, the assignment of clauses to clusters has to be done carefully. To this end we introduce the *clause conflict graph* $G_C = (\mathrm{Cl}, E_{G_C})$. Two different clauses C_1 and C_2 are adjacent in G_C if C_1 contains a variable x_1 and C_2 contains a variable x_2 such that $\mathrm{mid}(x_1) = \mathrm{mid}(x_2)$. Fix a variable x_1. Since $|\mathrm{mid}^{-1}(\mathrm{mid}(x_1))| \leq \lceil n^{1/3} \rceil$, there are at most $\lceil n^{1/3} \rceil$ variables x_2 such that $\mathrm{mid}(x_1) = \mathrm{mid}(x_2)$. Since every clause contains 3 variables, and each of them is in at most 4 clauses, $\Delta(G_C) \leq 12 \lceil n^{1/3} \rceil$. It follows that in polynomial time we can find a proper coloring β of the vertices of G_C into at most $12 \lceil n^{1/3} \rceil + 1$ colors. Moreover, if for any color j its color class $\beta^{-1}(j)$ is larger than $\lceil n^{2/3} \rceil$ we partition it into $\lceil |\beta^{-1}(j)| / \lceil n^{2/3} \rceil \rceil$ new colors. Clearly, in total we produce at most $\frac{4}{3} \lceil n^{1/3} \rceil$ new colors in this way because $m \leq \frac{4}{3} n \leq \frac{4}{3} \lceil n^{1/3} \rceil \cdot \lceil n^{2/3} \rceil$. Hence, in what follows we assume that each color class of β is of size at most $\lceil n^{2/3} \rceil$, and the total number of colors $s \leq 14 \lceil n^{1/3} \rceil + 1$. In what follows we construct s clusters $Q_1, \ldots, Q_s$. Every clause $C \in \mathrm{Cl}$ is going to correspond to a pair of vertices in the cluster $Q_{\beta(C)}$.

Fix $i = 1, \ldots, s$. Let us describe the subgraph induced by cluster Q_i. Define *cluster conflict graph* $G_i = (\beta^{-1}(i), E_{G_i})$. Two different clauses $C_1, C_2 \in \beta^{-1}(i)$ are adjacent in G_i if there are three variables x_1, x_2, and x_3 such that *(i)* C_1 contains x_1, *(ii)* C_2 contains x_2, *(iii)* $(\mathrm{lay}(x_1), \mathrm{up}(x_1)) = (\mathrm{lay}(x_3), \mathrm{up}(x_3))$ and *(iv)* $\mathrm{mid}(x_2) = \mathrm{mid}(x_3)$. Fix a variable x_1 which appears in a clause $C_1 \in \beta^{-1}(i)$. By our construction, there are at most $\lceil n^{1/3} \rceil + 2$ other variables x_3 that map to the same pair as x_1 by functions lay and up. For each such x_3 there are at most $\lceil n^{1/3} \rceil$ variables x_2 such that $\mathrm{mid}(x_2) = \mathrm{mid}(x_3)$; however, at most one of these variables belongs to a clause C_2 from the same cluster $\beta^{-1}(i)$, by the definition of the coloring β. It follows that $\Delta(G_i) \leq 12(\lceil n^{1/3} \rceil + 2)$. Hence in polynomial time we can find a proper coloring γ_i of the vertices of G_i into at most $12(\lceil n^{1/3} \rceil + 2) + 1$ colors. Similarly as in the case of the coloring β, we can assume that each of the color classes of γ_i has at most $\lceil n^{1/3} \rceil$ clauses, at the expense of at most $\lceil n^{1/3} \rceil$ additional colors. It follows that we can construct in polynomial time a function $g : \mathrm{Cl} \to [\lceil n^{1/3} \rceil]$ such that for every cluster $i = 1, \ldots, s$ and for every color class S of γ_i g is injective on S. Let $n_i \leq 13 \lceil n^{1/3} \rceil + 25$ be the number of colors used by γ_i. For notational convenience, let us define a function $\gamma : \mathrm{Cl} \to [\max_i n_i]$ such that for any clause C we have $\gamma(C) = \gamma_{\beta(C)}(C)$.

We are ready to define the vertices and edges of Q_i. It is a union of three disjoint vertex sets A_i, B_i, and C_i. We have $A_i = \{a_{i,j} \ : \ j = 1, \ldots, \lceil n^{1/3} \rceil\}$, $B_i = \{b^k_{i,j} \ : \ j = 1, \ldots, n_i, k = 1, 2, 3\}$, and $C_i = \{c_{i,j} \ : \ j = 1, \ldots, n_i\}$. For every $j = 1, \ldots, n_i$ and for every $k = 1, 2, 3$ we add edge $c_{i,j} b^k_{i,j}$ to G, and we color it by c_0 to color F. (These are the only edges pre-colored in the whole graph G.) For every clause $C \in \beta^{-1}(i)$ we do the following. For each $k = 1, 2, 3$, add the edge $(a_{i,g(C)}, b^k_{i,\gamma(C)})$ to G. Finally, add the pair $\{a_{i,g(C)}, c_{i,\gamma(C)}\}$ to S. Clearly, the following holds:

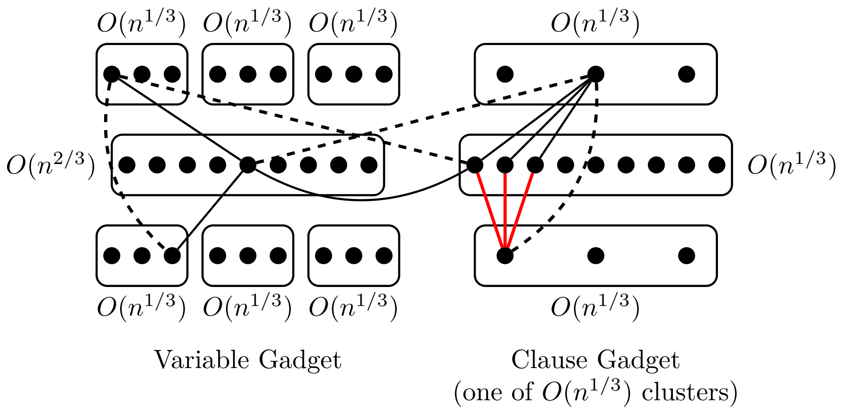


Figure 2 A simplified view of the obtained instance. Edges (solid lines) and requests (dashed lines) representing one variable and one clause that contains this variable are presented on the picture.

($\mathbf{P_5}$) Let C be any clause. Let $i = \beta(C)$ and let $j = g(C)$. Then there are exactly three 2-paths between $a_{\beta(C),g(C)}$ and $c_{\beta(C),\gamma(C)}$, each going through $b^k_{\beta(C),\gamma(C)}$ for $k = 1, 2, 3$.

Connections between the two parts. Consider a clause $C = \{\ell_1, \ell_2, \ell_3\}$ and its k-th literal ℓ_k for each $k = 1, 2, 3$. Then for some variable x we have $\ell_k = x$ or $\ell_k = \bar{x}$. We add the edge $b^k_{\beta(C),\gamma(C)}\mathrm{mid}(x)$ and we add the pair $\{\mathrm{mid}(x), a_{\beta(C),g(C)}\}$ to S. If $\ell_k = x$, we also add the pair $\{b^k_{\beta(C),\gamma(C)}, u_{\mathrm{lay}(x),\mathrm{up}(x)}\}$ to S; otherwise we add the pair $\{b^k_{\beta(C),\gamma(C)}, l_{\mathrm{lay}(x),\mathrm{low}(x)}\}$ to S. We claim the following.

($\mathbf{P_6}$) Every edge between the two parts was added exactly once, i.e., for every edge uv such that u is in the clause part and v is in the variable part, there is exactly one clause C and exactly one literal $\ell_k \in C$ such that $u = b^k_{\beta(C),\gamma(C)}$ and $v = \mathrm{mid}(x)$, where x is the variable in ℓ_k.

Indeed, assume for a contradiction that there is a clause C_1 with its k_1-th literal containing x_1 and a clause C_2 with its k_2-th literal containing x_2 such that $b^{k_1}_{\beta(C_1),\gamma(C_1)} = b^{k_2}_{\beta(C_2),\gamma(C_2)}$ and $\mathrm{mid}(x_1) = \mathrm{mid}(x_2)$. Then $C_1 \neq C_2$ by (P$_1$). Since $\mathrm{mid}(x_1) = \mathrm{mid}(x_2)$, C_1 and C_2 are adjacent in the clause conflict graph G_C. It follows that $\beta(C_1) \neq \beta(C_2)$, so two different clusters share a vertex, a contradiction.

This finishes the description of the instance (G, S, c_0). (See Fig. 2.)

From an assignment to a coloring. Let $\xi : \mathrm{Var} \to \{T, F\}$ be a satisfying assignment of φ. We claim that there is a coloring c of $E(G)$ which extends c_0 and satisfies all pairs in S. We define c as follows. Denote $\overline{F} = T$, $\overline{T} = F$ and $\xi(\overline{x}) = \overline{\xi(x)}$. For every variable $x \in \mathrm{Var}$ we put $c(u_{\mathrm{lay}(x),\mathrm{up}(x)}\mathrm{mid}(x)) = \xi(x)$ and $c(\mathrm{mid}(x)l_{\mathrm{lay}(x),\mathrm{low}(x)}) = \overline{\xi(x)}$. By (P$_3$) and (P$_4$) each edge is colored exactly once. Note that it satisfies all the pairs in S between vertices in the variable part.

For each clause C and each of its literals ℓ_k do the following. Let us color the edge $a_{\beta(C),g(C)}b^k_{\beta(C),\gamma(C)}$ with the color $\xi(\ell_k)$. Since g is injective on color classes of $\gamma_{\beta(C)}$, after processing all the literals in all the clauses, no edge is colored more than once. Recall that for every clause C we added exactly one pair to S, namely $\{a_{\beta(C),g(C)}, c_{\beta(C),\gamma(C)}\}$. Pick any of C's satisfied literals, say ℓ_k. Note that the pair $\{a_{\beta(C),g(C)}, c_{\beta(C),\gamma(C)}\}$ is then satisfied, because edge $a_{\beta(C),g(C)}b^k_{\beta(C),\gamma(C)}$ is colored by T and $b^k_{\beta(C),\gamma(C)}c_{\beta(C),\gamma(C)}$ is colored by F. Hence all the pairs in S between vertices in the clause part are satisfied.

Now let us color the edges between the clause part and the variable part. Consider any such edge uv, i.e., u is in the clause part and v is in the variable part. By (P$_6$), there is exactly one clause C and exactly one literal $\ell_k \in C$ such that $u = b^k_{\beta(C),\gamma(C)}$ and $v = \mathrm{mid}(x)$, where x is the variable in ℓ_k. Color the edge $b^k_{\beta(C),\gamma(C)}\mathrm{mid}(x)$ with the color $\overline{\xi(\ell_k)}$. Then the pair $\{\mathrm{mid}(x), a_{\beta(C),g(C)}\}$ is satisfied by the path $(\mathrm{mid}(x), b^k_{\beta(C),\gamma(C)}, a_{\beta(C),g(C)})$, since $c(b^k_{\beta(C),\gamma(C)}a_{\beta(C),g(C)}) = \xi(\ell_k)$. Assume $\ell_k = x$. Then the pair $\{b^k_{\beta(C),\gamma(C)}, u_{\mathrm{lay}(x),\mathrm{up}(x)}\}$ is satisfied by the path $(b^k_{\beta(C),\gamma(C)}, \mathrm{mid}(x), u_{\mathrm{lay}(x),\mathrm{up}(x)})$, since its first edge is colored by $\overline{\xi(\ell_k)} = \overline{\xi(x)}$ and its second edge is colored by $\xi(x)$. Analogously, when $\ell_k = \bar{x}$, then the pair $\{b^k_{\beta(C),\gamma(C)}, l_{\mathrm{lay}(x),\mathrm{low}(x)}\}$ is satisfied by the path $(b^k_{\beta(C),\gamma(C)}, \mathrm{mid}(x), l_{\mathrm{lay}(x),\mathrm{low}(x)})$, since its first edge is colored by $\overline{\xi(\ell_k)} = \xi(x)$ and its second edge is colored by $\overline{\xi(x)}$.

It follows that we colored all the edges and all the pairs in S are satisfied, so (G, S, c_0) is a YES-instance, as required.

From a coloring to an assignment. Let $c : E(G) \to \{T, F\}$ be a coloring which extends c_0 and satisfies all pairs in S. Consider the following variable assignment: for every $x \in \mathrm{Var}$, we put $\xi(x) = c(u_{\mathrm{lay}(x),\mathrm{up}(x)}\mathrm{mid}(x))$. We claim that ξ satisfies all the clauses of φ. Consider an arbitrary clause $C = \{\ell_1, \ell_2, \ell_3\}$.

Since the pair $\{a_{\beta(C),g(C)}, c_{\beta(C),\gamma(C)}\}$ is satisfied, there is a 2-color 2-path P between $a_{\beta(C),g(C)}$ and $c_{\beta(C),\gamma(C)}$. Recall that $N(c_{\beta(C),\gamma(C)}) = \{b^k_{\beta(C),\gamma(C)} : k = 1, 2, 3\}$, so there is $k = 1, 2, 3$ such that $b^k_{\beta(C),\gamma(C)}$ is the internal vertex on P. Since c extends c_0 and $c_0(b^k_{\beta(C),\gamma(C)}c_{\beta(C),\gamma(C)}) = F$, we infer that $c(a_{\beta(C),g(C)}b^k_{\beta(C),\gamma(C)}) = T$. Let x be the variable in the literal ℓ_k.

Since the pair $\{\mathrm{mid}(x), a_{\beta(C),g(C)}\}$ is satisfied, there is a 2-color 2-path Q between $\mathrm{mid}(x)$ and $a_{\beta(C),g(C)}$. Then the internal vertex of Q is $b^{k'}_{\beta(C'),\gamma(C')}$, for some clause C' and integer $k' = 1, 2, 3$. Let y be the variable in the k'-th literal of C'. Since there is an edge between $\mathrm{mid}(x)$ and $b^{k'}_{\beta(C'),\gamma(C')}$, from (P$_6$) we infer that $\mathrm{mid}(y) = \mathrm{mid}(x)$. If $C = C'$ and $k' \neq k$, then by (P$_1$) we get that $\mathrm{mid}(x) \neq \mathrm{mid}(y)$, a contradiction. If $C \neq C'$, since $\mathrm{mid}(y) = \mathrm{mid}(x)$, the clauses C and C' are adjacent in the clause conflict graph G_C, so $\beta(C') \neq \beta(C)$. However, then the edge $b^{k'}_{\beta(C'),\gamma(C')}a_{\beta(C),g(C)}$ of Q goes between two clusters, a contradiction. Hence $C' = C$ and $k' = k$, i.e., $Q = (\mathrm{mid}(x), b^k_{\beta(C),\gamma(C)}, a_{\beta(C),g(C)})$. Since $c(b^k_{\beta(C),\gamma(C)}a_{\beta(C),g(C)}) = T$, we get $c(\mathrm{mid}(x)b^k_{\beta(C),\gamma(C)}) = F$. Now assume w.l.o.g. that $\ell_k = x$, the case $\ell_k = \bar{x}$ is analogous.

Since the pair $\{b^k_{\beta(C),\gamma(C)}, u_{\mathrm{lay}(x),\mathrm{up}(x)}\}$ is satisfied, there is a 2-color 2-path R between $b^k_{\beta(C),\gamma(C)}$ and $u_{\mathrm{lay}(x),\mathrm{up}(x)}$. Then the internal vertex z of R belongs to M. By (P$_6$) there is a literal ℓ_k which belongs to a clause C_2 and contains a variable x_2 such that $z = \mathrm{mid}(x_2)$ and $b^k_{\beta(C),\gamma(C)} = b^k_{\beta(C_2),\gamma(C_2)}$. In particular, $\beta(C) = \beta(C_2)$ and $\gamma(C) = \gamma(C_2)$. Assume $C_2 \neq C$. There is a variable, say x_3, corresponding to edge $\mathrm{mid}(x_2)u_{\mathrm{lay}(x),\mathrm{up}(x)}$, i.e., $\mathrm{mid}(x_2) = \mathrm{mid}(x_3)$ and $u_{\mathrm{lay}(x),\mathrm{up}(x)} = u_{\mathrm{lay}(x_3),\mathrm{up}(x_3)}$. It follows that C and C_2 are adjacent in $G_{\beta(C)}$, which contradicts the fact that $\gamma(C) = \gamma(C_2)$. Hence $C_2 = C$, i.e., there is exactly one 2-path between $b^k_{\beta(C),\gamma(C)}$ and $u_{\mathrm{lay}(x),\mathrm{up}(x)}$, and it goes through $\mathrm{mid}(x)$. Since $c(\mathrm{mid}(x)b^k_{\beta(C),\gamma(C)}) = F$ and the path is 2-color, we get that $c(u_{\mathrm{lay}(x),\mathrm{up}(x)}\mathrm{mid}(x)) = T$. Hence $\xi(\ell_k) = \xi(x) = T$, so clause C is satisfied, as required.

It finishes the proof. (The analysis of the size of the resulting instance and its other properties described in the claim is not immediate; because of the space constraints we skip them here.) ◀

The proof of the following lemma is non-trivial, but standard (see lemmas 4 and 5 in the full version [17].)

▶ **Lemma 9** (★). *For any fixed $k \geq 2$, there is a polynomial time algorithm which given an instance $I = (G = (V, E), S, c_0)$ of* Subset Rainbow 2-Coloring Extension *constructs an equivalent instance $(G' = (V', E'), S')$ of* Subset Rainbow k-Coloring *such that $|V'| = O(k|V|k^2\ell)$, $|E'| = |E| + O(k|V|) + |\mathrm{Dom}(c_0)| + O(k^2\ell)$, $|S'| = |S| + |E| + 2|\mathrm{Dom}(c_0)| + O(k^2\ell)$. Let $G_S = (V, S)$ and $G_{S'} = (V', S')$. Then $\Delta(G_{S'}) = O(\Delta(G_S) + \Delta(G) + |\mathrm{Dom}(c_0)|/\ell)$. Moreover if we are given a proper vertex p-coloring of the graph $G_S = (V, S)$ then we can output also a proper vertex $(p + 3)$-coloring of the graph $G_{S'} = (V', S')$.*

2.3 From Subset Rainbow k-Coloring to Rainbow k-Coloring

The basic idea of our reduction from Subset Rainbow k-Coloring to Rainbow k-Coloring is to modify the graph so that the pairs of vertices from $\bar{E} \setminus S$ can be somehow trivially satisfied, without affecting the satisfiability of S. To this end we use a notion of biclique covering number (called also bipartite dimension). The *biclique covering number* $\mathrm{bc}(G)$ of a graph G is the smallest number of biclique subgraphs of G that cover all edges of G. The following proposition is well-known.

▶ **Proposition 10** (Folklore). *It holds that $\mathrm{bc}(K_n) = \lceil \log n \rceil$, and the corresponding cover can be constructed in polynomial time.*

Proof. Assume $V(K_n) = \{0, \ldots, n-1\}$. The i-th biclique contains edges between the vertices that have 0 at the i-th bit and the vertices that have 1 at the i-th bit. ◀

Let $G = (V_1, V_2, E)$ be a bipartite graph. Then $\hat{G}$ denotes the bipartite complement of G, i.e, the bipartite graph $(V_1, V_2, \{v_1v_2 \ : \ v_1 \in V_1,\ v_2 \in V_2, \text{ and } v_1v_2 \notin E\})$. We will use the following result of Jukna. Recall that we denote $\Delta_1(G) = \max\{\Delta(G), 1\}$.

▶ **Theorem 11** (Jukna [16]). *If G is an n-vertex bipartite graph, then $\mathrm{bc}(\hat{G}) = O(\Delta_1(G) \log n)$.*

Let us call the cover from Theorem 11 the *Jukna cover*. In our application we need to be able to *compute* the Jukna cover fast.

▶ **Lemma 12.** *The Jukna cover can be constructed in (i) expected polynomial time, or (ii) deterministic $2^n n^{O(1)}$ time.*

Proof. Denote $\Delta = \Delta(G)$. If $\Delta = 0$ the claim follows from Proposition 10, so in what follows assume $\Delta \geq 1$. Jukna [16] shows a simple worst-case linear time algorithm which samples a biclique in G. Then it is proved that after sampling t bicliques, the probability that there is an edge not covered by one of the bicliques is at most $n^2 e^{-t/(\Delta e)}$. It follows that the probability that more than $\Delta e(2 \ln n + 1)$ samples are needed is at most e^{-1}. If after $\Delta e(2 \ln n + 1)$ samples some edges is not covered, we discard all the bicliques found and repeat the whole algorithm from the scratch. The expected number of such restarts is $1/(1 - e^{-1}) = O(1)$.

Now we proceed to the second part of the claim. Let $G = (V_1, V_2, E)$. For every subset $A \subseteq V_1$ we define the biclique $B_A = (A, B, E_A)$, where B is the set of vertices of V_2 adjacent in $\hat{G}$ to all vertices of A. Clearly, B_A is a subgraph of $\hat{G}$ and for every subset $A \subseteq V_1$ it can be found in time linear in the size of $\hat{G}$. Our deterministic algorithm works as follows: as long as not all edges of $\hat{G}$ are covered, it picks the biclique B_A which maximizes the number of new covered edges of $\hat{G}$. Since all the bicliques in the set $\{B_A \ : \ A \subseteq V_1\}$ can be listed in time $O(2^n|E(\hat{G})|)$, the total running time is $t2^n n^{O(1)}$, where t is the size of the returned cover. It suffices to show that $t = O(\Delta \log n)$.

Jukna [16] shows that if set A is chosen by picking every vertex of V_1 independently with probability $\frac{1}{\Delta}$, then for any edge $uv \in E(\hat{G})$, $\Pr[uv \in E_A] \geq \frac{1}{\Delta e}$. Consider any step of our algorithm and let $R \subseteq E(\hat{G})$ be the set of the edges of $\hat{G}$ which are not covered yet. By the bound on $\Pr[uv \in E_A]$ and the linearity of expectation a set A sampled as described above covers at least $|R|/(\Delta e)$ new edges in expectation. In particular, it implies that there exists a set $A \subseteq V_1$ that covers at least $|R|/(\Delta e)$ new edges. Let $\alpha = (1 - \frac{1}{\Delta e})^{-1}$. By the Taylor expansion of $\log(1-x)$, it follows that $t = O(\log_\alpha |E(\hat{G})|) = O(\log n / \log \alpha) = O(\Delta \log n)$. ◀

▶ **Lemma 13.** *Let G be an n-vertex graph with a given proper vertex p-coloring. Then the edges of $\bar{G}$ can be covered by $O(p^2\Delta_1(G)\log n)$ bicliques from $\bar{G}$ so that any edge of G and any biclique have at most one common vertex. This cover can be constructed in (i) expected polynomial time, or (ii) deterministic $2^n n^{O(1)}$ time.*

Proof. The edges of $\bar{G}$ between the vertices of any color class form a clique, so by Proposition 10 we can cover its edges using $O(\log n)$ bicliques. If an edge of G has both endpoints in such a biclique, these endpoints have the same color, contradiction. For two different colors i and j the edges of G between their color classes form a bipartite graph of maximum degree at most $\Delta(G)$. Hence by Lemma 12 we can cover the edges of its bipartite complement using $O(\Delta_1(G)\log n)$ bicliques. If an edge uv of G has both endpoints in such a biclique, then either (i) these endpoints have the same color, contradiction, or (ii) these endpoints belong to two different parts of the biclique, so uv is in the biclique and hence $uv \in E(\bar{G})$, a contradiction. Summing over all color classes and pairs of color classes, we use $O(p^2\Delta_1(G)\log n)$ bicliques, as required. ◀

Now we proceed to the actual reduction.

▶ **Lemma 14.** *Given an instance $(G = (V,E), S)$ of* Subset Rainbow 2-Coloring *together with a proper p-coloring of the graph $G_S = (V,S)$, one can construct an equivalent instance G' of* Rainbow 2-Coloring *such that $|V(G')| = O(|V| + kp^2\Delta_1(G_S)\log|V|)$, $|E(G')| = O(|E(G)|+(|V|+p^2\Delta_1(G_S)\log|V|)\cdot p^2\Delta_1(G_S)\log|V|)$. The construction algorithm can run in (i) expected polynomial time, or (ii) deterministic $2^{|V|}|V|^{O(1)}$ time.*

Proof. Here we focus on the $k = 2$ case. The $k \geq 3$ case is significantly more technical — see the details in the full version. Let us consider a biclique covering of the complement of the graph G_S with $q = O(p^2\Delta_1(G_S)\log n)$ bicliques $(U_1, V_1; E_1), (U_2, V_2; E_2), \ldots, (U_q, V_q; E_q)$ as in Lemma 13. Let $W = \{w_1, w_2, \ldots, w_q\}$, $T = \{t_1, t_2, t_3\}$, $V(G') = V \cup W \cup T$ and $E(G') = E(G) \cup (W \times W) \cup (T \times T) \cup (\{t_2\} \times W) \cup (\{t_3\} \times (V \cup W)) \cup \left(\bigcup_{1 \leq i \leq q} \{w_i\} \times (U_i \cup V_i)\right)$ (we abuse the notation assuming that $\times$ operator returns *unordered* pairs minus loops). Because of the space limitation the equivalence proof is deferred to the full version. ◀

2.4 Putting everything together

By pipelining lemmas 7, 8, and 9 we get the following corollary.

▶ **Corollary 15.** *Fix $k \geq 2$. Given a* 3-SAT *formula φ with m clauses one can construct in polynomial time an equivalent instance $(G = (V,E), S)$ of* Subset Rainbow *k-*Coloring *such that $|V| = O(m^{2/3})$, $|E| = O(m)$, $\Delta((V,S)) = O(m^{1/3})$, and the graph $G_S = (V,S)$ is $O(1)$-colorable.*

Note that in Corollary 15 we have $|S| = |V|\Delta((V,S)) = O(m)$. It follows that the Sparsification Lemma (Lemma 6) and Corollary 15 imply Theorem 3.

Pipelining Corollary 15 and Lemma 14 gives the following corollary.

▶ **Corollary 16.** *Fix $k \geq 2$. Given a 3-SAT formula φ with $O(m)$ clauses one can construct an equivalent instance G of* Rainbow k-Coloring *with $O(m^{2/3})$ vertices and $O(m \log m)$ edges. The construction algorithm can run in (i) expected polynomial time, or (ii) deterministic $2^{O(m^{2/3})}$ time.*

Again, the above and the Sparsification Lemma immediately imply Theorem 2.

3 Algorithms for Subset Rainbow k-Coloring

In this section we study FPT algorithms for Subset Rainbow k-Coloring parameterized by $|S|$. We provide two such algorithms, based on different approaches: one for $k = 2$ case, and one (slightly slower) for the general case. Consider an instance (G, S) of the Subset Rainbow k-Coloring problem. Note that we can assume that $S \subseteq \bar{E}$, since any constraint $\{u, v\} \in E$ is satisfied in every edge coloring. Moreover, we say that a pair $\{u, v\}$ is *feasible* when the distance between u and v is at most k. The set of all feasible pairs is denoted by $F(G)$. Clearly, when S contains a request which is not feasible, then (G, S) is a trivial NO-instance. Hence, throughout this section we assume $S \subseteq \bar{E} \cap F(G)$.

3.1 The $k = 2$ case

For any $X \subseteq S$ let $\mathcal{P}_X$ be the set of all 2-edge paths between the pairs of vertices in X. Denote $E(\mathcal{P}_X) = \bigcup_{P \in \mathcal{P}_X} E(P)$. For two edges $e_1, e_2 \in E(G)$ we say that e_1 and e_2 are *linked by* X, denoted as $e_1 \sim_X e_2$ when there are two paths $P_1, P_2 \in \mathcal{P}_X$ (possibly $P_1 = P_2$) such that $e_1 \in E(P_1)$, $e_2 \in E(P_2)$ and $E(P_1) \cap E(P_2) \neq \emptyset$. Let $\approx_X$ be the transitive closure of $\sim_X$. Then $\approx_X$ is an equivalence relation. Recall that $E(G)/\approx_X$ denotes the quotient set of the relation $\approx_X$. The main observation of this section is the following theorem.

▶ **Theorem 17.** *The number of 2-colorings of $E(G)$ that satisfy all the pairs in S is equal to $\sum_{X \subseteq S} (-1)^{|X|} 2^{|E(G)/\approx_X|}$.*

In the proof we make use of the well-known inclusion-exclusion principle. Below we state it in the intersection version (see, e.g., [10])

▶ **Theorem 18** (Inclusion–exclusion principle, intersection version)**.** *Let $A_1, \ldots, A_n \subseteq U$, where U is a finite set. Denote $\bigcap_{i \in \emptyset}(U \setminus A_i) = U$. Then*

$$|\bigcap_{i \in [n]} A_i| = \sum_{X \subseteq [n]} (-1)^{|X|} |\bigcap_{i \in X} (U \setminus A_i)|.$$

Proof of Theorem 17. Let us define, for every pair $\{u, v\} \in S$ (say, $u < v$), the set $A_{u,v}$ of 2-edge colorings of G that satisfy $\{u, v\}$. Note that the number of rainbow 2-colorings of G that satisfy all the pairs in S is equal to $|\bigcap_{\{u,v\} \in S} A_{u,v}|$. By Theorem 18 it suffices to show that, for any subset $X \subseteq S$, the number $\#_X$ of 2-colorings such that *none* of the pairs in X is satisfied, equals $2^{|E(G)/\approx_X|}$.

Fix any coloring c that does not satisfy any pair from X. Then every path from $\mathcal{P}_X$ has both edges of the same color. Hence, for two edges $e_1, e_2 \in E(G)$, if $e_1 \sim_X e_2$ then e_1 and e_2 are colored by c with the same color. It follows that for any equivalence class A of $\approx_X$, all edges of A are have the same color in c. This proves that $\#_X \leq 2^{|E(G)/\approx_X|}$.

For every function $c_0 : (E(G)/\approx_X) \to \{1,2\}$ we can define the coloring $c : E(G) \to \{1,2\}$ by putting $c(e) = c_0([e]_{\approx_X})$ for every edge $e \in E(G)$. (Note that the edges that do not belong to any path in $\mathcal{P}_X$ form singleton equivalence classes.) Then, c does not satisfy any pair from X, because if some pair $\{u,v\}$ is satisfied then there is a 2-color path uxv; but $ux \sim_X xv$, so $[ux]_{\approx_X} = [xv]_{\approx_X}$ and $c(ux) = c(xv)$, a contradiction. It follows that $\#_X \geq 2^{|E(G)/\approx_X|}$. ◀

Since it is a standard exercise to compute the relation $X \subseteq S$ in $O(|E| + |S| \cdot |V|)$ time (see the full version), we get the following corollary. (Let us remark here that the algorithm from Corollary 19 only decides whether the coloring exists, without finding it. However, by a minor modification of the algorithm it can construct the coloring; see the full version.)

▶ **Corollary 19.** *For any graph $G = (V,E)$ and a set of requests S the number of 2-colorings of E that satisfy all the pairs in S can be computed in $O(2^{|S|}(|E| + |S| \cdot |V|))$ time and polynomial space. In particular,* Subset Rainbow 2-Coloring *can be decided within the same time.*

3.2 The general case

In this section we use partial colorings. For convenience, a partial coloring is represented as a function $c : E \to [k] \cup \{\perp\}$, where the value $\perp$ corresponds to an uncolored edge. By $\mathrm{Dom}(c)$ we denote the domain of the corresponding partial function, i.e., $\mathrm{Dom}(c) = c^{-1}([k])$. The partial coloring which does not color anything, i.e., is constantly equal to $\perp$ is denoted by $c_\perp$.

For a graph $G = (V,E)$ consider a partial edge coloring $c : E \to [k] \cup \{\perp\}$. A *guide function* is any function of the form $f : S \to \binom{\mathrm{Dom}(c)}{\leq k}$, i.e., any function that assigns sets of at most k colored edges to all requests in S. A constant guide function equal to $\emptyset$ for every request in S is denoted by $g_{S,\emptyset}$. Pick any pair $\{u,v\} \in S$. We say that a walk W connecting u and v is f-guided if every color appears at most once on W, and $f(\{u,v\}) \subseteq E(W)$. We say that a coloring c is (f,S)-rainbow when for every pair $\{u,v\} \in S$ there is an f-guided walk between u and v. Note that (G,S) is a YES-instance of Subset Rainbow k-Coloring iff there is an $(g_{S,\emptyset}, S)$-rainbow coloring. Indeed, every rainbow walk contains a rainbow path.

The following lemma is going to be useful in our branching algorithm.

▶ **Lemma 20.** *Let $G = (V,E)$ be a graph, and let S be a set of requests. Let $c_0 : E \to [k]$ be a partial edge coloring and let $f : S \to \binom{\mathrm{Dom}(c_0)}{\leq k}$ be a guide function. Then, given a pair $\{u,v\} \in S$ in time $2^k n^{O(1)}$ one can find an f-guided u-v walk of length at most k, if it exists.*

Proof. The algorithm is as follows. We can assume that $f(\{u,v\})$ does not contain two edges of the same color, for otherwise the requested walk does not exist. For every $e \in f(\{u,v\})$ we remove all the edges of color $c_0(e)$. Next, we put back edges of $f(\{u,v\})$. Then it suffices to find in the resulting graph G' any u-v path of length at most k and with no repeated colors that visits all the colors of the edges in $f(\{u,v\})$. This is done using dynamic programming. For every vertex $x \in V$, subset $X \subseteq [k]$ and integer $\ell = 0, \ldots, k$ we find the boolean value $T[x,X,\ell]$ which is true iff there is a u-x walk of length ℓ which does not repeat colors and visits all the colors from X, but not more. We initialize $T[u,\emptyset,0] = \texttt{true}$ and $T[x,\emptyset,0] = \texttt{false}$ for every $x \neq u$. Next we iterate through the remaining triples (x,X,ℓ), in the nondecreasing order of ℓ and X's cardinalities. The value of $T[x,X,\ell]$ is then computed using the formula

$$T[x,X,\ell] = \bigvee_{yx \in c_0^{-1}(X \cup \{\perp\}) \cap E(G')} T[y, X \setminus \{c_0(yx)\}, \ell - 1].$$

Pseudocode 1: FindColoring(S_0, c_0, f)

```
if $S_0 = \emptyset$ then
    return $c_0$
if for some $r \in S_0$ there are edges $e_1, e_2 \in f(r)$ with $c_0(e_1) = c_0(e_2)$ then
    return null
Pick any $\{u, v\} \in S_0$;
Find any $f$-guided $u$-$v$ walk $W$ of length at most $k$ using Lemma 20;
if $W$ does not exist then
    return null
Let $c_1$ be obtained from $c_0$ by coloring the uncolored edges of $W$ to get a rainbow walk;
if FindColoring($S_0 \setminus \{u, v\}, c_1, f|_{S_0 \setminus \{u,v\}}$) $\neq$ null then return the coloring found;
for $e \in E(W) \setminus \mathrm{Dom}(c_0)$ do
    for $\alpha \in [k]$ do
        for $r \in S_0 \setminus \{\{u, v\}\}$ do
            Let $c_{e,\alpha}$ be obtained from $c_0$ by coloring $e$ with $\alpha$;
            Let $f_{e,r}$ be obtained from $f$ by putting $f(r) := f(r) \cup \{e\}$;
            if FindColoring($S_0, c_{e,\alpha}, f_{e,r}$) $\neq$ null then return the coloring found;
return null
```

The requested walk exists iff $T[v, X, \ell] = \texttt{true}$ for any $\ell = 0, \ldots, k$ and X such that $c_0(f(\{u, v\})) \subseteq X$. The walk is retrieved using standard DP methods. ◀

Now we are ready to describe our branching algorithm. Let $(G = (V, E), S)$ be the input instance. Our algorithm consists of a recursive procedure FindColoring which gets three parameters: S_0 (a set of requests), $c_0 : E \to [k] \cup \{\perp\}$ (a partial coloring), and a guide function $f : S \to \binom{\mathrm{Dom}(c_0)}{\leq k}$. It is assumed that for every request $r \in S$, every pair of different edges $e_1, e_2 \in f(r)$ is colored differently by c_0. The goal of the procedure FindColoring is to find an (f, S_0)-rainbow coloring $c : E \to [k]$ which extends c_0. Thus the whole problem is solved by invoking FindColoring($S, c_\perp, g_{S,\emptyset}$). A rough description of FindColoring is as follows. We pick any pair $\{u, v\} \in S_0$ and we find any f-guided u-v walk W of length at most k using Lemma 20. Let c_1 be obtained from c_0 by coloring the uncolored edges of W to get a rainbow walk. If FindColoring($S_0 \setminus \{u, v\}, c_1, f|_{S_0 \setminus \{u,v\}}$) returns a coloring, we are done. But if no such coloring exists then we know that we made a wrong decision: coloring some of the uncolored edges e of W into $c_1(e)$ (instead of some color α) makes some other request $r \in S_0 \setminus \{\{u, v\}\}$ impossible to satisfy. For every possible triple (e, α, r) we invoke FindColoring with the same set of requests S_0, partial coloring c_0 extended by coloring e with α, and the guide function f extended by putting $f(r) := f(r) \cup \{e\}$.

A precise description of procedure FindColoring can be found in Pseudocode 1. The following lemma proves its correctness.

▶ **Lemma 21.** *Procedure* FindColoring *invoked with parameters* (S_0, c_0, f) *finds an* (f, S_0)*-rainbow coloring* $c : E \to [k]$ *which extends* c_0*, whenever it exists.*

Proof. The proof is by induction on the sum of $|S_0|$ and the number of uncolored edges. It is clear that if $|S_0| = 0$ or all the edges are colored then the algorithm behaves correctly. In the induction step, the only non-trivial thing to check is whether any of the calls in lines 10 or 16 returns a coloring, provided that there is a solution, i.e., an (f, S_0)-rainbow coloring $c : E \to [k]$ which extends c_0. Assume that no coloring is returned in Line 16. Then for every edge $e \in E(W) \setminus \mathrm{Dom}(c_0)$, and request $r \in S_0 \setminus \{\{u, v\}\}$ coloring c is not a $(f_{e,r}, S_0)$-rainbow coloring, for otherwise the call FindColoring($S_0, c_{e,c(e)}, f_{e,r}$) returns a coloring. If follows that for every edge $e \in E(W) \setminus \mathrm{Dom}(c_0)$ and request $r \in S_0 \setminus \{\{u, v\}\}$ the walk that realizes the request r in the coloring c does not contain e. Hence, the following coloring

$$c'(e) = \begin{cases} c(e) & \text{if } e \notin E(W), \\ c_1(e) & \text{if } e \in E(W). \end{cases}$$

is another (f, S_0)-rainbow coloring, and it extends c_1. It follows that the call in Line 10 returns a coloring, as required. ◀

▶ **Theorem 22.** *For every integer k, there is an FPT algorithm for* Subset Rainbow k-Coloring *parameterized by $|S|$. The algorithm runs in time $(k^2|S|)^{k|S|}2^k n^O(1)$, in particular in $|S|^{O(|S|)}n^{O(1)}$ time for every fixed k.*

Proof. By Lemma 21 Subset Rainbow k-Coloring is solved by invoking FindColoring($S, c_\perp, g_{S,\emptyset}$). Note that whenever we go deeper in the recursion either some request of S_0 gets satisfied, or $|f(r)|$ increases for some $r \in S_0$. When $|f(r)|$ increases to $k+1$, the corresponding recursive call returns `null` immediately (because the condition in Line 3 holds). It follows that the depth of the recursion is at most $|S|k$. Since in every call of Subset Rainbow k-Coloring the algorithm uses time $2^k n^{O(1)}$ (by Lemma 21) and branches into at most $1 + k^2(|S| - 1) \leq k^2|S|$ recursive calls, the total time is $(k^2|S|)^{k|S|}2^k n^O(1)$, as required. ◀

4 Further Work

We believe that this work only initiates the study of fine-grained complexity of variants of Rainbow k-Coloring. In particular, many open questions are still unanswered. The ultimate goal is certainly to get tight bounds. We pose the following two conjectures.

▶ **Conjecture 23.** *For any integer $k \geq 2$, there is no $2^{o(|E|)}n^{O(1)}$-time algorithm for* Rainbow k-Coloring*, unless ETH fails.*

▶ **Conjecture 24.** *For any integer $k \geq 2$, there is no $2^{o(n^2)}n^{O(1)}$-time algorithm for* Rainbow k-Coloring*, unless ETH fails.*

Note that in this work we have settled Conjecture 23 for Subset Rainbow k-Coloring, and for Rainbow k-Coloring we showed a slightly weaker, $2^{o(|E|/\log|E|)}n^{O(1)}$ bound. However, avoiding this $\log|E|$ factor seems to constitute a considerable technical challenge.

In this paper we gave two algorithms for Subset Rainbow k-Coloring parameterized by $|S|$, one working in $2^{|S|}n^{O(1)}$ time for $k = 2$ and another, working in time $|S|^{O(|S|)}n^{O(1)}$ for every fixed k. We conjecture that there exists an algorithm running in time $2^{O(|S|)}n^{O(1)}$ for every fixed k.

Finally, we would like to propose yet another parameterization of Rainbow k-Coloring. Assume we are given a graph $G = (V, E)$ and a subset of vertices $S \subseteq V$. In the Steiner Rainbow k-Coloring problem the goal is to determine whether there is a rainbow k-coloring such that every pair of vertices in S is connected by a rainbow path. By our Theorem 2, Steiner Rainbow k-Coloring has no algorithm running in time $2^{o(|S|^{3/2})}$, under ETH. On the other hand, our algorithm for Subset Rainbow k-Coloring implies that Steiner Rainbow k-Coloring parameterized by $|S|$ admits an FPT algorithm with running time of $2^{O(|S|^2 \log|S|)}n^{O(1)}$. It would be interesting make the gap between these bounds smaller.

References

1 Prabhanjan Ananth, Meghana Nasre, and Kanthi K. Sarpatwar. Rainbow connectivity: Hardness and tractability. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011)*, pages 241–251, 2011.

2 Yair Caro, Arie Lev, Yehuda Roditty, Zsolt Tuza, and Raphael Yuster. On rainbow connection. *Electron. J. Combin*, 15(1):R57, 2008.
3 Sourav Chakraborty, Eldar Fischer, Arie Matsliah, and Raphael Yuster. Hardness and algorithms for rainbow connection. *J. of Combinatorial Optimization*, 21(3):330–347, 2009.
4 L. Sunil Chandran and Deepak Rajendraprasad. Rainbow Colouring of Split and Threshold Graphs. *Computing and Combinatorics*, pages 181–192, 2012.
5 L. Sunil Chandran and Deepak Rajendraprasad. Inapproximability of rainbow colouring. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013)*, pages 153–162, 2013.
6 L. Sunil Chandran, Deepak Rajendraprasad, and Marek Tesař. Rainbow colouring of split graphs. *Discrete Applied Mathematics*, 2015. doi:10.1016/j.dam.2015.05.021.
7 Gary Chartrand, Garry L. Johns, Kathleen A. McKeon, and Ping Zhang. Rainbow connection in graphs. *Mathematica Bohemica*, 133(1), 2008.
8 Gary Chartrand and Ping Zhang. *Chromatic graph theory.* CRC press, 2008.
9 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 *Proc. of the 27th Annual ACM-SIAM Symp. on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1643–1649, 2016.
10 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Dániel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms.* Springer, 2015.
11 Reinhard Diestel. *Graph Theory.* Springer-Verlag Heidelberg, 2010.
12 Eduard Eiben, Robert Ganian, and Juho Lauri. On the complexity of rainbow coloring problems. In *Proceedings of the Twenty-Sixth International Workshop on Combinatorial Algorithms, IWOCA 2015, Verona, Italy, October 5-7*, pages 209–220, 2015. URL: http://arxiv.org/abs/1510.03614, doi:10.1007/978-3-319-29516-9_18.
13 Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. *Journal of SIAM*, 10:196–210, 1962.
14 Russell Impagliazzo and Ramamohan Paturi. On the Complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.
15 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.
16 Stasys Jukna. On set intersection representations of graphs. *Journal of Graph Theory*, 61(1):55–75, 2009. doi:10.1002/jgt.20367.
17 Lukasz Kowalik, Juho Lauri, and Arkadiusz Socala. On the fine-grained complexity of rainbow coloring. *CoRR*, abs/1602.05608, 2016. URL: http://arxiv.org/abs/1602.05608.
18 Eugene L. Lawler. A note on the complexity of the chromatic number problem. *Information Processing Letters*, 5(3):66–67, 1976.
19 Van Bang Le and Zsolt Tuza. Finding optimal rainbow connection is hard. Technical Report CS-03-09, Universität Rostock, 2009.
20 Xueliang Li, Yongtang Shi, and Yuefang Sun. Rainbow Connections of Graphs: A Survey. *Graphs and Combinatorics*, 29(1):1–38, 2012.
21 Arkadiusz Socała. Tight lower bound for the channel assignment problem. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 662–675, 2015.
22 Craig A. Tovey. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics*, 8(1):85–89, 1984. doi:10.1016/0166-218X(84)90081-7.
23 Kei Uchizawa, Takanori Aoki, Takehiro Ito, Akira Suzuki, and Xiao Zhou. On the Rainbow Connectivity of Graphs: Complexity and FPT Algorithms. *Algorithmica*, 67(2):161–179, 2013.

A Randomized Polynomial Kernelization for Vertex Cover with a Smaller Parameter

Stefan Kratsch

Universität Bonn, Institut für Informatik, Bonn, Germany
kratsch@cs.uni-bonn.de

Abstract

In the VERTEX COVER problem we are given a graph $G = (V, E)$ and an integer k and have to determine whether there is a set $X \subseteq V$ of size at most k such that each edge in E has at least one endpoint in X. The problem can be easily solved in time $\mathcal{O}^*(2^k)$, making it fixed-parameter tractable (FPT) with respect to k. While the fastest known algorithm takes only time $\mathcal{O}^*(1.2738^k)$, much stronger improvements have been obtained by studying *parameters that are smaller than* k. Apart from treewidth-related results, the arguably best algorithm for VERTEX COVER runs in time $\mathcal{O}^*(2.3146^p)$, where $p = k - LP(G)$ is only the excess of the solution size k over the best fractional vertex cover (Lokshtanov et al. TALG 2014). Since $p \leq k$ but k cannot be bounded in terms of p alone, this strictly increases the range of tractable instances.

Recently, Garg and Philip (SODA 2016) greatly contributed to understanding the parameterized complexity of the VERTEX COVER problem. They prove that $2LP(G) - MM(G)$ is a lower bound for the vertex cover size of G, where $MM(G)$ is the size of a largest matching of G, and proceed to study parameter $\ell = k - (2LP(G) - MM(G))$. They give an algorithm of running time $\mathcal{O}^*(3^\ell)$, proving that VERTEX COVER is FPT in ℓ. It can be easily observed that $\ell \leq p$ whereas p cannot be bounded in terms of ℓ alone. We complement the work of Garg and Philip by proving that VERTEX COVER admits a randomized polynomial kernelization in terms of ℓ, i.e., an efficient preprocessing to size polynomial in ℓ. This improves over parameter $p = k - LP(G)$ for which this was previously known (Kratsch and Wahlström FOCS 2012).

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Vertex cover, parameterized complexity, kernelization

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.59

1 Introduction

A *vertex cover* of a graph $G = (V, E)$ is a set $X \subseteq V$ such that each edge $e \in E$ has at least one endpoint in X. The VERTEX COVER problem of determining whether a given graph G has a vertex cover of size at most k has been an important benchmark problem in parameterized complexity for both *fixed-parameter tractability* and *(polynomial) kernelization*,[1] which are the two notions of tractability for parameterized problems. Kernelization, in particular, formalizes the widespread notion of efficient preprocessing, allowing a rigorous study (cf. [14]). We present a randomized polynomial kernelization for VERTEX COVER for the to-date smallest parameter, complementing a recent fixed-parameter tractability result by Garg and Philip [10].

[1] Definitions can be found in the full version. Note that we use ℓ, rather than k, as the default symbol for parameters and use VERTEX COVER(ℓ) to refer to the VERTEX COVER problem with parameter ℓ.

Let us first recall what is known for the so-called *standard parameterization* VERTEX COVER(k), i.e., with parameter $\ell = k$: There is a folklore $\mathcal{O}^*(2^k)$ time[2] algorithm for testing whether a graph G has a vertex cover of size at most k, proving that VERTEX COVER(k) is fixed-parameter tractable (FPT); this has been improved several times with the fastest known algorithm due to Chen et al. [4] running in time $\mathcal{O}^*(1.2738^k)$. Under the Exponential Time Hypothesis of Impagliazzo et al. [11] there is no algorithm with runtime $\mathcal{O}^*(2^{o(k)})$. The best known kernelization for VERTEX COVER(k) reduces any instance (G, k) to an equivalent instance (G', k') with $|V(G')| \leq 2k$; the total size is $\mathcal{O}(k^2)$ [3]. Unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$ and the polynomial hierarchy collapses there is no kernelization to size $\mathcal{O}(k^{2-\varepsilon})$ [8].

At first glance, the FPT and kernelization results for VERTEX COVER(k) seem essentially best possible. This is true for parameter $\ell = k$, but there are *smaller parameters* ℓ' for which both FPT-algorithms and polynomial kernelizations are known. The motivation for this is that even when $\ell' = \mathcal{O}(1)$, the value $\ell = k$ may be as large as $\Omega(n)$, making both FPT-algorithm and kernelization for parameter k useless for such instances (time $2^{\Omega(n)}$ and size guarantee $\mathcal{O}(n)$). In contrast, for $\ell' = \mathcal{O}(1)$ an FPT-algorithm with respect to ℓ' runs in polynomial time (with only leading constant depending on ℓ'). Let us discuss the relevant type of smaller parameter, which relates to *lower bounds on the optimum* and was introduced by Mahajan and Raman [19]; two other types are discussed briefly under related work.

Two well-known lower bounds for the size of vertex covers for a graph $G = (V, E)$ are the maximum size of a matching of G and the smallest size of fractional vertex covers for G; we (essentially) follow Garg and Philip [10] in denoting these two values by $MM(G)$ and $LP(G)$. Note that the notation $LP(G)$ comes from the fact that fractional vertex covers come up naturally in the linear programming relaxation of the VERTEX COVER problem, where we must assign each vertex a fractional value such that each edge is incident with total value of at least 1. In this regard, it is useful to observe that the LP relaxation of the MAXIMUM MATCHING problem is exactly the dual of this. Accordingly, we have $MM(G) \leq LP(G)$ since each integral matching is also a fractional matching, i.e., with each vertex incident to a total value of at most 1. Similarly, using $VC(G)$ to denote the minimum size of vertex covers of G we get $VC(G) \geq LP(G)$ and, hence, $VC(G) \geq LP(G) \geq MM(G)$.

A number of papers have studied vertex cover with respect to "above lower bound" parameters $\ell' = k - MM(G)$ or $\ell'' = k - LP(G)$ [24, 23, 7, 21, 17]. Observe that $k \geq k - MM(G) \geq k - LP(G)$. For the converse, note that k can be unbounded in terms of $k - MM(G)$ and $k - LP(G)$, whereas $k - MM(G) \leq 2(k - LP(G))$ holds [16, 12]. Thus, from the perspective of achieving fixed-parameter tractability (and avoiding large parameters) both parameters are equally useful for improving over parameter k. Razgon and O'Sullivan [24] proved fixed-parameter tractability of ALMOST 2-SAT(k), which implies that VERTEX COVER(k − MM) is FPT due to a reduction to ALMOST 2-SAT(k) by Mishra et al. [20]. Using $k - MM(G) \leq 2(k - LP(G))$, this also entails fixed-parameter tractability of VERTEX COVER(k − LP).

After several improvements [23, 7, 21, 17] the fastest known algorithm, due to Lokshtanov et al. [17], runs in time $\mathcal{O}^*(2.3146^{k-MM(G)})$. The algorithms of Narayanaswamy et al. [21] and Lokshtanov et al. [17] achieve the same parameter dependency also for parameter $k - LP(G)$. The first (and to our knowledge only) kernelization result for these parameters is a randomized polynomial kernelization for VERTEX COVER(k − LP) by Kratsch and Wahlström [16], which of course applies also to the larger parameter $k - MM(G)$.

Recently, Garg and Philip [10] made an important contribution to understanding the

[2] We use $\mathcal{O}^*$ notation, which suppresses polynomial factors.

parameterized complexity of the VERTEX COVER problem by proving it to be FPT with respect to parameter $\ell = k - (2LP(G) - MM(G))$. Building on an observation of Lovász and Plummer [18] they prove that $VC(G) \geq 2LP(G) - MM(G)$, i.e., that $2LP(G) - MM(G)$ is indeed a lower bound for the minimum vertex covers size of any graph G. They then design a branching algorithm with running time $\mathcal{O}^*(3^\ell)$ that builds on the well-known Gallai-Edmonds decomposition for maximum matchings to guide its branching choices.

VERTEX COVER$(k - (2\text{LP} - \text{MM}))$
Input: A graph $G = (V, E)$ and an integer $k \in \mathbb{N}$.
Parameter: $\ell = k - (2LP(G) - MM(G))$ where $LP(G)$ is the minimum size of fractional vertex covers for G and $MM(G)$ is the maximum cardinality of matchings of G.
Question: Does G have a vertex cover of size at most k, i.e., a set $X \subseteq V$ of size at most k such that each edge of E has at least one endpoint in X?

Since $LP(G) \geq MM(G)$, we clearly have $2LP(G) - MM(G) \geq LP(G)$ and hence $\ell = k - (2LP(G) - MM(G))$ is indeed at most as large as the previously best parameter $k - LP(G)$. We can easily observe that $k - LP(G)$ cannot be bounded in terms of ℓ: For any odd cycle C of length $2s + 1$ we have $LP(C) = \frac{1}{2}(2s + 1)$, $VC(C) = s + 1$, and $MM(C) = s$. Thus, a graph G consisting of t vertex-disjoint odd cycles of length $2s + 1$ has $LP(G) = \frac{1}{2}t(2s + 1)$, $VC(G) = t(s + 1)$, and $MM(G) = ts$. For $k = VC(G) = t(s + 1)$ we get

$$\ell = k - (2LP(G) - MM(G)) = t(s + 1) - t(2s + 1) + ts = 0$$

whereas

$$k - LP(G) = t(s + 1) - \frac{1}{2}t(2s + 1) = \frac{1}{2}t(2s + 2) - \frac{1}{2}t(2s + 1) = \frac{1}{2}t.$$

Generally, it can be easily proved that $LP(G)$ and $2LP(G) - MM(G)$ differ by exactly $\frac{1}{2}$ on any *factor-critical* graph (cf. Proposition 4).

As always in parameterized complexity, when presented with a new fixed-parameter tractability result, the next question is whether the problem also admits a polynomial kernelization. It is well known that decidable problems are fixed-parameter tractable if and only if they admit a (not necessarily polynomial) kernelization.[3] Nevertheless, not all problems admit polynomial kernelizations and, in the present case, both an extension of the methods for parameter $k - LP(G)$ [16] or a lower bound proof similar to Cygan et al. [6] or Jansen [12, Section 5.3] (see related work) are conceivable.

Our result. We give a randomized polynomial kernelization for VERTEX COVER$(k - (2\text{LP} - \text{MM}))$. This improves upon parameter $k - LP(G)$ by giving a strictly smaller parameter for which a polynomial kernelization is known. At high level, the kernelization takes the form of a (randomized) polynomial parameter transformation from VERTEX COVER$(k - (2\text{LP} - \text{MM}))$ to VERTEX COVER$(k - \text{MM})$, i.e., a polynomial-time many-one (Karp) reduction with *output parameter polynomially bounded in the input parameter*. It is well known (cf. Bodlaender et

[3] We sketch this folklore fact for VERTEX COVER$(k - (2\text{LP} - \text{MM}))$: If the input is larger than 3^ℓ, where $\ell = k - (2LP(G) - MM(G))$, then the algorithm of Garg and Philip [10] runs in polynomial time and we can reduce to an equivalent small yes- or no-instance; else, the instance size is bounded by 3^ℓ; in both cases we get size at most 3^ℓ in polynomial time. The converse holds since a kernelization followed by any brute-force algorithm on an instance of, say, size $g(\ell)$ gives an FPT running time in terms of ℓ.

al. [2]) that this implies a polynomial kernelization for the source problem, i.e., for VERTEX COVER($k-(2\text{LP}-\text{MM})$) in our case. Let us give some more details of this transformation.

Since the transformation is between different parameterizations of the same problem, it suffices to handle parts of any input graph G where the input parameter $\ell = k-(2LP(G)-MM(G))$ is (much) smaller than the output parameter $k-MM(G)$. After the well-known LP-based preprocessing (cf. [10]), the difference in parameter values is equal to the number of vertices that are exposed (unmatched) by any maximum matching M of G. Consider the Gallai-Edmonds decomposition $V = A \mathbin{\dot\cup} B \mathbin{\dot\cup} D$ of $G=(V,E)$, where D contains the vertices that are exposed by at least one maximum matching, $A = N(D)$, and $B = V \setminus (A \cup D)$. Let M be a maximum matching and let t be the number of exposed vertices. There are t components of $G[D]$ that have exactly one exposed vertex each. The value $2LP(G)-MM(G)$ is equal to $|M|+t$ when $LP(G) = \frac{1}{2}|V|$, as implied by LP-based preprocessing.

To reduce the difference in parameter values we will remove all but $\mathcal{O}(\ell^4)$ components of $G[D]$ that have an exposed vertex; they are called *unmatched components* for lack of a matching edge to A and we can ensure that they are not singletons. It is known that any such component C is factor-critical and hence has no vertex cover smaller than $\frac{1}{2}(|C|+1)$; this exactly matches its contribution to $|M|+t$: It has $\frac{1}{2}(|C|-1)$ edges of M and one exposed vertex. Unless the instance is trivially **no** all but at most ℓ of these components C have a vertex cover of size $\frac{1}{2}(|C|+1)$, later called a *tight vertex cover*. The only reason not to use a tight vertex cover for C can be due to adjacent vertices in A that are not selected; this happens at most ℓ times. A technical lemma proves that this can always be traced to at most three vertices of C and hence at most three vertices in A that are adjacent with C.

In contrast, there are (matched, non-singleton) components C of $G[C]$ that together with a matched vertex $v \in A$ contribute $\frac{1}{2}(|C|+1)$ to the lower bound due to containing this many matching edges. To cover them at this cost requires not selecting vertex v. This in turn propagates along M-alternating paths until the cover picks both vertices of an M-edge, which happens at most ℓ times, or until reaching an unmatched component, where it may help prevent a tight vertex cover. We translate this effect into a two-way separation problem in an auxiliary directed graph. Selecting both vertices of an M-edge is analogous to a adding a vertex to the separator. Relative to a separator the question becomes which sets of at most three vertices of A that can prevent tight vertex covers are still reachable by propagation. At this point we can apply representative set tools from Kratsch and Wahlström [16] to identify a small family of such triplets that works for all separators (and hence for all so-called *dominant* vertex covers) and keep only the corresponding components.

Related work. Let us mention some further kernelization results for VERTEX COVER with respect to nonstandard parameters. There are two further types of interesting parameters:

1. *Width-parameters:* Parameters such as treewidth allow dynamic programming algorithms running in time, e.g., $\mathcal{O}^*(2^{\text{tw}})$, independently of the size of the vertex cover. It is known that there are no polynomial kernels for VERTEX COVER (or most other NP-hard problems) under such parameters [1]. The treewidth of a graph is upper bounded by the smallest vertex cover, whereas graphs of bounded treewidth can have vertex cover size $\Omega(n)$.
2. *"Distance to tractable case"-parameters:* VERTEX COVER can be efficiently solved on forests. By a simple enumeration argument it is fixed-parameter tractable when ℓ is the minimum number of vertices to delete such that G becomes a forest. Jansen and Bodlaender [13] gave a polynomial kernelization to $\mathcal{O}(\ell^3)$ vertices. Note that the vertex cover size is an upper bound on ℓ, whereas trees can have unbounded vertex cover size. The FPT-result can be carried over to smaller parameters corresponding to distance from

larger graph classes on which VERTEX COVER is polynomial-time solvable, however, Cygan et al. [6] and Jansen [12, Section 5.3] ruled out polynomial kernels for some of them. E.g., if ℓ is the deletion-distance to an outerplanar graph then there is no kernelization for VERTEX COVER(ℓ) to size polynomial in ℓ unless the polynomial hierarchy collapses [12].

Organization. Section 2 gives some preliminaries. In Section 3 we discuss vertex covers of factor-critical graphs and prove the claimed lemma about critical sets. Section 4 introduces a relaxation of the Gallai-Edmonds decomposition, called *nice decomposition*, and Section 5 explores the relation between nice decompositions and vertex covers. The kernelization for VERTEX COVER($k - (2\text{LP} - \text{MM})$) is given in Section 6. We conclude in Section 7.

2 Preliminaries

Parameterized complexity. We use standard definitions from parameterized complexity, with the difference of using ℓ as the default symbol for the parameter. We use VERTEX COVER(ℓ) to refer to the VERTEX COVER problem parameterized by ℓ, e.g., $\ell = k$ for the *standard parameterization* or $\ell = k - LP(G)$. For a detailed introduction to parameterized complexity we recommend the recent books by Downey and Fellows [9] and Cygan et al. [5].

Graphs. We require both directed and undirected graphs; all graphs are finite and simple, i.e., they have no parallel edges or loops. Accordingly, an undirected graph $G = (V, E)$ consists of a finite set V of vertices and a set $E \subseteq \binom{V}{2}$ of edges; a directed graph $H = (V, E)$ consists of a finite set V and a set $E \subseteq V^2 \setminus \{(v, v) \mid v \in V\}$. For clarity, all undirected graphs are called G and all directed graphs are called H (possibly with indices etc.). For a graph $G = (V, E)$ and vertex set $X \subseteq V$ we use $G - X$ to denote the graph induced by $V \setminus X$; we also use $G - v$ if $X = \{v\}$. Analogous definitions are used for directed graphs H.

Let $H = (V, E)$ be a directed graph and let S and T be two not necessarily disjoint vertex sets in H. A set $X \subseteq V$ is an *S, T-separator* if in $G - X$ there is no path from $S \setminus X$ to $T \setminus X$; note that X may overlap both S and T and that $S \cap T \subseteq X$ is required. The set T is *closest to* S if there is no S, T-separator X with $X \neq T$ and $|X| \leq |T|$, i.e., if T is the unique minimum S, T-separator in G (cf. [16]). Both separators and closeness have analogous definitions in undirected graphs but they are not required here.

▶ **Proposition 1** (cf. [16]). *Let $H = (V, E)$ be a directed graph and let $S, T \subseteq V$ such that T is closest to S. For any vertex $v \in V \setminus T$ that is reachable from S in $H - T$ there exist $|T| + 1$ (fully) vertex-disjoint paths from S to $T \cup \{v\}$.*

Proof. Assume for contradiction that such $|T| + 1$ directed paths do not exist. By Menger's Theorem there must be an $S, T \cup \{v\}$-separator X of size at most $|T|$. Observe that $X \neq T$ since v is reachable from S in $H - T$. Thus, X is an S, T-separator of size at most $|T|$ that is different from T; this contradicts closeness of T. ◀

For an undirected graph $G = (V, E)$, a *matching* is any set $M \subseteq E$ such that no two edges in M have an endpoint in common. If M is a matching in $G = (V, E)$ then we will say that a path is M-alternating if its edges are alternatingly from M and from $\overline{M} := E \setminus M$. An M, M-path is an M-alternating path whose first and last edge are from M; it must have odd length. Similarly, we define $\overline{M}, M$-paths, $M, \overline{M}$-paths (both of even length), and $\overline{M}, \overline{M}$-paths (of odd length). If M is a matching of G and v is incident with an edge of M then we use $M(v)$ to denote the other endpoint of that edge, i.e., the *mate* or *partner* of v. Say that a vertex v is *exposed by M* if it is not incident with an edge of M; we say that

v is *exposable* if it is exposed by some maximum matching of G. A graph $G = (V, E)$ is *factor-critical* if for each vertex $v \in V$ the graph $G - v$ has a perfect matching (a *near-perfect matching of* G); observe that all factor-critical graphs must have an odd number of vertices.

We denote by $LP(G)$ the optimum value of fractional vertex covers, which are exactly the feasible solutions of the well-known LP-relaxation of VERTEX COVER. It is known that the extremal points x of that linear program are half-integral, i.e., $x \in \{0, \frac{1}{2}, 1\}^V$. With this in mind, we will tacitly assume that all considered fractional vertex covers are half-integral. We use the shorthand $[n] := \{1, \ldots, n\}$. By $A \dot\cup B$ to denote the disjoint union of A and B.

Gallai-Edmonds decomposition. We will now recall the Gallai-Edmonds decomposition following the well-known book of Lovász and Plummer [18].[4]

▶ **Definition 2.** Let $G = (V, E)$ be a graph. The *Gallai-Edmonds decomposition* of G is a partition of V into three sets A, B, and D where

- D consists of all vertices v of G such that there is a maximum matching M of G that contains no edge incident with v, i.e., that leaves v exposed,
- A is the set of neighbors of D, i.e., $A := N(D)$, and
- B contains all remaining vertices, i.e., $B := V \setminus (A \cup D)$.

It is known (and easy to verify) that the Gallai-Edmonds decomposition of any graph G is unique and can be computed in polynomial time. The Gallai-Edmonds decomposition has a number of useful properties; the following theorem states some of them.

▶ **Theorem 3** (cf. [18, Theorem 3.2.1]). *Let $G = (V, E)$ be a graph and let $V = A \dot\cup B \dot\cup D$ be its Gallai-Edmonds decomposition. The following properties hold:*

1. *The connected components of $G[D]$ are factor-critical.*
2. *The graph $G[B]$ has a perfect matching.*
3. *Every maximum matching M of G consists of a perfect matching of $G[B]$, a near-perfect matching of each component of $G[D]$, and a matching of A into D.*

3 Tight vertex covers of factor-critical graphs

In this section we study vertex covers of factor-critical graphs, focusing on those that are of smallest possible size (later called tight vertex covers). We first recall the fact that any factor-critical graph with $n \geq 3$ vertices has no vertex cover of size less than $\frac{1}{2}(n+1)$. By a similar argument such graphs have no fractional vertex cover of cost less than $\frac{1}{2}n$.

▶ **Proposition 4** (folklore). *Let $G = (V, E)$ be a factor-critical graph with at least three vertices. Every vertex cover X of G has cardinality at least $\frac{1}{2}(|V|+1)$ and every fractional vertex cover $x\colon V \to \mathbb{R}_{\geq 0}$ of G has cost at least $\frac{1}{2}|V|$.*

Proof. Let $X \subseteq V$ be a vertex cover of G. Since G has at least three vertices and is factor-critical, it has a maximum matching M of size $\frac{1}{2}(|V|-1) \geq 1$. It follows that X has size at least one. (This is not true for graphs consisting of a single vertex, which are also factor-critical. All other factor-critical graphs have at least three vertices.) Pick any vertex $v \in X$. Since G is factor-critical, there is a maximum matching M_v of $G - v$ of size $\frac{1}{2}(|V|-1)$. It follows that X must contain at least one vertex from each edge of M_v, and no

[4] We use B instead of C for $V \setminus (A \cup D)$ to leave the letter C for cycles and connected components.

vertex is contained in two of them. Together with v, which is not in any edge of M_v, this gives a lower bound of $1 + \frac{1}{2}(|V| - 1) = \frac{1}{2}(|V| + 1)$, as claimed.

Let $x\colon V \to \mathbb{R}_{\geq 0}$ be a fractional vertex cover of G. We use again the matching M of size at least one from the previous case; let $\{u, v\} \in M$. It follows that $x(u) + x(v) \geq 1$; w.l.o.g. we have $x(v) \geq \frac{1}{2}$. Let M_v be a maximum matching of $G - v$ of size $\frac{1}{2}(|V| - 1)$. For each edge $\{p, q\} \in M_v$ we have $x(p) + x(q) \geq 1$. Since the matching edges are disjoint we get a lower bound of $\sum_{p \in V \setminus \{v\}} x(p) \geq \frac{1}{2}(|V| - 1)$. Together with $x(v) \geq \frac{1}{2}$ we get the claimed lower bound of $\frac{1}{2}|V|$ for the cost of x. ◀

Note that Proposition 4 is tight for example for all odd cycles of length at least three, all of which are factor-critical. We now define tight vertex covers and critical sets.

▶ **Definition 5** (tight vertex covers, critical sets)**.** Let $G = (V, E)$ be a factor-critical graph with $|V| \geq 3$. A vertex cover X of G is *tight* if $|X| = \frac{1}{2}(|V| + 1)$. Note that this is different from a minimum vertex cover, and a factor-critical graph need not have a tight vertex cover; e.g., odd cliques with at least five vertices are factor-critical but have no tight vertex cover.

A set $Z \subseteq V$ is called a *bad set* of G if there is no tight vertex cover of G that contains Z. The set Z is a *critical set* if it is a minimal bad set, i.e., no tight vertex cover of G contains Z but for all proper subsets Z' of Z there is a tight vertex cover containing Z'.

Observe that a factor-critical graph $G = (V, E)$ has no tight vertex cover if and only if $Z = \emptyset$ is a critical set of G. It may be interesting to note that a set $X \subseteq V$ of size $\frac{1}{2}(|V| + 1)$ is a vertex cover of G if and only if it contains no critical set. (We will not use this fact and hence leave its two line proof to the reader.) The following lemma proves that all critical sets of a factor-critical graph have size at most three; this is of central importance for our kernelization. For the special case of odd cycles, the lemma has a much shorter proof and we point out that all critical sets of odd cycles have size exactly three.

▶ **Lemma 6.** *Let $G = (V, E)$ be a factor-critical graph with at least three vertices. All critical sets Z of G have size at most three.*

Proof. Let $\ell \in \mathbb{N}$ with $\ell \geq 1$ such that $|V| = 2\ell + 1$; recall that all factor-critical graphs have an odd number of vertices.

Assume for contradiction that there is a critical set Z of G of size at least four. Let $w, x, y, z \in Z$ be any four pairwise different vertices from Z. Let M be a maximum matching of $G - w$. Since G is factor-critical, we get that M is a perfect matching of $G - w$ and has size $|M| = \ell$. Observe that any tight vertex cover of G that contains w must contain exactly one vertex from each edge of M, since its total size is $\frac{(}{|}V| + 1) = \ell + 1$. We will first analyze G and show that the presence of certain structures would imply that some proper subset Z' of Z is bad, contradicting the assumption that Z is critical. Afterwards, we will use the absence of these structures to find a tight vertex cover that contains Z, contradicting the fact that it is a critical set.

If there is an M, M-path from x to y then $\{w, x, y\}$ is a bad set, i.e., no tight vertex cover of G contains all three vertices w, x, and y, contradicting the choice of Z: Let $P = (v_1, v_2, \ldots, v_{p-1}, v_p)$ denote an M, M-path from $v_1 = x$ to $v_p = y$. Accordingly, we have $\{v_1, v_2\}, \ldots, \{v_{p-1}, v_p\} \in M$ and the path P has odd length. Assume that X is a tight vertex cover containing w, x, and y. It follows, since $w \in X$, that X contains exactly one vertex per edge in M; in particular it contains exactly one vertex per matching edge on the path P. Since $v_1 = x \in X$ we have $v_2 \notin X$. Thus, as $\{v_2, v_3\}$ is an edge of G, we must have $v_3 \in X$ to cover this edge; this in turn implies that $v_4 \notin X$ since it already contains v_3 from the matching edge $\{v_3, v_4\}$. Continuing this argument along the path P we conclude that $v_{p-1} \in X$ and $v_p \notin X$, contradicting the fact that $v_p = y \in X$. Thus, if there is an M, M-path from x to y then there is no tight vertex cover of G that contains w, x, and y, making $\{w, x, y\}$ a bad set and contradicting the assumption that Z is a critical set. It follows that there can be no M, M-path from x to y. The same argument can be applied also to x and z, and to y and z, ruling out M, M-paths connecting them.

Similarly, if there is an edge $\{u, v\} \in M$ such that z reaches both u and v by (different, not necessarily disjoint) $M, \overline{M}$-paths then no tight vertex cover of G contains both w and z, contradicting the choice of Z: Let $P = (v_1, v_2, \ldots, v_{p-1}, v_p)$ denote an $M, \overline{M}$-path from $v_1 = z$ to $v_p = u$ with $\{v_1, v_2\}, \{v_3, v_4\}, \ldots, \{v_{p-2}, v_{p-1}\} \in M$. Let X be a tight vertex cover of G that contains w and z. It follows (as above) that $v_1, v_3, \ldots, v_{p-2} \in X$ and $v_2, v_4, \ldots, v_{p-1} \notin X$, by considering the induced M, M-path from $z = v_1$ to v_{p-1}. The fact that $v_{p-1} \notin X$ directly implies that $v_p = u \in X$ in order to cover the edge $\{v_{p-1}, v_p\}$. Repeating the same argument on an $M, \overline{M}$-path from z to v we get that $v \in X$. Thus, we conclude that u and v are both in X, contradicting the fact that X must contain exactly one vertex of each edge in X. Hence, there is no tight vertex cover of G that contains both w and z. We conclude that $\{w, z\}$ is a bad set, contradicting the choice of Z. Hence, there is no edge $\{u, v\} \in M$ such that z has $M, \overline{M}$-paths (not necessarily disjoint) to both u and v.

Now we will complete the proof by using the established properties, i.e., the non-existence of certain M-alternating paths starting in z, to construct a tight vertex cover of G that contains all of Z, giving the final contradiction. Using minimality of Z, let X be a tight vertex cover of G that contains $Z \setminus \{z\}$; by choice of Z we have $z \notin X$. We construct the claimed vertex cover $X' \supseteq Z$ from $X' = X$ as follows:

1. Add vertex z to X and remove $M(z)$, i.e., remove the vertex that z is matched to.
2. Add all vertices v to X' that can be reached from z by an $M, \overline{M}$-path.
3. Remove all vertices from X' that can be reached from z by an M, M-path of length at least three. (There is a single such path of length one from z to $M(z)$ which, for clarity, was handled already above.)

We need to check four things: (1) The procedure above is well-defined, i.e., no vertex can be reached by both M, M- and $M, \overline{M}$-paths from z. (2) The size of X' is at most $|X| = \ell + 1$. (3) X' is a vertex cover. (4) The set X' contains w, x, y, and z.

(1) Assume that there is a vertex v such that z reaches v both by an M, M-path $P = (v_1, v_2, \ldots, v_p)$ with $v_1 = z$ and $v_p = v$, and by an $M, \overline{M}$-path P'. Observe that $\{v_{p-1}, v_p\} \in M$ since P is an M, M-path and, hence, that $P'' = (v_1, \ldots, v_{p-1})$ is an $M, \overline{M}$-path from v to v_{p-1}. Together, P' and P'' constitute two $M, \overline{M}$-paths from z to both endpoints v_{p-1} and v_p of the matching edge $\{v_{p-1}, v_p\}$; a contradiction (since we ruled out this case earlier).

(2) In the first step, we add z and remove $M(z)$. Note that $z \notin X$ implies that $M(z) \in X$ (we start with $X' = X$). Thus the size of X' does not change. Consider a vertex v that is added in the second step, i.e., with $v \notin X$: There is an $M, \overline{M}$-path P from z to v. Since $w \in X$ we know that $v \neq w$. Thus, since M is a perfect matching of $G - w$, there is a vertex u with $u = M(v)$. The vertex $u := M(v)$ must be in X to cover the edge $\{v, u\} \in M$, as $v \notin X$. Moreover, u cannot be on P since that would make it incident with a second matching edge other than $\{u, v\}$. Thus, by extending P with $\{v, u\}$ we get an M, M-path from z to u, implying that u is removed in the second step. Since $u \in X$ the total size change is zero. Observe that the vertex $u = M(v)$ used in this argument is not used for any other vertex v' added in the second step since it is only matched to v. Similarly, due to (1), the vertex u is not also added in the second step since it cannot be simultaneously have an $M, \overline{M}$-path from z.

(3) Assume for contradiction that some edge $\{u, v\}$ is not covered by X', i.e., that $u, v \notin X'$. Since $w \in X'$ is the only unmatched vertex it follows that both u and v are incident with some edge of M. We distinguish two cases, namely (a) $\{u, v\} \in M$ and (b) $\{u, v\} \notin M$.

(3.a) If $\{u, v\} \in M$ then without loss of generality assume $u \in X$ (as X is a vertex cover). By our assumption we have $u \notin X'$, which implies that we have removed it on account of having an M, M-path P from z to u. Since $\{u, v\} \in M$ the path P must visit v as its penultimate vertex; there is no other way for an M, M-path to reach u. This, however, implies that there is an $M, \overline{M}$-path from z to v, and that we have added v in the second step; a contradiction.

(3.b) In this case we have $\{u, v\} \notin M$. Again, without loss of generality, assume that $u \in X$. Since $u \notin X'$ there must be an M, M-path P from z to u. If P does not contain v then extending P by edge $\{u, v\} \notin M$ would give an $M, \overline{M}$-path from z to v and imply that $v \in X'$; a contradiction. In the remaining case, the vertex v is contained in P; let P' denote the induced path from z to v (not containing u as it is the final vertex of P). Since $v \notin X'$ we know that P' cannot be an $M, \overline{M}$-path, or else we would have $v \in X'$, and hence it must be an M, M-path. Now, however, extending P' via $\{v, u\} \notin M$ yields an $M, \overline{M}$-path from z to u, contradicting (1). Altogether, we conclude that X' is indeed a vertex cover.

(4) Clearly, $z \in X'$ by construction. Similarly, $w \in X'$ since it is contained in X and it cannot be removed since there is no incident M-edge (i.e., no M, M-paths from z can end in w). Finally, regarding x and y, we proved earlier that there are no M, M-paths from z to x or from z to y. Thus, since both x and y are in X they must also be contained in X'.

We have showed that under the assumption of minimality of Z and using $|Z| \geq 4$ one can construct a vertex cover X' of optimal size $\ell + 1$ that contains Z entirely. This contradicts the choice of Z and completes the proof. ◀

4 Nice decompositions

The well-known Gallai-Edmonds decomposition plays an important role in the FPT-algorithm of Garg and Philip [10]. It is, in principle, also very useful for our kernelization result, but it is much more convenient to use a form that is both relaxed (in part) but also includes a certain maximum matching of the graph, called *nice decomposition*. Due to space restrictions, we give the definition directly rather than first defining a natural intermediate form.

▶ **Definition 7** (nice decomposition)**.** Let $G = (V, E)$ be a graph. A *nice decomposition of* G is a tuple (A, B, D, M) where $V = A \,\dot\cup\, B \,\dot\cup\, D$ and M is a maximum matching of G such that

1. $A = N(D)$, i.e., all vertices not in D that are adjacent to D,
2. each connected component of $G[D]$ is factor-critical,
3. M restricted to B is a perfect matching of $G[B]$,
4. M restricted to any component C of $G[D]$ is a near-perfect matching of $G[C]$,
5. each vertex of A is matched by M to a vertex of D, and
6. for each singleton component $\{v\}$ of $G[D]$ there is a vertex $u \in A$ with $\{u, v\} \in M$.

For a nice decomposition (A, B, D, M) of G it will be of importance for us which components of $G[D]$ are matched to a vertex in A. Since M induces a near-perfect matching on each component of $G[D]$, there is always at most one such vertex per component of $G[D]$.

▶ **Definition 8** (matched/unmatched components of $G[D]$)**.** Let $G = (V, E)$ be a graph and let (A, B, D, M) be a nice decomposition of G. We say that a connected component C of $G[D]$ is *matched* if there are vertices $v \in C$ and $u \in N(C) \subseteq A$ such that $\{u, v\} \in M$; we will also say that u and C are matched to one another. Otherwise, we say that C is *unmatched*. Note that edges of M with both ends in C have no influence on whether C is matched or unmatched.

We use $\mathcal{C}_1$ to denote the set of matched singleton components in $G[D]$; a nice decomposition has no unmatched singleton components. We use $\mathcal{C}_3$ and $\hat{\mathcal{C}}_3$ for matched and unmatched non-singleton components. By A_1 and A_3 we denote the set of vertices in A that are matched to singleton respectively non-singleton components of $G[D]$; note that $A = A_1 \dot\cup A_3$.

Our main reason for preferring nice decompositions is captured in the following lemma, namely that deleting certain types of components, e.g., in a reduction rule, allows the obtained graph G' to effectively inherit a nice decomposition.

▶ **Lemma 9.** *Let $G = (V, E)$ be a graph, (A, B, D, M) a nice decomposition, and $C \in \hat{\mathcal{C}}_3$ an unmatched component of $G[D]$. Then (A, B, D', M') is a nice decomposition of $G' = G - C$ where M' is M restricted to $V(G') = V \setminus C$ and where $D' := D \setminus C$. The corresponding sets A_1, A_3, $\mathcal{C}_1$, and $\mathcal{C}_3$ are the same as for G. For $\hat{\mathcal{C}}_3$ we have $\hat{\mathcal{C}}'_3 = \hat{\mathcal{C}}_3 \setminus \{C\}$.*

5 Nice decompositions and vertex covers

Due to space restrictions, this section gives a brief summary of the relation between a nice decomposition (A, B, D, M) of a graph G and (certain) vertex covers of G. We first prove a lower bound on $VC(G)$ in terms of (A, B, D, M) and relate it to $2LP(G) - MM(G)$. Note that Garg and Philip [10] proved that $2LP(G) - MM(G)$ is a lower bound for the vertex cover size for every graph G, but we require the bound of $|M| + |\hat{\mathcal{C}}_3|$ related to our decompositions, and the equality to $2LP(G) - MM(G)$ is "only" required to complete the kernelization later.

▶ **Lemma 10.** *Let $G = (V, E)$ be a graph and let (A, B, D, M) be a nice decomposition of G. Each vertex cover of G has size at least $|M| + |\hat{\mathcal{C}}_3| = 2LP(G) - MM(G)$.*

Intuitively, a vertex cover X of size at most $|M| + |\hat{\mathcal{C}}_3| + \ell$ can "overpay" only ℓ times as compared to spending one vertex per edge of M and $\frac{1}{2}(|C| + 1)$ for any component $C \in \hat{\mathcal{C}}_3$. Conversely, X must induce tight vertex covers of all but at most ℓ components $C \in \hat{\mathcal{C}}_3$.

▶ **Definition 11** (active component)**.** Let $G = (V, E)$ be a graph, let (A, B, D, M) be a nice decomposition of G, and let X be a vertex cover of G. A component $C \in \hat{\mathcal{C}}_3$ is *active (w.r.t. X)* if X contains more than $\frac{1}{2}(|C| + 1)$ vertices of C, i.e., if $X \cap C$ is not a tight vertex cover of $G[C]$.

▶ **Definition 12** (set X_{op})**.** Let $G = (V, E)$ be a graph and let (A, B, D, M) be a nice decomposition of G. For $X \subseteq V$ define $X_{\mathsf{op}} = X_{\mathsf{op}}(A_1, A_3, M, X) \subseteq A \cap X$ to contain all vertices v that fulfill either of the following two conditions:

1. $v \in A_1$ and X contains both v and $M(v)$.
2. $v \in A_3$ and X contains v.

Both conditions of Definition 12 capture parts of the graph where X contains more vertices than implied by the lower bound. To see this for the second condition, note that if $v \in A_3 \cap X$ then X still needs at least $\frac{1}{2}(|C|+1)$ vertices of the component $C \in \mathcal{C}_3$ that v is matched to; since there are $\frac{1}{2}(|C|+1)$ matching edges that M has between vertices of $C \cup \{v\}$ we find that X (locally) exceeds the lower bound, as $|X \cap (C \cup \{v\})| \geq 1 + \frac{1}{2}(|C|+1)$. Conversely, if X does match the lower bound on $C \cup \{v\}$ then it cannot contain v.

We now prove formally that a vertex cover X of size close to the lower bound of Lemma 10 has only few active components and only a small set $X_{\mathsf{op}} \subseteq X$.

▶ **Lemma 13.** *Let $G = (V, E)$ be a graph, let (A, B, D, M) be a nice decomposition of G, let X be a vertex cover of G, and let $X_{\mathsf{op}} = X_{\mathsf{op}}(A_1, A_3, M, X)$. The set X_{op} has size at most ℓ and there are at most ℓ active components in $\hat{\mathcal{C}}_3$ with respect to X where $\ell = |X| - (|M| + |\hat{\mathcal{C}}_3|) = |X| - (2LP(G) - MM(G))$.*

The central question is of course how the different structures where X exceeds the lower bound interact. We are only interested in aspects that are responsible for not allowing a tight vertex cover for any (unmatched, non-singleton) components $C \in \hat{\mathcal{C}}_3$. This happens exactly due to vertices in A that are adjacent to C and that are not selected by X. Between components of $G[B]$ and non-singleton components of $G[D]$ there are M-alternating paths with vertices alternatingly from A and from singleton components of $G[D]$ since vertices in A are all matched to D and singleton components in $G[D]$ have all their neighbors in A. Unless X contains both vertices of a matching edge, it contains the A- or the D-vertices of such a path. Unmatched components of $G[D]$ and components of $G[B]$ have all neighbors in A. Matched components C in $G[D]$ with matched neighbor $v \in A$ enforce not selecting v for X unless X spends more than the lower bound; in this way, they lead to selection of D-vertices on M-alternating paths. Intuitively, this leads to two "factions" that favor either A- or D-vertices and that are effectively separated when X selects both A- and D-endpoint of a matching edge. An optimal solution need not separate all neighbors in A of any component $C \in \hat{\mathcal{C}}_3$, and C may still have a tight vertex cover or paying for a larger cover of C is overall beneficial. The following auxiliary directed graph H captures this situation and for certain vertex covers X reachability of $v \in A$ in $H - X_{\mathsf{op}}$ will be proved to be equivalent with $v \notin X$.

▶ **Definition 14** (auxiliary directed graph H). Let $G = (V, E)$ be a graph and let (A, B, D, M) be a nice decomposition of G. Define a *directed graph* $H = H(G, A, B, D, M)$ on vertex set A by letting (u, v) be a directed edge of H, for $u, v \in A$, whenever there is a vertex $w \in D$ with $\{u, w\} \in E \setminus M$ and $\{w, v\} \in M$.

There are three important properties of vertex covers in relation to the corresponding graph H. Two of them hold only for what we call *dominant* vertex covers and their proofs build on a fairly technical replacement argument. We will define dominant vertex covers next and then summarize the properties in a single lemma.

▶ **Definition 15** (dominant vertex cover). Let $G = (V, E)$ be a graph and let (A, B, D, M) be a nice decomposition of G. A vertex cover $X \subseteq V$ of G is *dominant* if G has no vertex cover of size less than $|X|$ and no vertex cover of size $|X|$ contains fewer vertices of D.

▶ **Lemma 16.** *Let $G = (V, E)$ be a graph, let (A, B, D, M) be a nice decomposition of G, and X a vertex cover of G. Let $H = H(G, A, B, D, M)$ and let $X_{\mathsf{op}} = X_{\mathsf{op}}(A_1, A_3, M, X)$. The following properties hold:*

1. *If X is dominant then X_{op} is closest to A_3 in H.*
2. *If $v \in A$ is reachable from A_3 in $H - X_{\mathsf{op}}$ then X does not contain v.*
3. *If X is dominant and $v \in A$ is not reachable from A_3 in $H - X_{\mathsf{op}}$ then X contains v.*

All three properties are crucial for applying the matroid tools of Kratsch and Wahlström [16]. Closeness of X_{op} is needed to translate between reachability and independence in an appropriate matroid. The latter two properties are required to translate between the directed graph, where the tools are applied, and the undirected input graph G.

6 Randomized polynomial kernelization

In this section, we describe our randomized polynomial kernelization for VERTEX COVER($k-(2\text{LP}-\text{MM})$). For convenience, let us fix an input instance (G,k,ℓ), i.e., $G=(V,E)$ is a graph for which we want to know whether it has a vertex cover of size at most k; the parameter is $\ell = k-(2LP(G)-MM(G))$, where $LP(G)$ is the minimum cost of a fractional vertex cover of G and $MM(G)$ is the size of a largest matching.

From previous work of Garg and Philip [10] we know that the well-known linear program-based preprocessing for VERTEX COVER (cf. [5]) can also be applied to VERTEX COVER($k-(2\text{LP}-\text{MM})$); the crucial new aspect is that this operation does not increase the value $k-(2LP-MM)$. The LP-based preprocessing builds on the half-integrality of fractional vertex covers and a result of Nemhauser and Trotter [22] stating that all vertices with value 1 and 0 in an optimal fractional vertex cover $x\colon V\to\{0,\frac{1}{2},1\}$ are included respectively excluded in at least one minimum (integral) vertex cover. Thus, only vertices with value $x(v)=\frac{1}{2}$ remain and the best LP solution costs exactly $\frac{1}{2}$ times number of (remaining) vertices. For our kernelization we only require the fact that if G is reduced under this reduction rule then $LP(G)=\frac{1}{2}(|V(G)|)$; e.g., we do not require $x\colon V\to\{\frac{1}{2}\}$ to be the unique optimal fractional vertex cover. Without loss of generality, we assume that our given graph $G=(V,E)$ already fulfills $LP(G)=\frac{1}{2}|V|$.

▶ **Observation 17.** *If $LP(G)=\frac{1}{2}|V|$ then $2LP(G)-MM(G)=|V|-MM(G)$. In other words, if M is a maximum matching of G then the lower bound $2LP(G)-MM(G)=|V|-MM(G)=|V|-|M|$ is equal to cardinality of M plus the number of isolated vertices.*

As a first step, let us compute the Gallai-Edmonds decomposition $V=A\,\dot\cup\,B\,\dot\cup\,D$ of G; this can be done in polynomial time.[5] Using $LP(G)=\frac{1}{2}|V|$ we can find a maximum matching M of G such that (A,B,D,M) is a nice decomposition of G.

▶ **Lemma 18.** *Given $G=(V,E)$ with $LP(G)=\frac{1}{2}|V|$ and a Gallai-Edmonds decomposition $V=A\,\dot\cup\,B\,\dot\cup\,D$ of G one can in polynomial time compute a maximum matching M of G such that (A,B,D,M) is a nice decomposition of G.*

We fix a nice decomposition (A,B,D,M) of G obtained via Lemma 18. We have already learned about the relation of dominant vertex covers X, their intersection with the set A, and separation of A vertices from A_3 in $H-X_{\mathsf{op}}$, where $H=H(G,A,B,D,M)$. It is safe to assume that solutions are dominant vertex covers as among minimum vertex covers there is a minimum intersection with D. We would now like to establish that most components of $\hat{\mathcal{C}}_3$ can be deleted (while reducing k by the cost for corresponding tight vertex covers). Clearly, since any vertex cover pays at least for tight covers of these components, we cannot turn a yes- into a no-instance this way. However, if the instance is no then it might become yes.

[5] The main expenditure is finding the set D. A straightforward approach is to compute a maximum matching M_v of $G-v$ for each $v\in V$. If $|M_v|=MM(G)$ then v is in D as M_v is maximum and exposes v; otherwise $v\notin D$ as no maximum matching exposes v.

In the following, we will try to motivate both the selection process for components of $\hat{\mathcal{C}}_3$ that are deleted as well as the high-level proof strategy for establishing correctness. We will tacitly ignore most technical details, like parameter values, getting appropriate nice decompositions, etc., and refer to the formal proof instead. Assume that we are holding a no-instance (G, k, ℓ). Consider for the moment, the effect of deleting all components $C \in \hat{\mathcal{C}}_3$ that have tight vertex covers and updating the budget accordingly; for simplicity, say they all have such vertex covers. Let (G_0, k_0, ℓ) be the obtained instance; if this instance is no as well, then deleting any subset of $\hat{\mathcal{C}}_3$ also preserves the correct answer (namely: no). Else, if (G_0, k_0, ℓ) is yes then pick any dominant vertex cover X^0 for it. We could attempt to construct a vertex cover of G of size at most k by adding back the components of C and picking a tight vertex cover for each; crucially, these covers must also handle edges between C and A. Since (G, k, ℓ) was assumed to be a no-instance, there must be too many components $C \in \hat{\mathcal{C}}_3$ for which this approach fails. For any such component, the adjacent vertices in $A \setminus X^0$ force a selection of their neighbors $Z_A = N(A) \cap C$ that cannot be completed to a tight vertex cover of C. To avoid turning the no-instance (G, k, ℓ) into a yes-instance (G', k', ℓ) we have to keep enough components of $\hat{\mathcal{C}}_3$ in order to falsify any suggested solution X' of size at most k' for G. The crux is that there may be an exponential number of such solutions and that we do not know any of them. This is where the auxiliary directed graph and related technical lemmas as well as the matroid-based tools of Kratsch and Wahlström [16] are essential.

Let us outline how we arrive at an application of the matroid-based tools. Crucially, if C (as above) has no tight vertex cover containing $Z_A = N(A) \cap C$ then, by Lemma 6, there is a set $Z \subseteq Z_A$ of size at most three such that no tight vertex cover contains Z. Accordingly, there is a set $T \subseteq A \setminus X^0$ of size at most three whose neighborhood in C contains Z. Thus, the fact that X^0 contains no vertex of T is responsible for not allowing a tight vertex cover of C. This in turn, by Lemma 16 means that all vertices in T are reachable from A_3 in $H - X^0_{\mathsf{op}}$. Recalling that a set X^0_{op} corresponding to a dominant vertex cover is also closest to A_3, we can apply a result from [16] that generates a sufficiently small representative set of sets T corresponding to components of $\hat{\mathcal{C}}_3$. If a dominant vertex cover has any reachable sets T then the lemma below guarantees that at least one such set is in the output. For each set we select a corresponding component $C \in \hat{\mathcal{C}}_3$ and then start over on the remaining components. After $\ell + 1$ iterations we can prove that for any not selected component C, which we delete, and any proposed solution X' for the resulting graph that does not allow a tight vertex cover for C, there are $\ell + 1$ other selected components on which X' cannot be tight. This is a contradiction as there are at most ℓ such active components by Lemma 13.

Concretely, we will use the following lemma about representative sets of vertex sets of size at most three regarding reachability in a directed graph (modulo deleting a small set of vertices). Notation of the lemma is adapted to the present application. The original result is for pairs of vertices in a directed graph (see [15, Lemma 2]) but extends straightforwardly to sets of fixed size q and to sets of size at most q. We provide a proof in the full version for completeness. Note that the lemma is purely about reachability of small sets in a directed graph (like the DIGRAPH PAIR CUT problem studied in [15, 16]) and we require the structural lemmas proved so far to negotiate between this an VERTEX COVER$(k - (2\text{LP} - \text{MM}))$.

▶ **Lemma 19.** *Let $H = (V_H, E_H)$ be a directed graph, let $S_H \subseteq V_H$, let $\ell \in \mathbb{N}$, and let $\mathcal{T}$ be a family of nonempty vertex sets $T \subseteq V_H$ each of size at most three. In randomized polynomial time, with failure probability exponentially small in the input size, we can find a set $\mathcal{T}^* \subseteq \mathcal{T}$ of size $\mathcal{O}(\ell^3)$ such that for any set $X_H \subseteq V_H$ of size at most ℓ that is closest to S_H if there is a set $T \in \mathcal{T}$ such that all vertices $v \in T$ are reachable from S_H in $H - X_H$ then there is a corresponding set $T^* \in \mathcal{T}^*$ satisfying the same properties.*

Using the lemma we will be able to identify a small set $\mathcal{C}_{\texttt{rel}}$ of components of $\hat{\mathcal{C}}_3$ that contains for each dominant vertex cover X of G of size at most k all active components with respect to X. Conversely, if there is no solution of size k, we will have retained enough components of $\hat{\mathcal{C}}_3$ to preserve this fact. Concretely, the set $\mathcal{C}_{\texttt{rel}}$ is computed as follows:

1. Let $\mathcal{C}^0_{\texttt{rel}}$ contain all components $C \in \hat{\mathcal{C}}_3$ that have no vertex cover of size at most $\frac{1}{2}(|C|+1)$. Clearly, these components are active for every vertex cover of G. We know from Lemma 13 that there are at most ℓ such components if the instance is **yes**. We can use the algorithm of Garg and Philip [10] to test in polynomial time whether any $C \in \hat{\mathcal{C}}_3$ has a vertex cover of size at most $k_C := \frac{1}{2}(|C|+1)$: We have parameter value
$$k_C - (2LP(G[C]) - MM(G[C])) = \frac{1}{2}(|C|+1) - (|C| - \frac{1}{2}(|C|-1)) = 0.$$
We could of course also use an algorithm for VERTEX COVER parameterized above maximum matching size, where we would have parameter value 1. If there are more than ℓ components C with no vertex cover of size $\frac{1}{2}(|C|+1)$ then we can safely reject the instance. Else, as indicated above, let $\mathcal{C}^0_{\texttt{rel}}$ contain all these components and continue.
2. Let $i = 1$. We will repeat the following steps for $i \in \{1, \ldots, \ell+1\}$.
3. Let $\mathcal{T}^i$ contain all nonempty sets $T \subseteq A$ of size at most three such that there is a component $C \in \hat{\mathcal{C}}_3 \setminus (\mathcal{C}^0_{\texttt{rel}} \cup \ldots \cup \mathcal{C}^{i-1}_{\texttt{rel}})$ such that:
   a. There is a set $Z \subseteq N_G(T) \cap C$ of at most three neighbors of T in C such that no vertex cover of $G[C]$ of size $\frac{1}{2}(|C|+1)$ contains Z. Note that $Z \neq \emptyset$ since $C \notin \mathcal{C}^0_{\texttt{rel}}$ implies that it has at least some vertex cover of size $\frac{1}{2}(|C|+1)$.
   b. For each C and $Z \subseteq C$ of size at most three, existence of a vertex cover of $G[C]$ of size $k_C := \frac{1}{2}(|C|+1)$ containing Z can be tested by the algorithm of Garg and Philip [10] since the parameter value is constant. Concretely, run the algorithm on $G[C \setminus Z]$ and solution size $k_C - |Z|$ and observe that the parameter value is
   $$(k_C - |Z|) - (2LP(G[C \setminus Z]) - MM(G[C \setminus Z])).$$
   Using that $LP(G[C \setminus Z]) \geq LP(G[C]) - |Z|$ and $MM(G[C \setminus Z]) \leq MM(G[C]) = \frac{1}{2}(|C|-1)$ this value can be upper bounded by
   $$\begin{aligned} & k_C - |Z| - 2LP(G[C]) + 2|Z| + MM(G[C]) \\ = & \frac{1}{2}(|C|+1) - |Z| - |C| + 2|Z| + \frac{1}{2}(|C|-1) = |Z|. \end{aligned}$$
   Since $|Z| \leq 3$ the parameter value is at most three and the FPT-algorithm of Garg and Philip [10] runs in polynomial time.

   Intuitively, C must always be active for vertex covers not containing T, but for the formal correctness proof that we give later the above description is more convenient.
4. Apply Lemma 19 to graph $H = H(G, A, B, D, M)$ on vertex set $V_H = A$, set $S_H = A_3 \subseteq A$, integer ℓ, and family $\mathcal{T}^i$ of nonempty subsets of A of size at most three to compute a subset $\mathcal{T}^{i*}$ of $\mathcal{T}^i$ in randomized polynomial time. The size of $|\mathcal{T}^{i*}|$ is $\mathcal{O}(\ell^3)$.
5. Select a set $\mathcal{C}^i_{\texttt{rel}}$ as follows: For each $T \in \mathcal{T}^{i*}$ add to $\mathcal{C}^i_{\texttt{rel}}$ a component $C \in \hat{\mathcal{C}}_3 \setminus (\mathcal{C}^0_{\texttt{rel}} \cup \ldots \cup \mathcal{C}^{i-1}_{\texttt{rel}})$ such that C fulfills the condition for T in Step 3, i.e., such that:
   a. There is a set $Z \subseteq N_G(T) \cap C$ of at most three neighbors of T in C such that no vertex cover of $G[C]$ of size $\frac{1}{2}(|C|+1)$ contains Z. (We know that Z must be nonempty.)

   Clearly, the size of $|\mathcal{C}^i_{\texttt{rel}}|$ is $\mathcal{O}(\ell^3)$. Note that the same component C can be chosen for multiple sets $T \in \mathcal{T}^{i*}$ but we only require an upper bound on $|\mathcal{C}^i_{\texttt{rel}}|$

6. If $i < \ell+1$ then increase i by one and return to Step 3. Else return the set $\mathcal{C}_{\texttt{rel}} := \bigcup_{i=0}^{\ell+1} \mathcal{C}^i_{\texttt{rel}}$. The size of $\mathcal{C}_{\texttt{rel}}$ is $\mathcal{O}(\ell^4)$ since it is the union of $\ell+2$ sets that are each of size $\mathcal{O}(\ell^3)$.

In particular, we will be interested in the components $C \in \hat{\mathcal{C}}_3$ that are not in $\mathcal{C}_{\texttt{rel}}$. We call these *irrelevant components* and let $\mathcal{C}_{\texttt{irr}} := \hat{\mathcal{C}}_3 \setminus \mathcal{C}_{\texttt{rel}}$ denote the set of all irrelevant components. (Of course we still need to prove that they are true to their name.)

▶ **Lemma 20.** *Let G' be obtained by deleting from G all vertices of irrelevant components, i.e., $G' := G - \bigcup_{C\in\mathcal{C}_{\texttt{irr}}} C$, and let $k' = k - \sum_{C\in\mathcal{C}_{\texttt{irr}}} \frac{1}{2}(|C|+1)$, i.e., k' is equal to k minus the lower bounds for vertex covers of the irrelevant components. Then G has a vertex cover of size at most k if and only if G' has a vertex cover of size at most k'. Moreover, $k - (2LP(G) - MM(G)) = k' - (2LP(G') - MM(G'))$, i.e., the instances (G, k, ℓ) and (G', k, ℓ') of* VERTEX COVER$(k - (2\text{LP} - \text{MM}))$ *have the same parameter value $\ell = \ell'$.*

We can now complete our kernelization. According to Lemma 20 we may delete all irrelevant components and update k. We obtain a graph G' and integer k' such that:

1. G' has a vertex cover of size at most k' if and only if G has a vertex cover of size at most k, i.e., the instances (G, k) and (G', k') for VERTEX COVER are equivalent.
2. As a part of the proof of Lemma 20 we showed that $k' = |M'| + |\hat{\mathcal{C}}'_3| + \ell$ where $\hat{\mathcal{C}}'_3$ is the set of unmatched non-singleton components of $G'[D']$ with respect to M'.
3. From Lemma 9 we know that $\hat{\mathcal{C}}'_3$ is equal to the set $\hat{\mathcal{C}}_3$ minus the components $C \in \mathcal{C}_{\texttt{irr}}$ that were removed to obtain G'. In other words, $\hat{\mathcal{C}}'_3 = \hat{\mathcal{C}}_3 \setminus \mathcal{C}_{\texttt{irr}} = \mathcal{C}_{\texttt{rel}}$.
4. We know from Step 6 that $|\mathcal{C}_{\texttt{rel}}| = \mathcal{O}(\ell^4)$. Hence, $|\hat{\mathcal{C}}'_3| = \mathcal{O}(\ell^4)$.
5. Let us consider $p := k' - |M'|$, which is the parameter value of (G', k') when considered as an instance of VERTEX COVER parameterized above the size of a maximum matching. Clearly, $p = k' - |M'| = |M'| + |\hat{\mathcal{C}}'_3| + \ell - |M'| = \ell + \mathcal{O}(\ell^4) = \mathcal{O}(\ell^4)$.
6. We can now apply the randomized polynomial kernelization for VERTEX COVER$(k - \text{MM})$ [16] to get a polynomial kernelization for VERTEX COVER$(k - (2\text{LP} - \text{MM}))$. On input of (G', k', p) it returns an equivalent instance (G^*, k^*, p^*) of size $\mathcal{O}(p^c)$ for some constant c. We may assume that $k^* = \mathcal{O}(p^c)$ since else it would exceed the number of vertices in G^* and we may as well return a **yes**-instance of constant size. Let $\ell^* = k^* - (2LP(G^*) - MM(G^*))$, i.e., the parameter value of the instance (G^*, k^*, ℓ^*) of VERTEX COVER$(k - (2\text{LP} - \text{MM}))$. Clearly, $\ell^* \leq k^* = \mathcal{O}(p^c)$. Thus, (G^*, k^*, ℓ^*) has size and parameter value $\mathcal{O}(p^c)$.

▶ **Theorem 21.** VERTEX COVER$(k - (2\text{LP} - \text{MM}))$ *has a randomized polynomial kernelization with error probability exponentially small in the input size.*

7 Conclusion

We have presented a randomized polynomial kernelization for VERTEX COVER$(k - (2\text{LP} - \text{MM}))$ by giving a (randomized) polynomial parameter transformation to VERTEX COVER$(k - \text{MM})$. This improves upon the smallest parameter, namely $k - LP(G)$, for which such a result was known [16]. The kernelization for VERTEX COVER$(k - \text{MM})$ [16] involves reductions to and from ALMOST 2-SAT(k), which can be done without affecting the parameter value (cf. [23]). We have not attempted to optimize the total size. Given an instance (G, k, ℓ) for VERTEX COVER(k-(2LP-MM)) we get an equivalent instance of ALMOST 2-SAT(k) with $\mathcal{O}(k^{24})$ variables and size $\mathcal{O}(k^{48})$, which still needs to be reduced to a VERTEX COVER instance.

It seems likely that the kernelization can be improved if one avoids the blackbox use of the kernelization for VERTEX COVER$(k - \text{MM})$ and the detour via ALMOST 2-SAT(k). In particular, the underlying kernelization for ALMOST 2-SAT(k) applies, in part, the same representative set machinery to reduce the number of a certain type of clauses. Conceivably the two applications can be merged, thus avoiding the double blow-up in size. As a caveat, it appears to be likely that this would require a much more obscure translation into a directed separation problem. Moreover, the kernelization for ALMOST 2-SAT(k) requires an approximate solution, and it is likely that the same would be true for this approach. It would of course also be interesting whether a deterministic polynomial kernelization is possible, but this is, e.g., already not known for ALMOST 2-SAT(k) and VERTEX COVER($k-\text{MM}$).

We find the appearance of a notion of critical sets of size at most three and the derived separation problem in the auxiliary directed graph quite curious. For the related problem of separating at least one vertex from each of a given set of triples from some source s by deleting at most ℓ vertices (a variant of DIGRAPH PAIRCUT [16]) there is a natural $\mathcal{O}^*(3^\ell)$ time algorithm that performs at most ℓ three-way branchings before finding a solution (if possible). It would be interesting whether a complete encoding of VERTEX COVER($k-(2\text{LP}-\text{MM})$) into a similar form would be possible, since that would imply an algorithm that exactly matches the running time of the algorithm of the algorithm by Garg and Philip [10].

References

1 Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009. doi:10.1016/j.jcss.2009.04.001.

2 Hans L. Bodlaender, Stéphan Thomassé, and Anders Yeo. Kernel bounds for disjoint cycles and disjoint paths. *Theor. Comput. Sci.*, 412(35):4570–4578, 2011. doi:10.1016/j.tcs.2011.04.039.

3 Jianer Chen, Iyad A. Kanj, and Weijia Jia. Vertex cover: Further observations and further improvements. *J. Algorithms*, 41(2):280–301, 2001. doi:10.1006/jagm.2001.1186.

4 Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theor. Comput. Sci.*, 411(40-42):3736–3756, 2010. doi:10.1016/j.tcs.2010.06.026.

5 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.

6 Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. On the hardness of losing width. *Theory Comput. Syst.*, 54(1):73–82, 2014. doi:10.1007/s00224-013-9480-1.

7 Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. On multiway cut parameterized above lower bounds. *TOCT*, 5(1):3, 2013. doi:10.1145/2462896.2462899.

8 Holger Dell and Dieter van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. *J. ACM*, 61(4):23:1–23:27, 2014. doi:10.1145/2629620.

9 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

10 Shivam Garg and Geevarghese Philip. Raising the bar for vertex cover: Fixed-parameter tractability above a higher guarantee. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1152–1166. SIAM, 2016. doi:10.1137/1.9781611974331.ch80.

11 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

12 Bart M. P. Jansen. *The power of data reduction: Kernels for fundamental graph problems*. PhD thesis, Utrecht University, 2013.

13 Bart M. P. Jansen and Hans L. Bodlaender. Vertex cover kernelization revisited - upper and lower bounds for a refined parameter. *Theory Comput. Syst.*, 53(2):263–299, 2013. `doi:10.1007/s00224-012-9393-4`.

14 Stefan Kratsch. Recent developments in kernelization: A survey. *Bulletin of the EATCS*, 113, 2014. URL: `http://eatcs.org/beatcs/index.php/beatcs/article/view/285`.

15 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. *CoRR*, abs/1111.2195, 2011. URL: `http://arxiv.org/abs/1111.2195`.

16 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 450–459. IEEE Computer Society, 2012. `doi:10.1109/FOCS.2012.46`.

17 Daniel Lokshtanov, N. S. Narayanaswamy, Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. Faster parameterized algorithms using linear programming. *ACM Transactions on Algorithms*, 11(2):15:1–15:31, 2014. `doi:10.1145/2566616`.

18 László Lovász and Michael D. Plummer. *Matching Theory*. North-Holland, 1986.

19 Meena Mahajan and Venkatesh Raman. Parameterizing above guaranteed values: Maxsat and maxcut. *J. Algorithms*, 31(2):335–354, 1999. `doi:10.1006/jagm.1998.0996`.

20 Sounaka Mishra, Venkatesh Raman, Saket Saurabh, Somnath Sikdar, and C. R. Subramanian. The complexity of König subgraph problems and above-guarantee vertex cover. *Algorithmica*, 61(4):857–881, 2011. `doi:10.1007/s00453-010-9412-2`.

21 N. S. Narayanaswamy, Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. LP can be a cure for parameterized problems. In Christoph Dürr and Thomas Wilke, editors, *29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012, February 29th - March 3rd, 2012, Paris, France*, volume 14 of *LIPIcs*, pages 338–349. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012. `doi:10.4230/LIPIcs.STACS.2012.338`.

22 George L. Nemhauser and Leslie E. Trotter Jr. Vertex packings: Structural properties and algorithms. *Math. Program.*, 8(1):232–248, 1975. `doi:10.1007/BF01580444`.

23 Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. Paths, flowers and vertex cover. In Camil Demetrescu and Magnús M. Halldórsson, editors, *Algorithms - ESA 2011 - 19th Annual European Symposium, Saarbrücken, Germany, September 5-9, 2011. Proceedings*, volume 6942 of *Lecture Notes in Computer Science*, pages 382–393. Springer, 2011. `doi:10.1007/978-3-642-23719-5_33`.

24 Igor Razgon and Barry O'Sullivan. Almost 2-sat is fixed-parameter tractable. *J. Comput. Syst. Sci.*, 75(8):435–450, 2009. `doi:10.1016/j.jcss.2009.04.002`.

The Strongly Stable Roommates Problem

Adam Kunysz[*]

Institute of Computer Science, University of Wrocław, Wrocław, Poland
aku@cs.uni.wroc.pl

Abstract

An instance of the strongly stable roommates problem with incomplete lists and ties (SRTI) is an undirected non-bipartite graph $G = (V, E)$, with an adjacency list being a linearly ordered list of ties, which are vertices equally good for a given vertex. Ties are disjoint and may contain one vertex. A matching M is a set of vertex-disjoint edges. An edge $\{x, y\} \in E \setminus M$ is a *blocking edge* for M if x is either unmatched or strictly prefers y to its current partner in M, and y is either unmatched or strictly prefers x to its current partner in M or is indifferent between them. A matching is *strongly stable* if there is no blocking edge with respect to it. We present an $O(nm)$ time algorithm for computing a strongly stable matching, where we denote $n = |V|$ and $m = |E|$. The best previously known solution had running time $O(m^2)$ [16]. We also give a characterisation of the set of all strongly stable matchings. We show that there exists a partial order with $O(m)$ elements representing the set of all strongly stable matchings, and we give an $O(nm)$ algorithm for constructing such a representation. Our algorithms are based on a simple reduction to the bipartite version of the problem.

1998 ACM Subject Classification G.2.2 [Discrete Mathematics] Graph Theory, Graph Algorithms

Keywords and phrases strongly stable matching, stable roommates, rotations, matching theory

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.60

1 Introduction

An instance of the STABLE ROOMMATES PROBLEM WITH TIES AND INCOMPLETE LISTS (SRTI) involves a non-bipartite graph $G = (V, E)$, where an adjacency list of each vertex is a linearly ordered list of ties, which are subsets of vertices equally good for a given vertex. Ties are disjoint and may contain one vertex. Thus if vertices b_1 and b_2 are neighbours of a in G then one of the following holds:

- a strictly prefers b_1 to b_2, which we denote by $b_1 \succ_a b_2$
- b_1 and b_2 are tied on the preference list of a, which we denote by $b_1 =_a b_2$
- a strictly prefers b_2 to b_1, which we denote by $b_1 \prec_a b_2$

If vertex a strictly prefers b_1 to b_2 or is indifferent between them, then we say that a *weakly prefers* b_1 to b_2 and we denote it by $b_1 \succeq_a b_2$. A matching M is a set of edges, no two of which share an endpoint. Let $e = (v, w)$ be an edge contained in a matching M. Then we say that vertices v and w are matched in M and that v is the partner of w in M, which we also denote as $v = M(w)$. If a vertex v has no edge of M incident to it, then we say that v is *free* or *unmatched* in M. An edge $(x, y) \in E \setminus M$ is a *blocking edge* for M if x is either unmatched or strictly prefers y to its current partner in M, and y is either unmatched or weakly prefers x to its current partner. A matching is *strongly stable* if there is no edge blocking it. The goal is to determine a strongly stable matching of a given instance or to

[*] Partly supported by Polish National Science Center grant UMO-2013/11/B/ST6/01748.

report that no such matching exists. The STABLE MARRIAGE PROBLEM WITH TIES AND INCOMPLETE LISTS (SMTI) is a version of the problem, such that the underlying graph G is bipartite.

Motivation. As the problem name suggests, applications of SRTI arise in the context of assigning students to dormitories [13], [14]. In the problem we try to assign students to share two-person rooms. An instance of SRTI arises in a natural way based on students preferences over one another. The notion of strong stability allows us to prevent the following scenarios. Suppose that we assign a student a to share a room with a student $M(a)$ and a student b to share a room with a student $M(b)$. Assume also that a and b accept each other as a potential rommmates and that a prefers b to $M(a)$ but b is indifferent between a and $M(b)$. Then to improve their situation student a may try to bribe student b, in order to convince b to accept them. Since b is indifferent between a and $M(b)$, they may be happy to share a room with a, denying our assignment.

Previous results. Several algorithms for computing a strongly stable matching in bipartite instances of SMTI have already been given. Let us denote $n = |V|$, $m = |E|$. Irving [5] gave an $O(n^4)$ algorithm for computing strongly stable matchings for instances of SMTI in which the graph is complete. In [9] Manlove extended the algorithm to general SMTI instances, obtaining $O(m^2)$ runtime. In [7] Kavitha, Mehlhorn, Michail and Paluch gave an $O(nm)$ algorithm for the problem. Several structural results related to the problem in instances of SMTI have been given. In [10] the set of strongly stable matchings has been shown to form a distributive lattice (defined in Preliminaries). Recently, in [8] Ghosal, Kunysz and Paluch characterised the set of strongly stable matchings. They described an $O(nm)$ algorithm for constructing a partial order with $O(m)$ elements representing the set of solutions to the problem.

Contrary to the bipartite version of the problem, its non-bipartite generalisation has not received much attention in the literature. The problem of computing a strongly stable matching in non-bipartite instances of SRTI was first solved by Scott [16]. He obtained an $O(m^2)$ algorithm for the problem (The algorithm contained some flaws that can be removed using results from this paper). To the best of our knowledge no structural results related to the problem have been published so far.

Our results. Scott [16] and Manlove [11] asked whether it was possible to use techniques from [7] in order to speed up Scott's algorithm for computing a single strongly stable matching in instances of SRTI from $O(m^2)$ time to $O(nm)$ time. We describe an $O(nm)$ time algorithm for the problem, however we would like to remark that our algorithm is not an extension of Scott's algorithm. Our approach is based on a simple reduction to the bipartite version of the problem. Let $\mathcal{I}$ be an instance of SRTI, and $G = (V, E)$ be an underlying graph. We define an auxiliary instance $\mathcal{I}'$ of SMTI along with its underlying graph $G' = (A \cup B, E')$ as follows. We make two copies $v^p \in A$ and $v^r \in B$ of each vertex $v \in V$. For each edge $\{v, w\} \in E$ we add (v^p, w^r) and (w^p, v^r) to E'. Preference lists in $\mathcal{I}'$ are inherited from preference lists in $\mathcal{I}$. Most of the strongly stable matchings in G' correspond to certain cycles in G, however a deep understanding of the structure of the bipartite instance $\mathcal{I}'$ allows us to filter out matchings which do not correspond to strongly stable matchings in G. This approach allows us not only to obtain a faster algorithm for computing a single strongly stable matching, but also to characterise the set of all strongly stable matchings in instances of SRTI. Our characterisation is based on the construction of a certain partial order with

$O(m)$ elements which allows us to represent all the strongly stable matchings. No such characterisation has been known so far in this setting, however we would like to remark that our construction resembles the one given by Gusfield and Irving [3] for instances of SRI. The presented characterisation can be used to solve a number of problems connected with strongly stable matchings such as enumeration of strongly stable matchings, the minimum regret matching problem and the problem of computing all strongly stable pairs. We would like to point out that we do not address these problems in the paper. The main advantage of our approach is its simplicity. Due to the complicated nature of the problem, it would require a lot of effort to extend Scott's algorithm in order to achieve an $O(nm)$ running time for finding a single strongly stable matching and construct a representation of all the strongly stable matchings. Our algorithms completely avoid the need for low level technical details. The reduction to the bipartite version of the problem allows us also to construct an alternative version of Irving's algorithm [4] for computing stable matchings in instances of SRI. We remark that this has already been observed by Dean and Munshi in [1], where they also use the bipartite formulation of the problem to obtain their results.

Related work. Depending on the way we define a blocking edge in an instance of SRTI we can get two other versions of the stable matching problem. In the weakly stable matching problem an edge $e = (x, y)$ is blocking if by getting matched to each other both x and y would become better off. In the super stable matching problem an edge e is said to be blocking if neither x nor y would become worse off. A matching is respectively *weakly stable* or *super stable* if no blocking edge exists with respect to it.

Super stable matchings in instances of SRTI were investigated by Irving and Manlove [6]. They gave an $O(m)$ time algorithm for finding a super stable matching or reporting that no such matching exists. The algorithm is an extension of Irving's algorithms for finding stable matching in the SRI setting [4] and for finding super stable matching in instances of SMTI [5]. Using a polynomial time reduction from SRTI under super stability to 2-SAT, Fleiner, Irving and Manlove [2] deduced a number of structural results involving super stable matchings. These structural results allowed authors to give algorithms for computing all super stable pairs, enumeration of super stable matchings and finding a minimum regret super stable matching.

In contrast to strongly stable matchings and super stable matchings, the problem of determining whether a weakly stable matching exists in a non-bipartite graph was proven to be NP-complete by Ronn [15]. He proved that NP-completeness holds even if each preference list is either strictly ordered or contains a tie of length 2 at the head.

2 Preliminaries

We start with some additional notation. Let $\mathcal{I}$ be an instance of either SMTI or SRTI. Denote the set of all strongly stable matchings in $\mathcal{I}$ by $\mathcal{M}(\mathcal{I})$. Denote by $V(\mathcal{I})$, $E(\mathcal{I})$ the set of vertices and edges respectively of the underlying graph of $\mathcal{I}$. We say that an instance $\mathcal{I}$ is solvable if there is a strongly stable matching in the underlying graph. We define the rank of w in v's preference list, denoted by $rank(v, w)$, to be 1 plus the number of ties which are preferred to w by v. If $\mathcal{I}$ is a an instance of SMTI, then the underlying bipartite graph is of the form $G = (A \cup B, E)$. As is customary we call the vertices of A and B respectively men and women.

Below we give an overview of known structural results related to strongly stable matchings in bipartite graphs.

▶ **Theorem 1** ([7]). *There is an $\mathcal{O}(nm)$ algorithm to determine a man-optimal strongly stable matching of the given instance or report that no such matching exists.*

We say that a matching is *man-optimal* if every man gets the best partner among all his possible partners in any strongly stable matching. It can be proven that such a matching always exists if a given instance is solvable.

▶ **Theorem 2** (Rural Hospitals Theorem [9]). *In a given instance of* SMTI*, the same vertices are matched in all strongly stable matchings.*

We define an equivalence relation $\sim$ on $\mathcal{M}(\mathcal{I})$ as follows.

▶ **Definition 3.** For two strongly stable matchings M and N, $M \sim N$ if and only if each man m is indifferent between $M(m)$ and $N(m)$. Denote by $[M]$ the equivalence class containing M and denote by $\mathcal{X}$ the set of equivalence classes of $\mathcal{M}(\mathcal{I})$ under $\sim$.

Note that if there are no ties in the instance i.e. $\mathcal{I}$ is an instance of SMI, then each equivalence class of $\sim$ contains exactly one matching. It turns out that if ties are present in the instance, then an equivalence class can contain exponentially many matchings. To see that consider any bipartite graph G which admits a perfect matching. We construct an instance $\mathcal{J}$ of SMTI from G such that the preference list of every vertex is a single tie. Note that perfect matchings in G are strongly stable in $\mathcal{J}$ and all perfect matchings belong to the same equivalence class. If G admits exponentially many perfect matchings, then there are exponentially many strongly stable matchings in $\mathcal{J}$ as well.

Strongly stable matchings belonging to the same equivalence class can be easily characterised (more details in [10]). Thus we focus on structural results related to the set of equivalence classes of $\sim$.

For two strongly stable matchings M and N we say that M dominates N and write $N \preceq M$ if each man m weakly prefers $M(m)$ to $N(m)$. If M dominates N and there exists a man m who prefers $M(m)$ to $N(m)$ then we say that M strictly dominates N and we call N a successor of M.

Next we define a partial order $\preceq^*$ on $\mathcal{X}$.

▶ **Definition 4.** For any two equivalence classes $[M]$ and $[N]$, $[M] \preceq^* [N]$ if and only if $M \preceq N$.

Let M and N be two strongly stable matchings. Consider the symmetric difference $M \oplus N$. By Theorem 2 this set contains only alternating cycles. These cycles display an interesting property captured in:

▶ **Lemma 5** ([10]). *Let M and N be two strongly stable matchings. Consider any alternating cycle C of $M \oplus N$. Let $(m_0, w_0, m_1, w_1, ..., m_{k-1}, w_{k-1})$ be the sequence of vertices of C where m_i are men and w_i are women. Then there are only three possibilities:*
- $(\forall m_i) w_i =_{m_i} w_{i+1}$ *and* $(\forall w_i) m_i =_{w_i} m_{i-1}$
- $(\forall m_i) w_i \prec_{m_i} w_{i+1}$ *and* $(\forall w_i) m_i \succ_{w_i} m_{i-1}$
- $(\forall m_i) w_i \succ_{m_i} w_{i+1}$ *and* $(\forall w_i) m_i \prec_{w_i} m_{i-1}$

Subscripts are taken modulo k.

Below we introduce two operations transforming pairs of strongly stable matchings into other strongly stable matchings.

▶ **Definition 6.** Let M and N be two strongly stable matchings. Consider any man m and his partners $M(m)$ and $N(m)$.

By $M \wedge N$ we denote the matching such that:

- if $M(m) \succeq_m N(m)$ then $(m, M(m)) \in M \wedge N$
- if $M(m) \prec_m N(m)$ then $(m, N(m)) \in M \wedge N$

Similarly by $M \vee N$ we denote the matching such that:
- if $M(m) \succ_m N(m)$ then $(m, N(m)) \in M \vee N$
- if $M(m) \preceq_m N(m)$ then $(m, M(m)) \in M \vee N$

From [10] it follows that both $M \vee N$ and $M \wedge N$ are strongly stable matchings, and $M, N \succeq M \vee N$ and $M, N \preceq M \wedge N$.

We extend operations $\vee$ and $\wedge$ to the set $\mathcal{X}$ of equivalence classes. Let $[M], [N] \in \mathcal{X}$. Denote $[M] \vee [N] = [M \vee N]$, $[M] \wedge [N] = [M \wedge N]$.

A *lattice* is a partially ordered set in which every two elements a, b have a unique infimum (denoted $a \vee b$) and a unique supremum (denoted $a \wedge b$). A lattice L with operations join $\vee$ and meet $\wedge$ is *distributive* if for any three elements x, y, z of L the following holds: $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$.

▶ **Theorem 7** ([10]). *The partial order $(\mathcal{X}, \preceq^*)$ with operations meet $\vee$ and join $\wedge$ defined above forms a distributive lattice.*

It is easy to give an example such that $\mathcal{X}$ is of exponential size. It turns out that it is possible to build a representation of the lattice which is of polynomial size. In order to describe its construction a few more definitions are needed.

▶ **Definition 8.** Let M and N be two strongly stable matchings such that $N \prec M$. We say that N is a **strict successor** of M if and only if there is no strongly stable matching M' such that $N \prec M' \prec M$.

Let M_0 be a man-optimal strongly stable matching, and let M_z be a woman optimal strongly stable matching. We call a sequence $(M_0, M_1, \ldots, M_z)$ such that $M_0 \succ M_1 \succ \ldots \succ M_z$ and M_{i+1} is a strict successor of M_i, a *maximal sequence of strongly stable matchings.*

▶ **Theorem 9** ([8]). *There is an $\mathcal{O}(nm)$ time algorithm to compute a maximal sequence of strongly stable matchings.*

This algorithm works as follows. We first compute a man-optimal matching M_0 in $\mathcal{O}(nm)$ time using the standard algorithm [7], then given a matching M_i we find a strict successor M_{i+1} or determine that M_i is a woman-optimal matching (more details in [8]). We iterate over i until we reach a woman-optimal matching. Using amortized analysis it can be proven that the algorithm runs in $\mathcal{O}(nm)$ time. The algorithm can be easily modified so that it starts with an arbitrary strongly stable matching instead of a man-optimal one.

▶ **Corollary 10.** *Let M_0 be a strongly stable matching. There is an $\mathcal{O}(nm)$ algorithm to compute a sequence of strongly stable matchings $(M_0, M_1, \ldots, M_k)$ such that M_k is a woman-optimal matching, and M_{i+1} is a strict successor of M_i for each i.*

An important property of this algorithm is that in successive matchings each vertex either stays matched to the same partner or gets a partner of a different rank.

▶ **Corollary 11.** *Let $\mathcal{S} = (M_0, M_1, \ldots, M_z)$ be any sequence of strongly stable matchings produced by the algorithm of Corollary 10. Then for each $v \in V(\mathcal{I})$ and $i < z$ we have either $M_i(v) = M_{i+1}(v)$ or $rank(v, M_i(v)) \neq rank(v, M_{i+1}(v))$.*

Let M and N be two strongly stable matchings such that N is a strict successor of M. Recall that $M \oplus N$ is a set of alternating cycles. Consider some matchings $M' \in [M]$, $N' \in [N]$. Depending on the choice of matchings M' and N' it might happen that $M \oplus N \neq M' \oplus N'$. Note that from the definition of $\sim$ it follows that for every vertex v we have $rank(v, M(v)) - rank(v, N(v)) = rank(v, M'(v)) - rank(v, N'(v))$. In other words when we transform a matching from $[M]$ into some matching from $[N]$ the change of v's rank does not depend on the choice of matchings from equivalence classes. This observation motivates the following definition.

▶ **Definition 12.** Let M and N be two strongly stable matchings such that N is a strict successor of M. For any vertex v denote $r_v = rank(v, M(v))$ and $r'_v = rank(v, N(v))$. We say that a set of triples $\rho([M], [N]) = \{(v, r_v, r'_v) : v \in V(\mathcal{I}), r_v \neq r'_v\}$ is a **rotation** transforming $[M]$ into $[N]$.

Let ρ be a rotation and M, N be two strongly stable matchings such that N is a strict successor of M. We say that the set of alternating cycles $M \oplus N$ *realizes* a rotation ρ if $\rho = \rho([M], [N])$. As noted above there are potentially many sets of cycles realizing a given rotation. A rotation ρ is *exposed* in $[M]$ if $\rho = \rho([M], [N])$ for some N which is a strict successor of M. We say that such a rotation *transforms* $[M]$ into $[N]$. Note that a given rotation may be exposed in many equivalence classes.

▶ **Theorem 13** ([8]). *Let $\mathcal{S} = (M_0, M_1, \ldots, M_z)$ be a maximal sequence of strongly stable matchings. For $i \in \{0, 1, \ldots, z-1\}$ denote $\rho_i = \rho([M_i], [M_{i+1}])$. Then the set $D(\mathcal{I}) = \{\rho_0, \rho_1, \ldots, \rho_{z-1}\}$ does not depend on the choice of $\mathcal{S}$, and $\rho_i \neq \rho_j$ for $i \neq j$.*

Note that given a maximal sequence of strongly stable matchings $(M_0, M_1, \ldots, M_z)$ we can easily compute rotations $(\rho_0, \rho_1, \ldots, \rho_{z-1})$ where $\rho_i = \rho([M_i], [M_{i+1}])$. Moreover the set $C_{\rho_i} = M_i \oplus M_{i+1}$ realizes ρ_i for each i. It is important to note that depending on the choice of the maximal sequence $\mathcal{S}$ alternating cycles in C_{ρ_i} may differ.

Let $v \in V(\mathcal{I})$, and let ρ be a rotation. If $(v, f, s) \in \rho$, then we say that ρ *moves* v from rank f to rank s. If a particular maximal sequence of strongly stable matchings $\mathcal{S}$ is given and $\rho = \rho([M_i], [M_{i+1}])$ for some i then we say that a rotation ρ moves v from $M_i(v)$ to $M_{i+1}(v)$.

Note that in an instance of SMI for each rotation there is exactly one set of alternating cycles realizing this rotation. This follows easily from the definition of a rotation and the fact that each equivalence class consists of exactly one matching. It can be proven that in this setting a set of cycles realizing a given rotation always consists of one cycle. Thus in an instance of SMI a rotation can be viewed as a single cycle. In the more general SMTI setting it may happen that a set of cycles realizing a given rotation consists of more than one cycle (Figure 1).

▶ **Definition 14.** Let $D(\mathcal{I})$ be the set of all rotations in a given instance $\mathcal{I}$. We define the order $\prec$ on elements of $D(\mathcal{I})$ as follows. We say that a rotation ρ precedes rotation ρ' and write $\rho \prec \rho'$ if and only if for every maximal sequence $\mathcal{S} = (M_0, M_1, \ldots, M_z)$ of strongly stable matchings we have $\rho = \rho([M_i], [M_{i+1}])$ and $\rho' = \rho([M_j], [M_{j+1}])$ for some i, j such that $i < j$.

Let Z be a subset of $D(\mathcal{I})$. We say that Z is a *closed set* if there is no $\rho \in D(\mathcal{I}) \setminus Z$ such that $\rho \prec \rho'$ for some $\rho' \in Z$. It turns out that each closed set corresponds to an equivalence class of $\sim$. Moreover given such a set we can efficiently find an equivalence class corresponding to it. We briefly explain how to do it.

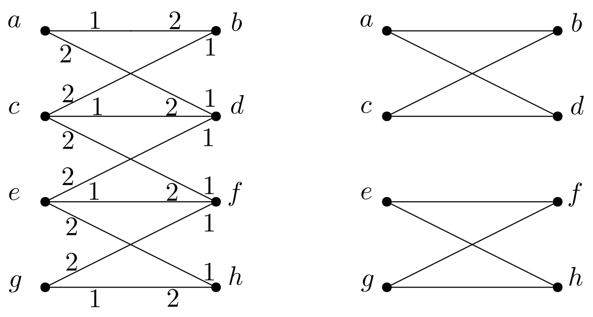


Figure 1 An example instance of SMTI is presented on the left hand side. Natural numbers denote ranks of the corresponding edges. This instance admits two strongly stable matchings $M_1 = \{(a,b),(c,d),(e,f),(g,h)\}$ and $M_2 = \{(a,d),(c,b),(e,h),(g,f)\}$. Cycles realizing the only rotation are presented on the right hand side.

First assume that we are given some particular maximal sequence $\mathcal{S} = (M_0, M_1, \ldots, M_z)$ of strongly stable matchings, the set of rotations $D(\mathcal{I})$, and for each rotation $\rho_i = \rho([M_i],[M_{i+1}])$ a set of cycles $C_{\rho_i} = M_i \oplus M_{i+1}$ realizing it.

▶ **Definition 15.** Let $Z = \{\rho_{a_0}, \rho_{a_1}, \ldots, \rho_{a_{k-1}}\}$ be a closed set. We order its elements so that there are no i, j such that $i < j$ and $\rho_{a_i} \succ \rho_{a_j}$. Let $N_0 = M_0$, $N_{i+1} = N_i \oplus C_{\rho_{a_i}}$. We denote $f_{\mathcal{S}}(Z) = N_k$.

Note that the sequence $\{N_i\}$ depends on the ordering of elements of Z, however its last element $f_{\mathcal{S}}(Z) = M_0 \oplus C_{\rho_{a_0}} \oplus C_{\rho_{a_1}} \oplus \ldots \oplus C_{\rho_{a_{k-1}}}$ is the same regardless of the ordering.

Intuition behind this definition is as follows. We start with an equivalence class $[N_0]$ ($N_0 = M_0$), and some ordering of elements of Z. First we apply a rotation ρ_{a_0} and get a matching $N_1 = N_0 \oplus C_{\rho_{a_0}}$ belonging to $[N_1]$. Then we apply ρ_{a_1} to N_1 and get $N_2 = N_1 \oplus C_{\rho_{a_1}}$ belonging to $[N_2]$. We continue this process until we apply all the rotations. In the end we get a matching $f_{\mathcal{S}}(Z) = N_k$, and a class $[N_k]$. Note that depending on the ordering of Z, sequences $\{N_i\}$ may go through different equivalence classes, however all possible orderings result in the same matching N_k, and an equivalence class $[N_k]$.

The next lemma says that every equivalence class can be obtained in this way from some closed set of rotations.

▶ **Lemma 16.** *For each equivalence class $[M]$ there is a closed set X such that $f_{\mathcal{S}}(X) \in [M]$. Let Z_1 and Z_2 be closed sets. Then $Z_1 \neq Z_2$ implies that $[f_{\mathcal{S}}(Z_1)] \neq [f_{\mathcal{S}}(Z_2)]$.*

For each closed set Z we define $g_{\mathcal{S}}(Z) = [f_{\mathcal{S}}(Z)]$. It can be proven that $g_{\mathcal{S}}$ does not depend on the choice of $\mathcal{S}$ and that $g_{\mathcal{S}}$ is a bijection between closed sets of $D(\mathcal{I}, \prec)$ and the set $\mathcal{X}$. The above discussion is summarized in the following theorem.

▶ **Theorem 17** ([8]). *There is a one-to-one correspondence between the set $\mathcal{X}$ of equivalence classes of $\sim$ and the closed sets of $(D(\mathcal{I}), \prec)$.*

It is important to note that given the function $f_{\mathcal{S}}$ we can get one strongly stable matching from each equivalence class and that depending on the choice of $\mathcal{S}$ these matchings may differ. In other words if $\mathcal{S} \neq \mathcal{S}'$ then it may happen that $f_{\mathcal{S}}(Z) \neq f_{\mathcal{S}'}(Z)$ for some Z, however regardless of the choice of $\mathcal{S}$ and $\mathcal{S}'$ we have $[f_{\mathcal{S}}(Z)] = [f_{\mathcal{S}'}(Z)]$. We emphasize this fact as our algorithm for the non-bipartite version of the problem is based on it.

Note that from Definition 14 alone it is non-trivial how to efficiently construct the relation $\prec$ on $D(\mathcal{I})$. Construction of an explicit representation of the relation $\prec$ would take $\Omega(m^2)$

time, because $D(\mathcal{I})$ might have $\Omega(m)$ elements. It can be proven that we can efficiently construct a sparse subgraph of $(D(\mathcal{I}), \prec)$ such that the closed sets of these two posets are identical.

▶ **Theorem 18** ([8]). *There is a graph $G' = (D(\mathcal{I}), E')$ such that $|E'| = \mathcal{O}(m)$, and the closed sets in G' are exactly the same as the closed sets in the poset $(D(\mathcal{I}), \prec)$. Such a graph can be constructed in $\mathcal{O}(nm)$ time.*

3 The Strongly Stable Roommates Problem

Let $\mathcal{I}$ be an instance of SRTI and let $G = (V, E)$ be the underlying graph. We define an auxiliary instance $\mathcal{I}'$ of SMTI and its underlying bipartite graph $H = (A \cup B, E')$. We make two copies of each vertex $v \in V(G)$, $v^p \in A$ – a *proposing node* and $v^r \in B$ – a *responding node*. For each edge $\{v, w\} \in E$ we add two edges (v^p, w^r) and (w^p, v^r) to E'. Each node in H inherits its preference list from the original instance i.e. for each edge $\{v, w\} \in E$ we have $rank(v^p, w^r) = rank(v, w)$ and $rank(w^r, v^p) = rank(w, v)$. Following the notation from [3] we denote edges of non-bipartite graphs as $\{x, y\}$ rather than (x, y) to emphasise the fact that pairs are unordered.

In the next few lemmas we show that we can derive some useful properties of the structure of strongly stable matchings in $\mathcal{I}$ from the structure of $\mathcal{I}'$. Throughout this section we assume that $\mathcal{I}$ is an instance of SRTI and that $\mathcal{I}'$ is defined as above.

▶ **Definition 19.** Let M be a matching in $\mathcal{I}'$. We say that M is a **symmetric matching** if for each edge $(v, w) \in E$ we have $(v^p, w^r) \in M \iff (w^p, v^r) \in M$.

▶ **Lemma 20.** *There is a one-to-one correspondence between strongly stable matchings in $\mathcal{I}$ and symmetric strongly stable matchings in $\mathcal{I}'$.*

Proof. Let M be a strongly stable matching in $\mathcal{I}$. Let M' be a symmetric matching such that for each $\{v, w\} \in M$ we add (v^p, w^r) and (w^p, v^r) to M'. We can easily see that M' is strongly stable. If there was some edge (x^p, y^r) blocking M' then M would be blocked by $\{x, y\}$. Similarly we can see that for each symmetric strongly stable matching M in $\mathcal{I}'$ there is a corresponding strongly stable matching M' in $\mathcal{I}$. For each $(v^r, w^p) \in M$ we add $\{v, w\}$ to M'. Stability of M' is simple to establish. It is obvious that it is a one-to-one correspondence. ◀

Let M be a strongly stable matching M in $\mathcal{I}$. We denote its symmetric counterpart in $\mathcal{I}'$ by $s(M)$. One can easily see that the equivalence class $[s(M)]$ might contain some matchings which are not symmetric. Moreover, some classes may not contain any symmetric matchings. For instance let M be a matching in $\mathcal{I}'$ such that for some vertex $v \in V(\mathcal{I})$ we have $rank(v^p, M(v^p)) \neq rank(v^r, M(v^r))$. It is clear that none of the matchings from $[M]$ can be symmetric. This observation leads to the following definition.

▶ **Definition 21.** Let M be a strongly stable matching in $\mathcal{I}'$. We say that the class $[M]$ is **symmetric** if for each $v \in V(\mathcal{I})$ we have $rank(v^p, M(v^p)) = rank(v^r, M(v^r))$.

It is obvious that if M is a symmetric matching then $[M]$ is also symmetric. Note that if a matching belongs to a symmetric equivalence class it does not imply that it is actually a symmetric matching. Do all the symmetric equivalence classes contain at least one symmetric matching? It turns out that the answer to this question is negative. A counterexample is presented in Figure 2. It turns out that the following theorem holds:

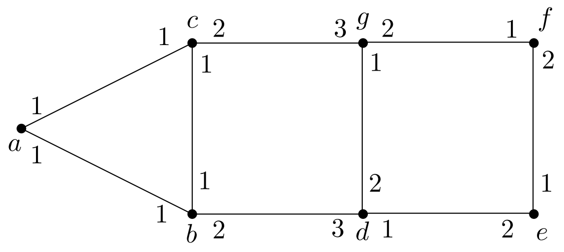


Figure 2 An example of an unsolvable instance $\mathcal{I}$ of SRTI. There are four strongly stable matchings in the corresponding auxiliary instance $\mathcal{I}'$ of SMTI:

- $M_1 = \{(a^p, b^r), (b^p, c^r), (c^p, a^r), (d^p, e^r), (e^p, f^r), (f^p, g^r), (g^p, d^r)\}$
- $M_2 = \{(a^p, b^r), (b^p, c^r), (c^p, a^r), (d^p, e^r), (e^p, d^r), (f^p, g^r), (g^p, f^r)\}$
- $M_3 = \{(a^p, b^r), (b^p, c^r), (c^p, a^r), (d^p, g^r), (g^p, d^r), (e^p, f^r), (f^p, e^r)\}$
- $M_4 = \{(a^p, b^r), (b^p, c^r), (c^p, a^r), (d^p, g^r), (g^p, f^r), (f^p, e^r), (e^p, d^r)\}$

Matchings M_2 and M_3 belong to symmetric equivalence classes but neither of them is symmetric.

▶ **Theorem 22.** *An instance $\mathcal{I}$ is solvable if and only if the following hold:*
- *at least one symmetric equivalence class exists in $\mathcal{I}'$*
- *each symmetric equivalence class in $\mathcal{I}'$ contains at least one symmetric strongly stable matching*

The above theorem implies that if we find a symmetric equivalence class which does not contain any symmetric matching, then the instance is unsolvable. Moreover if we find a strongly stable matching in a given instance $\mathcal{I}$ then every single symmetric equivalence class of $\mathcal{I}'$ contains at least one symmetric matching. Note that the above theorem is somewhat trivial in the case of strict preferences. In an instance of SMI each equivalence class contains exactly one matching. There are no ties, thus M is symmetric if and only if $[M]$ is symmetric.

In order to prove Theorem 22 we are going to make use of the function $f_{\mathcal{S}}$ (Definition 15). It turns out that if an instance $\mathcal{I}$ is solvable, then we can pick a maximal sequence $\mathcal{S}$ such that if a closed set Z corresponds to a symmetric equivalence class, then $f_{\mathcal{S}}(Z)$ is a symmetric matching. Below we give a candidate for such a sequence.

Denote $q(v^r) = v^p$ and $q(v^p) = v^r$ for $v \in V(\mathcal{I})$. For a given matching M in $\mathcal{I}'$ we denote $S(M) = \{(q(w), q(v)) : (v, w) \in M\}$.

▶ **Definition 23.** We say that a maximal sequence of strongly stable matchings $\mathcal{S} = (M_0, M_1, \ldots M_{2k})$ is **symmetric** if M_k is a symmetric matching and $S(M_i) = M_{2k-i}$ for each i.

It turns out that if a given instance of $\mathcal{I}'$ admits a symmetric strongly stable matching, then there is a symmetric maximal sequence of strongly stable matchings containing this matching.

▶ **Lemma 24.** *Let M_k be a woman-optimal matching in $\mathcal{I}'$, M_0 be a symmetric strongly stable matching, and let $(M_0, M_1, \ldots, M_k)$ be a sequence of strongly stable matchings such that M_{i+1} is a strict successor of M_i. Then the sequence:*

$$\mathcal{Q} = (S(M_k), S(M_{k-1}), \ldots, S(M_1), M_0, M_1, \ldots, M_k)$$

is a symmetric maximal sequence of strongly stable matchings.

Proof. It suffices to prove that for each i the matching $S(M_i)$ is strongly stable, $S(M_k)$ is man-optimal, and that $S(M_{i-1})$ is a strict successor of $S(M_i)$. Strong stability of $S(M_i)$

follows easily from strong stability of M_i. If there was an edge (v^p, w^r) blocking $S(M_i)$ then (w^p, v^r) would block M_i. It can be easily proven that for any two strongly stable matchings N, N' we have $N \prec N' \iff S(N') \prec S(N)$. If there was a matching N such that $S(M_k) \prec N$, then $S(N) \prec M_k$ would hold, thus M_k would not be woman-optimal. Similarly if there was a strongly stable matching M' such that $S(M_i) \succ M' \succ S(M_{i-1})$, then there would be $M_{i-1} \succ S(M') \succ M_i$. This would contradict the assumption that M_i is a strict successor of M_{i-1}. Thus for each i the matching $S(M_{i-1})$ is a strict successor of $S(M_i)$. Hence $\mathcal{Q}$ is a symmetric maximal sequence of strongly stable matchings. ◀

Corollary 10 along with the proof of Lemma 24 imply that given a symmetric strongly stable matching M in $\mathcal{I}'$ we can compute a symmetric maximal sequence of strongly stable matchings.

▶ **Corollary 25.** *Assume that we are given a symmetric strongly stable matching M in $\mathcal{I}'$. Then there exists a symmetric maximal sequence of strongly stable matchings $(M_0, M_1, \ldots, M_{2k})$ such that $M_k = M$. We can compute such a sequence in $\mathcal{O}(nm)$ time.*

Before we prove Theorem 22 we need to investigate the structure of the set of rotations $D(\mathcal{I}')$. Let M be a symmetric strongly stable matching. Consider a symmetric sequence of strongly stable matchings $\mathcal{S} = (M_0, M_1, \ldots, M_{2k})$ such that $M_k = M$. Let $\rho_i = \rho([M_i], [M_{i+1}])$ for each i. Theorem 13 implies that $D(\mathcal{I}') = \{\rho_0, \rho_1, \ldots, \rho_{2k-1}\}$. From Theorem 17 we know that the set $Z = \{\rho_0, \rho_1, \ldots, \rho_{k-1}\}$ is closed and it corresponds to the equivalence class $[M]$. It turns out that the set of remaining rotations $D(\mathcal{I}') \setminus Z$ has very similar structure to Z. In order to see this we consider matchings M_i, M_{i+1} and their symmetric counterparts $M_{2k-i} = S(M_i)$, $M_{2k-i-1} = S(M_{i+1})$. Let $v \in V(\mathcal{I})$, $w_1^r = M_i(v^p)$, and $w_2^r = M_{i+1}(v^p)$. Observe that rotation ρ_i moves v^p from w_1^r to w_2^r. We can easily see that ρ_{2k-i-1} moves v^r from w_2^p to w_1^p. This observation motivates the following definition.

For a given rotation ρ in $\mathcal{I}'$ we denote $\overline{\rho} = \{(q(v), s, f) : (v, f, s) \in \rho\}$. Similarly for a given cycle C in $\mathcal{I}'$ we denote $\overline{C} = \{(q(w), q(v)) : (v, w) \in C\}$. We say that $\overline{\rho}$ is the *rotation dual* to ρ. Analogously for a given cycle C we say that $\overline{C}$ is the *cycle dual* to C. Note that the rotation dual to $\overline{\rho}$ is equal to ρ. Similarly the cycle dual to $\overline{C}$ is equal to C. We say that the set of all rotations $D(\mathcal{I}')$ is *symmetric* if for each $\rho \in D(\mathcal{I}')$ both $\overline{\rho} \in D(\mathcal{I}')$ and $\rho \neq \overline{\rho}$ hold.

▶ **Theorem 26.** *If $\mathcal{I}$ is solvable, then the set of rotations $D(\mathcal{I}')$ is symmetric.*

Proof. Let $\mathcal{S} = (M_0, M_1, \ldots, M_{2k})$ be a symmetric maximal sequence of strongly stable matchings. Such a sequence exists from Corollary 25. Denote $\rho_i = \rho([M_i], [M_{i+1}])$ for each i. From Theorem 13 we know that $D(\mathcal{I}') = \{\rho_0, \rho_1, \ldots, \rho_{2k-1}\}$, and that $\rho_i \neq \rho_j$ for $i \neq j$. One can easily check that ρ_i is dual to ρ_{2k-i-1} for each i. Thus $D(\mathcal{I}')$ is symmetric. ◀

From the above theorem we know that if $D(\mathcal{I}')$ is not symmetric, then for sure $\mathcal{I}$ does not admit any strongly stable matching. From Theorems 9 and 13 we know how to compute the set $D(\mathcal{I}')$ in $\mathcal{O}(nm)$ time. We can easily check if this set is symmetric.

▶ **Corollary 27.** *There is an $\mathcal{O}(nm)$ algorithm to determine if $D(\mathcal{I}')$ is symmetric.*

It turns out that if the set $D(\mathcal{I}')$ is symmetric then we can characterise the set of symmetric equivalence classes of $\mathcal{I}'$. We say that a set $Z \subseteq D(\mathcal{I}')$ is *complete* if for each rotation $\rho \in D(\mathcal{I}')$ either $\rho \in Z$ or $\overline{\rho} \in Z$. Similarly to Lemma 16, for a given maximal sequence $\mathcal{S}$ we consider the function $f_{\mathcal{S}}$ (Definition 15) however this time we restrict its domain to the set of complete closed sets.

▶ **Lemma 28.** *Let $D(\mathcal{I}')$ be symmetric, $\mathcal{S} = (M_0, M_1, \dots, M_z)$ be any maximal sequence of strongly stable matchings. For each set $Z \subseteq D(\mathcal{I}')$ which is complete and closed $f_\mathcal{S}(Z)$ belongs to a symmetric equivalence class. Moreover for each symmetric equivalence class $[M]$ there is a closed and complete set Z such that $f_\mathcal{S}(Z) \in [M]$.*

Proof. Let Z be a complete and closed set of rotations. We prove that $f_\mathcal{S}(Z)$ belongs to a symmetric equivalence class. First let us order rotations of $Z = \{\rho_0, \rho_1, \dots, \rho_k\}$, so that if $i < j$, then $\rho_i \not\succ \rho_j$. Then we denote $N_0 = M_0$, $N_{i+1} = N_i \oplus C_{\rho_i}$ for $0 \leq i < k$. From the definition of rotation it follows that N_{i+1} is a strict successor of N_i. Consider an arbitrary vertex $v \in V(\mathcal{I})$. In order to prove that $f_\mathcal{S}(Z)$ belongs to a symmetric equivalence class we need to show that $rank(v^p, N_k(v^p)) = rank(v^r, N_k(v^r))$. For each i the rotation ρ_i moves v^p from $N_i(v^p)$ to $N_{i+1}(v^p)$, thus Z moves v^p from $N_0(v^p)$ to $N_k(v^p)$. From the definition of dual rotation we know that $\overline{\rho_i}$ moves v^r from $q(N_{i+1}(v^p))$ to $q(N_i(v^p))$, hence $D(\mathcal{I}') \setminus Z$ moves v^r from $q(N_k(v^p))$ to $q(N_0(v^p))$ From the completeness of Z we obtain that Z moves v^r from $N_0(v^r)$ to $q(N_k(v^p))$. Thus $rank(v^r, N_k(v^r)) = rank(v^r, q(N_k(v^p)) = rank(v^p, N_k(v^p))$, hence $f_\mathcal{S}(Z)$ belongs to a symmetric equivalence class.

Let $[M]$ be a symmetric equivalence class. We prove that there is a closed and complete set Z such that $f_\mathcal{S}(Z) \in [M]$. From Lemma 24 there exists a symmetric maximal sequence of strongly stable matchings $(M_0, M_1, \dots, M_{2k})$ such that $M = M_k$. Analogously to the proof of Theorem 26 we denote $\rho_i = \rho([M_i], [M_{i+1}])$ for each i. From Theorem 13 we know that $D(\mathcal{I}') = \{\rho_0, \rho_1, \dots, \rho_{2k-1}\}$. Consider the set $Z = \{\rho_0, \rho_1, \dots, \rho_{k-1}\}$. This set is obviously closed. One can easily check that ρ_i is dual to ρ_{2k-i-1}, thus Z is complete. From Theorem 17 we have $f_\mathcal{S}(Z) \in [M]$. ◀

Analogously to Theorem 17 we consider a function $g_\mathcal{S}(Z) = [f_\mathcal{S}(Z)]$ and obtain the following theorem.

▶ **Theorem 29.** *There is a one-to-one correspondence between the symmetric equivalence classes and the complete closed subsets of rotations in $(D(\mathcal{I}'), \prec)$.*

It may happen that the set of rotations is symmetric but none of the symmetric classes contains a symmetric strongly stable matching (Figure 2). If an instance $\mathcal{I}$ is solvable, then we can pick a maximal sequence $\mathcal{S}$ of strongly stable matchings, such that all values of the function $f_\mathcal{S}$ are symmetric strongly stable matchings. Note that this implies Theorem 22 and allows us to characterise strongly stable matchings in a non-bipartite instance $\mathcal{I}$. Below we prove that it suffices to take a sequence obtained from Corollary 25 as $\mathcal{S}$.

▶ **Theorem 30.** *Let $\mathcal{Q} = (M_0, M_1, \dots, M_{2k})$ be a symmetric maximal sequence of strongly stable matchings obtained as in Corollary 25. For each complete and closed set Z of rotations $f_\mathcal{Q}(Z)$ is a symmetric matching.*

Proof. Let Z be a closed and complete set of rotations. Denote $M = f_\mathcal{Q}(Z)$. Let $v \in V(\mathcal{I})$ be any vertex, and let $M(v^p) = w^r$. Our goal is to prove that $M(v^r) = w^p$. From Lemma 28 we know that M belongs to a symmetric equivalence class, thus $rank(v^p, M(v^p)) = rank(v^r, M(v^r))$. Denote this rank by r. From the definition of $f_\mathcal{Q}$ if an edge belongs to M then it must belong to one of the matchings M_i. Recall that since $\mathcal{Q}$ is a sequence obtained from Corollary 25 we have that for each i, and a vertex x either $M_i(x) = M_{i+1}(x)$ or $rank(x, M_i(x)) \neq rank(x, M_{i+1}(x))$ (Corollary 11). This observation, and the fact that $M_0 \succ M_1 \succ \dots \succ M_{2k}$ imply that there exists exactly one vertex of rank r matched to v^p amongst all vertices matched to v^p in matchings of $\mathcal{Q}$. Let us denote this vertex by w^r, and assume that $M_j(v^p) = w^r$ for some j. Similarly it can be proven that there exists exactly

Algorithm 1 for computing a symmetric equivalence class

```
1: let $\mathcal{I}'$ be an auxiliary symmetric instance of SMTI
2: if $\mathcal{I}'$ is unsolvable or $D(\mathcal{I}')$ is not symmetric then
3:     return no
4: build a graph $G'$ representing the poset of rotations (Theorem 18)
5: $Z \leftarrow \emptyset$
6: for $\rho : \rho \in D(\mathcal{I}')$ do
7:     $indeg(\rho) \leftarrow 0$
8:     $marked(\rho) \leftarrow 0$
9: for $\rho : \rho \in D(\mathcal{I}')$ do
10:     for $\rho' : (\rho, \rho') \in G'$ do
11:         $indeg(\rho') \leftarrow indeg(\rho') + 1$
12: $Q \leftarrow \{\rho \in D(\mathcal{I}') : indeg(\rho) = 0\}$
13: while $Q \neq \emptyset$ do
14:     $\rho \leftarrow$ any element from $Q$
15:     $Q \leftarrow Q \setminus \rho$
16:     if $marked(\rho) = 0$ then
17:         $Z \leftarrow Z \cup \{\rho\}$
18:         $marked(\overline{\rho}) \leftarrow 1$
19:         for $\rho' : (\rho, \rho') \in G'$ do
20:             $indeg(\rho') \leftarrow indeg(\rho') - 1$
21:             if $indeg(\rho') = 0$ then
22:                 $Q \leftarrow Q \cup \{\rho'\}$
23: return a matching corresponding to $Z$
```

one vertex of rank r matched to v^r amongst all vertices matched to v^r in matchings of $\mathcal{Q}$. The fact that $\mathcal{Q}$ is symmetric implies that $M_{2k-j}(v^r) = w^p$, so v^r must be matched to w^p in M. This implies that M is a symmetric matching. ◀

Assuming that we are given one symmetric matching in $\mathcal{I}'$ we can efficiently construct $f_{\mathcal{Q}}$ as shown in Corollary 25. It remains to show how to find some symmetric equivalence class and check if there exists a symmetric matching belonging to this class. If such a matching does not exist then Theorem 22 implies that $\mathcal{I}$ is unsolvable.

Algorithm 1 for determining a symmetric equivalence class works as follows. We first build the symmetric instance $\mathcal{I}'$. If this instance is unsolvable or $D(\mathcal{I}')$ is not symmetric then $\mathcal{I}$ is unsolvable from Theorem 26. From now on we assume that $D(\mathcal{I}')$ is symmetric. We prove that in this case we can find a symmetric equivalence class in $\mathcal{I}'$. The intuition behind the algorithm is very simple. We first construct a set Z which is closed and complete. Initially we set $Z = \emptyset$. During each step of the algorithm we add to Z some rotations ρ such that there does not exist any rotation $\rho' \notin Z$ such that $\rho' \prec \rho$. It is enough to make sure that Z is closed at all times of the execution. In order to make sure that the resulting set is complete, when we add a rotation ρ to Z we mark $\overline{\rho}$, and never add any marked rotations to Z. Once the set Z is constructed we use Theorem 29 to obtain a symmetric equivalence class corresponding to it. Before we prove the correctness of the algorithm we need the following technical lemma.

▶ **Lemma 31.** *Let $\rho, \rho' \in D(\mathcal{I}')$ be two rotations such that $\rho \prec \rho'$. Then $\overline{\rho'} \prec \overline{\rho}$ holds.*

Proof. Assume that $\overline{\rho'} \prec \overline{\rho}$ does not hold. From the definition of $\prec$ we know that there exists a maximal sequence of strongly stable matchings $\mathcal{S} = (M_0, M_1, \ldots, M_{2k})$ such that $\overline{\rho'} = \rho_j$, $\overline{\rho} = \rho_l$ for some $l < j$, where we denote $\rho_i = \rho([M_i], [M_{i+1}])$ for each i. One can easily see that $\mathcal{S}' = (S(M_{2k}), S(M_{2k-1}), \ldots, S(M_0))$ is also a maximal sequence of strongly stable matchings. It can be proven that $\overline{\rho_i} = \rho([S(M_{i+1})], [S(M_i)])$ for each i. Thus $\rho' = \overline{\rho_j}$ and $\rho = \overline{\rho_l}$ – a contradiction with the definition of $\prec$. ◀

Correctness of the algorithm is proven in the following theorem.

▶ **Theorem 32.** *Let $\mathcal{I}$ be an instance of* SRTI*, and let $\mathcal{I}'$ be the auxiliary instance of* SMTI*. Algorithm 1 determines a matching belonging to a symmetric equivalence class of $\mathcal{I}'$ or reports that such a matching does not exist. The runtime of the algorithm is bounded by $\mathcal{O}(nm)$.*

Proof. It is clear from Theorem 26 that if the algorithm returns *no* in line 4, then $\mathcal{I}$ is unsolvable.

We first prove that Z is closed. Closed sets in the poset $(D(\mathcal{I}'), \prec)$, and in G' are identical, hence it suffices to prove that Z is closed in G'. At the start of the execution we have $Z = \emptyset$, so Z is closed. A rotation can be added to Z only if all its immediate predecessors in G' have already been added to Z. It implies that when a rotation is added to Z this set remains closed, hence Z is closed during the entire execution of the algorithm.

We prove that Z is complete when we exit the *while* loop (lines $14 - 23$). Assume to the contrary that Z is not complete. Then there is some rotation ρ such that $\rho, \overline{\rho} \notin Z$. From the pseudocode we can see that neither ρ nor $\overline{\rho}$ is marked. Since $\rho \notin Z$ there exists some rotation ρ' such that $(\rho', \rho) \in G'$ and $\rho' \notin Z$, otherwise at some point ρ would have been added to Q in the line 23, and either ρ or $\overline{\rho}$ would be added to Z. If ρ' is marked then $\overline{\rho'} \in Z$. From Lemma 31 we know that $\rho' \prec \rho$ implies that $\overline{\rho} \prec \overline{\rho'}$ – a contradiction with the fact that Z is closed. Hence ρ' cannot be marked. The rotation $\overline{\rho'}$ is also unmarked, because $\rho' \notin Z$. We can do the same reasoning for ρ' and $\overline{\rho'}$ and get another rotation $\rho'' \notin Z$, such that $\overline{\rho''} \notin Z$ and neither ρ'' nor $\overline{\rho''}$ is marked. We continue this process building a sequence of rotations $\rho \succ \rho' \succ \rho'' \succ \ldots$, and we eventually get a contradiction because our poset is finite. Thus Z is complete, and it corresponds to a symmetric equivalence class.

Let us estimate the complexity of the algorithm. From Corollary 27 we know that computations in lines $3 - 4$ take $\mathcal{O}(nm)$ time. Then we build a graph G' in time $\mathcal{O}(nm)$ (Theorem 18). Number of operations performed in lines $14 - 23$ is proportional to the number of edges of G', which is bounded by $\mathcal{O}(nm)$. Thus the algorithm runs in $\mathcal{O}(nm)$ time overall. ◀

The last step of the algorithm is to show how to determine if a symmetric matching exists in a given symmetric equivalence class.

▶ **Theorem 33.** *Let M be a strongly stable matching belonging to a symmetric equivalence class. We can determine in $\mathcal{O}(\sqrt{n}m)$ time if there is a symmetric strongly stable matching belonging to $[M]$.*

Proof. We will construct a subgraph $G' = (V', E')$ of G (recall that G is the underlying graph of $\mathcal{I}$) such that perfect matchings in G' correspond to symmetric matchings in $[M]$. Let us consider a matching M. For each $v \in V(\mathcal{I})$ such that v^p and v^r are matched in M we add v to V'. We also add to E' each edge (v, w) such that $rank(v, M(v)) = rank(v, w)$ and $rank(w, v) = rank(w, M(w))$. Similarly to the proof of Lemma 20 it can be shown that each symmetric matching in $[M]$ corresponds to a perfect matching in G'. It remains to compute a maximum matching in the non-bipartite graph G' [12]. ◀

The algorithm for computing a single strongly stable matching follows easily from the above discussion. Given an instance $\mathcal{I}$ of SRTI, we first use Algorithm 1 in order to compute a strongly stable matching M belonging to a symmetric equivalence class in an auxiliary instance $\mathcal{I}'$. Then from Theorem 33 we determine a symmetric matching belonging to $[M]$, and output its counterpart in $\mathcal{I}$. Thus we obtain the following theorem:

▶ **Theorem 34.** *There is an $O(nm)$ time algorithm to determine a single strongly stable matching in an instance of* SRTI *or to report that no such matching exists.*

In the Introduction we mentioned that Scott's $O(m^2)$ algorithm contained some flaws. We explain the problem with his algorithm using our terminology. Scott's algorithm can correctly determine whether a symmetric equivalence class exists in an auxiliary instance $\mathcal{I}'$. He claims that a symmetric strongly stable matching can always be found in a given symmetric equivalence class (Lemma 3.3.4 in [16]). This is a false statement as we can see in Figure 2. The algorithm can be repaired using for instance Theorem 33, however an analogue of Theorem 22 is needed to prove its correctness.

Given a single strongly stable matching we can easily construct a representation of the set of all strongly stable matchings. Using Corollary 25 we compute a symmetric maximal sequence of strongly stable matchings and then construct the poset of rotations as in Theorem 17. Closed and complete subsets of rotations correspond to strongly stable matchings from Theorem 29 hence the following holds:

▶ **Theorem 35.** *There is an $O(nm)$ time algorithm to construct a poset $(D(\mathcal{I}'), \prec)$, such that closed and complete subsets of $D(\mathcal{I}')$ correspond to symmetric equivalence classes of an auxiliary instance $\mathcal{I}'$.*

Acknowledgements. I would like to thank Katarzyna Paluch and the anonymous reviewers for their helpful comments.

References

1 Brian C. Dean and Siddharth Munshi. Faster algorithms for stable allocation problems. *Algorithmica*, 58(1):59–81, 2010. doi:10.1007/s00453-010-9416-y.

2 Tamás Fleiner, Robert W. Irving, and David F. Manlove. Efficient algorithms for generalized stable marriage and roommates problems. *Theor. Comput. Sci.*, 381(1-3):162–176, 2007. doi:10.1016/j.tcs.2007.04.029.

3 Dan Gusfield and Robert W. Irving. *The Stable marriage problem – structure and algorithms.* Foundations of computing series. MIT Press, 1989.

4 Robert W. Irving. An efficient algorithm for the "stable roommates" problem. *J. Algorithms*, 6(4):577–595, 1985. doi:10.1016/0196-6774(85)90033-1.

5 Robert W. Irving. Stable marriage and indifference. *Discrete Applied Mathematics*, 48(3):261–272, 1994. doi:10.1016/0166-218X(92)00179-P.

6 Robert W. Irving and David F. Manlove. The stable roommates problem with ties. *J. Algorithms*, 43(1):85–105, 2002. doi:10.1006/jagm.2002.1219.

7 Telikepalli Kavitha, Kurt Mehlhorn, Dimitrios Michail, and Katarzyna E. Paluch. Strongly stable matchings in time $O(nm)$ and extension to the hospitals-residents problem. *ACM Trans. Algorithms*, 3(2), 2007. doi:10.1145/1240233.1240238.

8 Adam Kunysz, Katarzyna E. Paluch, and Pratik Ghosal. Characterisation of strongly stable matchings. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 107–119, 2016. doi:10.1137/1.9781611974331.ch8.

9 David F. Manlove. Stable marriage with ties and unacceptable partners. Technical report, University of Glasgow, 1999.

10 David F. Manlove. The structure of stable marriage with indifference. *Discrete Applied Mathematics*, 122(1-3):167–181, 2002. `doi:10.1016/S0166-218X(01)00322-5`.

11 David F. Manlove. *Algorithmics of Matching Under Preferences*, volume 2 of *Series on Theoretical Computer Science*. WorldScientific, 2013. `doi:10.1142/8591`.

12 Silvio Micali and Vijay V. Vazirani. An o(sqrt(|v|) |e|) algorithm for finding maximum matching in general graphs. In *21st Annual Symposium on Foundations of Computer Science, Syracuse, New York, USA, 13-15 October 1980*, pages 17–27. IEEE Computer Society, 1980. `doi:10.1109/SFCS.1980.12`.

13 Nitsan Perach, Julia Polak, and Uriel G. Rothblum. A stable matching model with an entrance criterion applied to the assignment of students to dormitories at the technion. *Int. J. Game Theory*, 36(3-4):519–535, 2008. `doi:10.1007/s00182-007-0083-4`.

14 Nitsan Perach and Uriel G. Rothblum. Incentive compatibility for the stable matching model with an entrance criterion. *Int. J. Game Theory*, 39(4):657–667, 2010. `doi:10.1007/s00182-009-0210-5`.

15 Eytan Ronn. Np-complete stable matching problems. *J. Algorithms*, 11(2):285–304, 1990. `doi:10.1016/0196-6774(90)90007-2`.

16 Sandy Scott. *A study of stable marriage problems with ties.* PhD thesis, University of Glasgow, 2005.

Faster External Memory LCP Array Construction

Juha Kärkkäinen[1] **and Dominik Kempa**[2]

1 **Helsinki Institute for Information Technology HIIT & Department of Computer Science, University of Helsinki, Helsinki, Finland**
`juha.karkkainen@cs.helsinki.fi`

2 **Helsinki Institute for Information Technology HIIT & Department of Computer Science, University of Helsinki, Helsinki, Finland**
`dominik.kempa@cs.helsinki.fi`

Abstract

The suffix array, perhaps the most important data structure in modern string processing, needs to be augmented with the longest-common-prefix (LCP) array in many applications. Their construction is often a major bottleneck especially when the data is too big for internal memory. We describe two new algorithms for computing the LCP array from the suffix array in external memory. Experiments demonstrate that the new algorithms are about a factor of two faster than the fastest previous algorithm.

1998 ACM Subject Classification E.1 Data Structures, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases LCP array, suffix array, external memory algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.61

1 Introduction

The suffix array [22, 10], a lexicographically sorted list of the suffixes of a text, is the most important data structure in modern string processing. It is frequently augmented with the longest-common-prefix (LCP) array, which stores the lengths of the longest common prefixes between lexicographically adjacent suffixes. Together they are the basis of powerful text indexes such as enhanced suffix arrays [1] and many compressed full-text indexes [25]. Modern textbooks spend dozens of pages in describing their applications, see e.g. [28, 21].

The construction of the suffix and LCP arrays has been heavily studied over the years. Recently, several algorithms and implementations for constructing the suffix array in external memory have been published [7, 4, 6, 11, 27, 26, 19, 15]. Such algorithms are frequently needed for handling large texts or text collections that are too big to process in RAM. Some of the algorithms can also compute the LCP array simultaneously with the suffix array [4, 6] and others could probably be modified to do so. However, such a modification is unique to each algorithm and can significantly increase the construction time as well as the disk space usage of the algorithm [4].

A better solution for constructing the LCP array is to construct the suffix array separately first and then compute the LCP array from the suffix array. This has been a standard practice in internal memory for 15 years [18] but became possible in external memory only recently with the introduction of the LCPscan algorithm [12, 13]. This led to a significant improvement in construction time as well as in disk space usage over previous approaches.

Furthermore, since LCPscan can be combined with any suffix array construction algorithm, it can immediately benefit from any progress in the fast developing field of suffix array construction. For example, the recent pSAscan algorithm [15] can often construct the suffix

Table 1 The time and I/O complexities of LCPscan and the new algorithms in the standard external memory model [34]. The parameters are the text length n, the alphabet size σ, the available RAM M (in units of $\log n$ bits), and the disk block size B (in units of $\log n$ bits).

Algorithm	Time complexity	I/O complexity
LCPscan	$\mathcal{O}\left(\frac{n^2}{M(\log_\sigma n)^2} + n\log_{\frac{M}{B}}\frac{n}{B}\right)$	$\mathcal{O}\left(\frac{n^2}{MB(\log_\sigma n)^2} + \frac{n}{B}\log_{\frac{M}{B}}\frac{n}{B}\right)$
SPARSE-Φ	$\mathcal{O}\left(\frac{n^2}{M} + n\log_{\frac{M}{B}}\frac{n}{B}\right)$	$\mathcal{O}\left(\frac{n^2}{MB(\log_\sigma n)^2} + \frac{n}{B}\log_{\frac{M}{B}}\frac{n}{B}\right)$
SUCCINCTIRREDUCIBLE	$\mathcal{O}\left(\frac{n^2}{M(\log_\sigma n)^2} + n\log n\right)$	$\mathcal{O}\left(\frac{n^2}{MB(\log_\sigma n)^2} + \frac{n\log\sigma}{B} + \frac{n}{B}\log_{\frac{M}{B}}\frac{n}{B}\right)$

array significantly faster than the LCPscan algorithm can construct the LCP array [13, Table IX]. Thus there is a need for even faster LCP array construction in external memory.

Our contribution. In this paper, we describe two new external memory algorithms for constructing the LCP array from the suffix array. Although the new algorithms share some features with LCPscan their more immediate ancestors are two semi-external algorithms introduced in [16]. The semi-external algorithms need to keep the text and some additional small data structures in RAM but the larger suffix and LCP arrays are kept on disk and are accessed sequentially only. When there is enough RAM, these algorithms are many times faster than LCPscan.

We show how the requirement to keep the text (and some additional data structures) can be removed from the algorithms. Although this adds a significant amount of computation and I/O, the resulting algorithms are still about a factor of two faster than LCPscan in our experiments. Asymptotically, LCPscan has a slight advantage over the new algorithms (see Table 1). The main disadvantage of LCPscan is that it relies heavily on external memory sorting, which is completely avoided by the new algorithms.

The advantage of the new algorithms over LCPscan is particularly large when the text is only slightly larger than the available RAM. This is a common situation when dealing with compressed full-text self-indexes [25]. A compressed index can be significantly smaller than an uncompressed text and fit in RAM even though the text does not. However, the construction of the index still requires external memory computation.

Related work. Kasai et al. [18] introduced the first (internal memory) algorithm for computing the LCP array from the suffix array. It is simple and fairly fast but requires a lot of space. Thus a lot of the later work focused on reducing the space [20, 23, 30, 16, 33, 9, 3]. A culmination of this line of work are semi-external algorithms that keep most of the data structures on disk but need to have at least the text in RAM [30, 16]. There is also recent research on speeding up LCP computation by using parallelism [8, 32].

External memory algorithms for constructing the suffix array have been around since the early days [10], but until recently the only way to construct the LCP array when the text does not fit in RAM was as byproduct of a suffix array construction algorithm [17, 4, 6, 2]. To the best of our knowledge, LCPscan [12] is still the only external memory algorithm that can construct the LCP array from the suffix array independently of how it was constructed.

2 Basic Data Structures

Throughout we consider a string $\mathsf{X} = \mathsf{X}[0..n) = \mathsf{X}[0]\mathsf{X}[1]\ldots\mathsf{X}[n-1]$ of $|\mathsf{X}| = n$ symbols drawn from an alphabet of size σ. Here and elsewhere we use $[i..j)$ as a shorthand for

Table 2 Examples of the arrays used by the algorithms for the text $\mathsf{X} = \texttt{babaabbabbab}$.

i	0	1	2	3	4	5	6	7	8	9	10	11	12
$\mathsf{X}[i]$	b	a	b	a	a	b	b	a	b	b	a	b	-
$\mathrm{SA}[i]$	12	3	10	1	7	4	11	2	9	0	6	8	5
$\mathrm{BWT}[i]$	b	b	b	b	b	a	a	a	b	$	b	a	a
$\Phi[i]$	9	10	11	12	7	8	0	1	6	2	3	4	-
$\mathrm{LCP}[i]$	-	0	1	2	2	5	0	1	2	3	3	1	4
$\mathrm{PLCP}[i]$	3	2	1	0	5	4	3	2	1	2	1	0	-
$i + \mathrm{PLCP}[i]$	3	3	3	3	9	9	9	9	9	11	11	11	-
$\Phi[i] + \mathrm{PLCP}[i]$	12	12	12	12	12	12	3	3	7	4	4	4	-

$[i..j-1]$. For $i \in [0..n]$, we write $\mathsf{X}[i..n)$ to denote the *suffix* of X of length $n-i$, that is $\mathsf{X}[i..n) = \mathsf{X}[i]\mathsf{X}[i+1]\ldots\mathsf{X}[n-1]$. We will often refer to suffix $\mathsf{X}[i..n)$ simply as "suffix i".

The *suffix array* [22, 10] SA of X is an array $\mathrm{SA}[0..n]$ which contains a permutation of the integers $[0..n]$ such that $\mathsf{X}[\mathrm{SA}[0]..n) < \mathsf{X}[\mathrm{SA}[1]..n) < \cdots < \mathsf{X}[\mathrm{SA}[n]..n)$. In other words, $\mathrm{SA}[j] = i$ iff $\mathsf{X}[i..n)$ is the $(j+1)^{\mathrm{th}}$ suffix of X in ascending lexicographical order. Another representation of the permutation is the Φ *array* [16] $\Phi[0..n)$ defined by $\Phi[\mathrm{SA}[j]] = \mathrm{SA}[j-1]$ for $j \in [1..n]$. In other words, the suffix $\Phi[i]$ is the immediate lexicographical predecessor of the suffix i, and thus $\mathrm{SA}[n-k] = \Phi^k[\mathrm{SA}[n]]$ for $k \in [0..n]$. An example illustrating the arrays is given in Table 2.

Let $\mathrm{lcp}(i,j)$ denote the length of the longest-common-prefix (LCP) of suffix i and suffix j. For instance, in the example of Table 2, $\mathrm{lcp}(0,6) = 3 = |\texttt{bab}|$ and $\mathrm{lcp}(7,4) = 5 = |\texttt{abbab}|$. The *longest-common-prefix array* [22, 18], $\mathrm{LCP}[1..n]$, is defined such that $\mathrm{LCP}[i] = \mathrm{lcp}(\mathrm{SA}[i], \mathrm{SA}[i-1])$ for $i \in [1..n]$. The *permuted LCP array* [16] $\mathrm{PLCP}[0..n)$ is the LCP array permuted from the lexicographical order into the text order, i.e., $\mathrm{PLCP}[\mathrm{SA}[j]] = \mathrm{LCP}[j]$ for $j \in [1..n]$. Then $\mathrm{PLCP}[i] = \mathrm{lcp}(i, \Phi[i])$ for all $i \in [0..n)$. Table 2 shows example LCP and PLCP arrays. The last two rows in Table 2 illustrate the following property of the PLCP array, which is the basis of all efficient algorithms for LCP array construction.

▶ **Lemma 1** ([13]). *Let $i, j \in [0..n)$. If $i \leq j$, then $i + \mathrm{PLCP}[i] \leq j + \mathrm{PLCP}[j]$. Symmetrically, if $\Phi[i] \leq \Phi[j]$, then $\Phi[i] + \mathrm{PLCP}[i] \leq \Phi[j] + \mathrm{PLCP}[j]$.*

The *succinct PLCP array* [31] $\mathrm{PLCP}_{\mathrm{succ}}[0..2n)$ represents the PLCP array using $2n$ bits. Specifically, $\mathrm{PLCP}_{\mathrm{succ}}[j] = 1$ if $j = 2i + \mathrm{PLCP}[i]$ for some $i \in [0..n)$, and $\mathrm{PLCP}_{\mathrm{succ}}[j] = 0$ otherwise. Notice that the value $2i + \mathrm{PLCP}[i]$ must be unique for each i by Lemma 1. Any lcp value can be recovered by the equation $\mathrm{PLCP}[i] = \mathrm{select}(\mathrm{PLCP}_{\mathrm{succ}}, i) - 2i$, where $\mathrm{select}(\mathrm{PLCP}_{\mathrm{succ}}, i)$ returns the location of the $(i+1)^{\mathrm{th}}$ 1-bit in $\mathrm{PLCP}_{\mathrm{succ}}$. The select query can be answered in $\mathcal{O}(1)$ time given a precomputed data structure of $o(n)$ bits [5, 24].

For $q \geq 1$, the *sparse PLCP array* $\mathrm{PLCP}_q[0..\lceil n/q \rceil)$ is defined by $\mathrm{PLCP}_q[i] = \mathrm{PLCP}[iq]$, i.e., it contains every qth entry of PLCP. It can be used as a compact representation of the full PLCP array because the other entries can be bounded using the following lemma.

▶ **Lemma 2** ([16]). *For any $i \in [0..n)$, let $a = \lfloor i/q \rfloor$ and $b = i \bmod q$, so that $i = aq + b$. If $(a+1)q \leq n-1$, then $\mathrm{PLCP}_q[a] - b \leq \mathrm{PLCP}[i] \leq \mathrm{PLCP}_q[a+1] + q - b$. If $(a+1)q > n-1$, then $\mathrm{PLCP}_q[a] - b \leq \mathrm{PLCP}[i] \leq n - i \leq q$.*

Let the slack, denoted by $\mathrm{slack}_q(i)$, be the difference of the upper and lower bounds for $\mathrm{PLCP}[i]$ given by Lemma 2. Although there is no non-trivial bound on an individual slack, the sum of the slacks is bounded by the following lemma.

▶ **Lemma 3** ([16]). $\sum_{i\in[0..n)} \mathrm{slack}_q(i) \le (q-1)n + q^2$.

The *Burrows–Wheeler transform* $\mathrm{BWT}[0..n]$ of X is defined by $\mathrm{BWT}[i] = \mathsf{X}[\mathrm{SA}[i]-1]$ if $\mathrm{SA}[i] > 0$ and otherwise $\mathrm{BWT}[i] = \$$, where $\$$ is a special symbol that does not appear in the text. We say that that an lcp value $\mathrm{LCP}[i] = \mathrm{PLCP}[\mathrm{SA}[i]]$ is *reducible* if $\mathrm{BWT}[i] = \mathrm{BWT}[i-1]$ and *irreducible* otherwise. The significance of reducibility is summarized in the following two lemmas.

▶ **Lemma 4** ([16]). *If* $\mathrm{PLCP}[i]$ *is reducible, then* $\mathrm{PLCP}[i] = \mathrm{PLCP}[i-1]-1$ *and* $\Phi[i] = \Phi[i-1]+1$.

▶ **Lemma 5** ([16, 14]). *The sum of all irreducible lcp values is* $\le n\log n$.

3 Basic Semi-External Algorithms

We will next describe the two semi-external algorithms introduced by Kärkkäinen, Manzini and Puglisi [16]. The semi-external versions are only briefly mentioned in [16] but they are essentially the same as the space-efficient versions. Both algorithms need enough RAM for the text and for either the succinct or the sparse PLCP array. The larger data structures SA and LCP are stored on disk in the semi-external algorithms.

The first algorithm, which we call SPARSE-Φ, performs the following main steps:

1. Compute a sparse version Φ_q of the Φ-array defined so that $\mathrm{PLCP}_q[i] = \mathrm{lcp}(qi, \Phi_q[i])$.
2. Compute PLCP_q using Φ_q. When computing $\mathrm{PLCP}_q[i]$, we take advantage of the fact that $\mathrm{PLCP}_q[i] \ge \mathrm{PLCP}_q[i-1]-q$ (Lemma 1).
3. Compute LCP using PLCP_q based on Lemma 2.

The first two steps need $\mathcal{O}(n)$ time and the third step $\mathcal{O}(qn)$ time. The pseudocode for SPARSE-Φ is given in Figure 1. All accesses to SA and LCP are sequential allowing them to be stored on disk.

The second algorithm is called SUCCINCTIRREDUCIBLE and has the following steps:

1. Compute irreducible lcp values and store them in the succinct PLCP array $\mathrm{PLCP}_{\mathrm{succ}}$.
2. Compute the reducible lcp values using Lemma 4 and store them in $\mathrm{PLCP}_{\mathrm{succ}}$.
3. Compute LCP from $\mathrm{PLCP}_{\mathrm{succ}}$.

During steps 1 and 2 we also need a bitvector $\mathsf{R}[0..n]$ for marking the irreducible positions in PLCP. The first step needs $\mathcal{O}(n\log n)$ time by Lemma 5, and the other steps need $\mathcal{O}(n)$ time. Again, all accesses to SA and LCP are sequential.

The algorithm also uses BWT (line 4). Since $\mathrm{BWT}[i] = \mathsf{X}[\mathrm{SA}[i]-1]$ (unless $\mathrm{SA}[i] = 0$), we can compute the values on-the-fly when the text X is in RAM as was done in [16]. However, we want to get rid of the requirement that the text is in RAM and instead assume that the BWT is available on disk and is accessed sequentially during the algorithm.

4 Moving Text into External Memory

The semi-external algorithms described above need to have the text X in RAM. While a single comparison of two suffixes is sequential, the first character accesses in each comparison are random accesses often enough so that any straightforward way to deal with texts larger than RAM will not work. Instead, the computation in the steps involving text accesses have to be completely reorganized as will be described in this section. Here we still assume that Φ_q, PLCP_q, $\mathrm{PLCP}_{\mathrm{succ}}$ and R fit in RAM; handling them in a fully external memory algorithm is covered in the next section.

```
SPARSE-Φ
 — Step 1: Compute $\Phi_q$
1:  for $i \leftarrow 1$ to $n$ do
2:      if $\mathrm{SA}[i] \bmod q = 0$ then
3:          $\Phi_q[\mathrm{SA}[i]/q] \leftarrow \mathrm{SA}[i-1]$
 — Step 2: Compute $\mathrm{PLCP}_q$ using $\Phi_q$
4:  $\ell \leftarrow 0$
5:  for $i \leftarrow 0$ to $\lceil n/q \rceil - 1$ do
6:      $j \leftarrow \Phi_q[i]$
7:      while $\mathsf{X}[qi+\ell] = \mathsf{X}[j+\ell]$ do
8:          $\ell \leftarrow \ell+1$
9:      $\mathrm{PLCP}_q[i] \leftarrow \ell$
10:     $\ell \leftarrow$ max $(\ell - q, 0)$
 — Step 3: Compute LCP using $\mathrm{PLCP}_q$
11: for $i \leftarrow 1$ to $n$ do
12:     $j \leftarrow \mathrm{SA}[i-1]$
13:     $k \leftarrow \mathrm{SA}[i]$
14:     $k' \leftarrow \lfloor k/q \rfloor$
15:     $\ell \leftarrow \mathrm{PLCP}_q[k'] - (k - k'q)$
16:     $\ell \leftarrow$ max $(\ell, 0)$
17:     while $\mathsf{X}[k+\ell] = \mathsf{X}[j+\ell]$ do
18:         $\ell \leftarrow \ell+1$
19:     $\mathrm{LCP}[i] \leftarrow \ell$
```

```
SUCCINCTIRREDUCIBLE
 — Step 1: Compute irreducible lcps
1:  $\mathrm{PLCP}_{\mathrm{succ}}[0..2n) \leftarrow (0, 0, \ldots, 0)$
2:  $\mathsf{R}[0..n] \leftarrow (0, 0, \ldots, 0, 1)$
3:  for $i \leftarrow 1$ to $n$ do
4:      if $\mathrm{BWT}[i] \neq \mathrm{BWT}[i-1]$ then
5:          $j \leftarrow \mathrm{SA}[i-1]$; $k \leftarrow \mathrm{SA}[i]$
6:          $\mathsf{R}[k] \leftarrow 1$
7:          $\ell \leftarrow 0$
8:          while $\mathsf{X}[k+\ell] = \mathsf{X}[j+\ell]$ do
9:              $\ell \leftarrow \ell+1$
10:         $\mathrm{PLCP}_{\mathrm{succ}}[2k + \ell] \leftarrow 1$
 — Step 2: Fill in reducible lcps
11: $i \leftarrow 0$; $j \leftarrow 0$
12: while $i < n$ do
13:     while $\mathrm{PLCP}_{\mathrm{succ}}[j] = 0$ do $j \leftarrow j + 1$
      — Now $j = \mathrm{PLCP}[i] + 2i$
14:     $i \leftarrow i + 1$; $j \leftarrow j + 1$
15:     while $\mathsf{R}[i] = 0$ do
16:         $\mathrm{PLCP}_{\mathrm{succ}}[j] \leftarrow 1$
17:         $i \leftarrow i + 1$; $j \leftarrow j + 1$
 — Step 3: Compute LCP from $\mathrm{PLCP}_{\mathrm{succ}}$
18: Construct select-structure for $\mathrm{PLCP}_{\mathrm{succ}}$
19: for $j \leftarrow 1$ to $n$ do
20:     $i \leftarrow \mathrm{SA}[j]$
21:     $\mathrm{LCP}[j] \leftarrow \mathrm{select}(\mathrm{PLCP}_{\mathrm{succ}}, i) - 2i$
```

Figure 1 Two semi-external algorithms.

We will analyze the algorithms in the standard external memory model [34], where the memory system consists of a fast random access memory (RAM) of size M and a slow (disk) memory of unbounded size divided into blocks of size B, both measured in units of $\mathcal{O}(\log n)$ bits. We are primarily interested in the I/O complexity which measures the number of blocks read from or written to disk. Notice that we can fit $\mathcal{O}(M \log_\sigma n)$ characters in RAM and $\mathcal{O}(B \log_\sigma n)$ characters in a disk block.

All text accesses in both algorithms happen in loops where the goal is to compute $\mathrm{lcp}(i, \Phi[i])$ for some i. The basic idea is to divide the text into segments of size at most $m = \mathcal{O}(M \log_\sigma n)$ such that two segments fit in RAM. For each pair of segments at a time, we load them into RAM and compute $\mathrm{lcp}(i, \Phi[i])$ for each i such that i and $\Phi[i]$ are in those two segments. Further details we will consider separately for each step involving text accesses.

Step 1 in SuccinctIrreducible. The computation on lines 1–10 in SUCCINCTIRREDUCIBLE including the text access loop on line 8 is replaced by the following steps:

1.1. Scan SA and BWT to form a pair $(i, \Phi[i])$ for each i such that $\mathrm{PLCP}[i]$ is irreducible. The pairs are written to disk where there is a separate file for each pair of text segments. Simultaneously, compute the bitvector R, which is kept in RAM during the step and written to disk at the end of the step.

1.2. For each pair of text segments, load them to RAM and compute $\mathrm{PLCP}[i] = \mathrm{lcp}(i, \Phi[i])$ for each pair $(i, \Phi[i])$ obtained from the associated file. For each computed $\mathrm{PLCP}[i]$, we store the value $2i + \mathrm{PLCP}[i]$ to disk.

1.3. With $\mathrm{PLCP}_{\mathrm{succ}}$ in RAM, read the output of the previous step and set the corresponding bits of $\mathrm{PLCP}_{\mathrm{succ}}$ to 1. Then read R from disk.

The rest of the algorithm is as in Section 3. The total number of additional I/Os resulting from this procedure is $\mathcal{O}((n/\log_\sigma n)^2/(MB))$.

An additional detail to consider in step 1.2 is that although i and $\Phi[i]$ are in the segments in RAM, the common prefix of those suffixes may continue beyond the end of the segments. To deal with this, we also keep the first $B\log_\sigma n$ symbols after the segments in RAM. These are called *overflow buffers*. If the comparison continues beyond the overflow buffers, we read the relevant parts of the text from disk sequentially. Since the total length of the irreducible lcps is $\mathcal{O}(n\log n)$, the additional number of I/Os from this is never more than $\mathcal{O}((n\log n)/(B\log_\sigma n)) = \mathcal{O}((n\log\sigma)/B)$.

Step 3 in Sparse-Φ. The loop on lines 11-19 in Sparse-Φ including the text access loop on line 17 is replaced with the following steps:

3.1. Scan SA to generate all $(i, \Phi[i])$ pairs. For each pair use PLCP_q (stored in RAM) to compute the lower bound $\ell_{\min}$ (and the upper bound $\ell_{\max}$) for PLCP$[i]$. Write the pair $(i+\ell_{\min}, \Phi[i]+\ell_{\min})$ to disk, where there is a separate file for each pair of text segments.

3.2. For each pair of text segments, load them to RAM and compute $\mathrm{lcp}(i,j)$ for each pair (i,j) obtained from the associated file. The resulting value $\mathrm{lcp}(i,j)$ is written to disk to a separate file for each pair of text segments. The order of the lcp values in the output file must be the same as the order of the pairs in the input file.

3.3. Scan SA to generate all $(i, \Phi[i])$ pairs. For each pair, compute $\ell_{\min}$ as in step 1 and read the value $\ell' = \mathrm{lcp}(i+\ell_{\min}, \Phi[i]+\ell_{\min})$ from the appropriate file. Then $\mathrm{PLCP}[i] = \ell_{\min}+\ell'$ is the next value in the LCP array.

The total number of additional I/Os from reading text segments is $\mathcal{O}((n/\log_\sigma n)^2/(MB))$.

Again, we have to deal with lcp comparisons continuing beyond the end of the segments in step 3.2. In step 3.1, we use the upper bound $\ell_{\max}$ to determine whether such an overflow is possible, and if it is, we generate additional pairs/triples for each possible boundary crossing. For example, if for some $\ell' \in [\ell_{\min}..\ell_{\max}]$ we find out that $i+\ell'$ is at a segment boundary, we generate the triple $(i+\ell', \Phi[i]+\ell', \ell_{\max}-\ell')$. The third value is used as an upper bound for the length of the comparison, which might be needed in the case where $\mathrm{lcp}(i,\Phi[i]) < \ell'$. Otherwise, the triple is treated as a normal pair in step 3.2. All the comparisons in step 3.2 end at a segment boundary. In step 3.3, we generate pairs and triples as in step 3.1, read the corresponding lcp values from the appropriate files, and combine them to obtain the final lcp value. The total number of the extra triples is at most $\mathcal{O}(n + qn/(M\log_\sigma n))$. Also notice that the total number of character comparisons is still bounded by $\mathcal{O}(qn)$.

Step 2 in Sparse-Φ. The third and final place with a text access loop is on line 7 in algorithm Sparse-Φ. In this case, the text segment size is $2m$ and only one segment is kept in RAM while the rest of the text is scanned sequentially. The loop on lines 5–10 is replaced with the following steps:

2.1. Scan Φ_q to generate all pairs $(i, \Phi[i])$ such that i is a multiple of q. Write the pairs to disk into the file associated with the text segment that contains $\Phi[i]$. Notice that the pairs in each file are sorted by i.

2.2. For each segment, load the segment into RAM. Read the pairs $(i, \Phi[i])$ from the associated file while simultaneously scanning the text so that the position $\mathsf{X}[i]$ is reached when the pair $(i, \Phi[i])$ is processed. For each pair, compute $\mathrm{lcp}(i, \Phi[i])$ and write it to disk into a separate file for each segment. When computing $\ell = \mathrm{lcp}(i,\Phi[i])$ we use the fact that $\ell \geq \mathrm{lcp}(i', \Phi[i']) - (i-i')$, where $(i', \Phi[i'])$ is the pair processed just previously. This ensures that the text scan never needs to backtrack.

2.3. Scan Φ_q and for each $i \in [0..\lceil n/q \rceil)$ read $\mathrm{PLCP}_q[i]$ from the file associated with the text segment containing $\Phi_q[i]$.

The total number of additional I/Os from scanning the text is $\mathcal{O}((n/\log_\sigma n)^2/(MB))$.

Here too, we have the possibility that the position $\Phi[i] + \ell$ moves beyond the end of the segment during the comparison. We deal with this again by having an overflow buffer of $\mathcal{O}(B \log_\sigma n)$ characters and by reading text from disk when the overflow buffer is not enough. Then the extra I/Os for reading the positions $\Phi[i] + \ell$ is never more than the regular I/Os for reading the positions $i + \ell$.

As a final note, both algorithms distribute pairs into $\mathcal{O}(s^2)$ files at some point, where $s = \mathcal{O}(n/(M \log_\sigma n))$ is the number of segments. To do this efficiently we need to have enough RAM for $\mathcal{O}(s^2)$ buffers of size $\mathcal{O}(B)$ each, which means that we must have $s = \mathcal{O}(\sqrt{M/B})$ and thus $n = \mathcal{O}\left(M\sqrt{M/B}\log_\sigma n\right)$. We can get rid of this constraint by doing the distribution in multiple rounds. That is, we first distribute the pairs into $\mathcal{O}(M/B)$ files and then those files are divided into smaller files and so on. The I/O complexity of the multiround distribution is the same as for external memory sorting, $\mathcal{O}((n/B)\log_{M/B}(n/B))$, and the time complexity is $\mathcal{O}(n \log_{M/B}(n/B))$.

5 Fully External Memory Algorithms

Let us now complete the transformation of the semi-external algorithms into external memory algorithms by describing how to deal with Φ_q, PLCP_q, $\mathrm{PLCP}_{\mathrm{succ}}$ and R.

Consider first Sparse-Φ. We set $q = \min\{n/M, M \log_\sigma n\}$. This choice ensures that the number of extra triples generated in Step 3 is $\mathcal{O}(n + qn/(M \log_\sigma n)) = \mathcal{O}(n)$. When $n \le M^2 \log_\sigma n$, we have $q = \Theta(n/M)$ so that Φ_q and PLCP_q fit in RAM and the algorithm is exactly as described above. When $n > M^2 \log_\sigma n$, we divide Φ_q and PLCP_q into $s = \mathcal{O}(n/(M^2 \log_\sigma n))$ segments that fit in RAM. In Steps 1 and 3, instead of scanning SA once, we scan it s times, once for each segment. Step 3 also produces s subsequences of LCP which are then merged with the help of one more scan of SA. The additional I/O from the extra scans is $\mathcal{O}(n^2/(M^2 B \log_\sigma n))$, which is less than the I/O for text scanning (assuming $M = \Omega(\log_\sigma n)$).

The time complexity has three main components: $\mathcal{O}(qn)$ time for comparing suffixes, $\mathcal{O}(n^2/(M(\log_\sigma n)^2))$ for loading text segments and scanning the text, and $\mathcal{O}(n \log_{M/B}(n/B))$ time for multiround distribution. This gives the following result.

▶ **Theorem 6.** *Given a text of length n over an integer alphabet $[0..\sigma)$ and its suffix array, the associated* LCP *array is computed by* Sparse-Φ *in*

$$\mathcal{O}\left(\min\left\{\frac{n^2}{M}, nM\log_\sigma n\right\} + \frac{n^2}{M(\log_\sigma n)^2} + n\log_{\frac{M}{B}}\frac{n}{B}\right) = \mathcal{O}\left(\frac{n^2}{M} + n\log_{\frac{M}{B}}\frac{n}{B}\right) \text{ time}$$

and $\mathcal{O}\left(\frac{n^2}{MB(\log_\sigma n)^2} + \frac{n}{B}\log_{\frac{M}{B}}\frac{n}{B}\right)$ *I/Os using* $\mathcal{O}(n/B)$ *blocks of disk space.*

Consider then SuccinctIrreducible. Since we cannot make $\mathrm{PLCP}_{\mathrm{succ}}$ and R arbitrarily small, we must be able to handle them even if they do not fit in RAM. We do this by dividing them into segments that are small enough. Consider first the computation of R. While scanning BWT and SA we create a list of irreducible positions, which are then distributed into files corresponding to segments of R. Then each segment of R can be computed by scanning the corresponding file. Similarly, the irreducible bits in $\mathrm{PLCP}_{\mathrm{succ}}$ are set one segment at a time by reading the $2i + \mathrm{PLCP}[i]$ values from disk, which are now in separate files for each segment. The reducible bits in $\mathrm{PLCP}_{\mathrm{succ}}$ can be easily set by scanning $\mathrm{PLCP}_{\mathrm{succ}}$ and R simultaneously. None of this increases the complexity of the algorithm.

Finally, the last stage of SuccinctIrreducible is performed as follows when $\mathrm{PLCP}_{\mathrm{succ}}$ does not fit in RAM:

3.1. Scan SA and store each value $\mathrm{SA}[i]$ into the file corresponding to the $\mathrm{PLCP}_{\mathrm{succ}}$ segment that contains the $(\mathrm{SA}[i] + 1)^{\mathrm{th}}$ 1-bit in $\mathrm{PLCP}_{\mathrm{succ}}$.

3.2. For each segment of $\mathrm{PLCP}_{\mathrm{succ}}$ read the $\mathrm{SA}[i]$ values from the corresponding file, compute $\mathrm{LCP}[i]$ by a select query, and write it to a separate file for each segment.

3.3. Scan SA and for each element $\mathrm{SA}[i]$ determine which segment file contains $\mathrm{LCP}[i]$ and move it to the final output file.

Again, none of this increases the complexity of the algorithm.

The following theorem summarizes the complexities of SuccinctIrreducible. The $\mathcal{O}(n \log n)$ and $\mathcal{O}((n \log \sigma)/B)$ terms come from the irreducible lcp comparisons.

▶ **Theorem 7.** *Given a text of length n over an integer alphabet $[0..\sigma)$ and its suffix array and BWT, the associated LCP array is computed by* SuccinctIrreducible *in*

$$\mathcal{O}\left(\frac{n^2}{M(\log_\sigma n)^2} + n \log n\right) \text{ time and } \mathcal{O}\left(\frac{n^2}{MB(\log_\sigma n)^2} + \frac{n \log \sigma}{B} + \frac{n}{B} \log_{\frac{M}{B}} \frac{n}{B}\right) \text{ I/Os}$$

using $\mathcal{O}(n/B)$ blocks of disk space.

In practice, we have noticed that the n select queries performed at the last stage of SuccinctIrreducible often dominate the time. In the implementation, we have replaced $\mathrm{PLCP}_{\mathrm{succ}}$ with the plain PLCP array, but only in the last stage. That is, instead of loading a segment of $\mathrm{PLCP}_{\mathrm{succ}}$ into RAM, we construct a segment of PLCP in RAM by reading a part of $\mathrm{PLCP}_{\mathrm{succ}}$ from disk. Then we use simple accesses to PLCP instead of select queries on $\mathrm{PLCP}_{\mathrm{succ}}$. This modification does not affect the time or I/O complexities of the algorithm.

As a final note, SuccinctIrreducible needs the BWT. Any suffix array construction algorithm can be modified to compute the BWT too with little overhead by storing $\mathrm{BWT}[i]$ together with $\mathrm{SA}[i]$. Since the algorithm has to access $\mathsf{X}[\mathrm{SA}[i]]$ at some point, we can compute $\mathrm{BWT}[i]$ at the same time. We have also implemented a simple external memory algorithm for computing BWT from SA and show its performance in the next section.

6 Experimental Results

Algorithms. We performed experiments using the following algorithms:

- LCPscan, the fastest external-memory LCP array construction algorithm in previous studies [13]. The number of rounds of partial processing in LCPscan was set to 4, as this gives a similar peak disk space usage ($\sim 16n$ bytes) to the new algorithms presented in this paper (see [13] for more details). In our experiments LCPscan serves as a baseline.
- SE-SΦ, the semi-external version of the Sparse-Φ algorithm described by Kärkkäinen et al. [16] (see also Section 3 of this paper).
- EM-SΦ, the fully external-memory version of the Sparse-Φ algorithm described in Sections 3–5. The algorithm is the first contribution of this paper.
- EM-PLCP, the external-memory version of SuccinctIrreducible algorithm described in this paper restricted to perform only Step 1 and 2, i.e., the algorithm produces the $\mathrm{PLCP}_{\mathrm{succ}}$ array but does not convert it to LCP array. We separately consider the construction of $\mathrm{PLCP}_{\mathrm{succ}}$ because first, for some applications computing $\mathrm{PLCP}_{\mathrm{succ}}$ is sufficient

Table 3 Statistics of data used in the experiments. In addition to basic parameters, we show the percentage of irreducible lcp values among all lcp values (expression $100r/n$, where r denotes the number of irreducible lcps) and the average length of the irreducible lcp value (Σ_r/r, where Σ_r is the sum of all irreducible lcps).

Name	$n/2^{30}$	σ	$100r/n$	Σ_r/r
kernel	128.0	229	0.09	1494.76
geo	128.1	211	0.15	1221.49
wiki	128.7	213	16.71	29.40
dna	128.0	6	18.46	23.79
dna	512.0	6	16.13	27.25
debruijn	128.0	2	99.26	35.01

and avoiding the conversion to LCP array is a big time save, and second, this allows us to visualize differences in the methods that convert $\mathrm{PLCP}_{\mathrm{succ}}$ to LCP. Note: small files than can be handled by the original semi-external version of SuccinctIrreducible are processed using the original algorithm from [16]. The increase in I/O and runtime that occurs when we switch to fully-external procedure is discussed in one of the experiments.

- SE-SI, the semi-external version of the SuccinctIrreducible algorithm. It first runs EM-PLCP and then converts $\mathrm{PLCP}_{\mathrm{succ}}$ (held in RAM) to LCP using select queries as originally described [16]. To implement the select queries, we use the variant of the *darray* data structure [29] described in [16]. We set the darray overhead to 6.25% as we did not observe a significant speedup from using more space (e.g., increasing the overhead to 50% speeds up select queries only by about 10%).
  This algorithm is essentially identical to the original semi-external algorithm in [16] when the text and the bitvectors $\mathsf{R}[0..n]$ and $\mathrm{PLCP}_{\mathrm{succ}}[0..2n)$ fit in RAM, but it can also extend beyond that limit since it uses EM-PLCP. It is still a semi-external algorithm though as it needs to have enough RAM for $\mathrm{PLCP}_{\mathrm{succ}}[0..2n)$ in the last stage.
- EM-SI, the fully-external version of the SuccinctIrreducible algorithm described in Sections 3–5. It first uses EM-PLCP to compute $\mathrm{PLCP}_{\mathrm{succ}}$ and then computes LCP using plain accesses to PLCP segments instead of select queries on $\mathrm{PLCP}_{\mathrm{succ}}$ segments as described in Section 5. The time and I/O of EM-PLCP is included in the runtime and I/O volume of EM-SI. Together with EM-PLCP this algorithm is the second contribution of this paper.

All algorithms use 8 bits to represent characters and 40 bits to represent integers. The implementations of all LCP array construction algorithms used in experiments are available at `http://www.cs.helsinki.fi/group/pads/`.

Datasets. For the experiments we used the following files varying in the number of repetitions and alphabet size (see Table 3 for some statistics):

- kernel: a concatenation of ∼10.7 million source files from over 300 versions of Linux kernel [1]. This is an example of highly repetitive file;
- geo: a concatenation of all versions (edit history) of Wikipedia articles about all countries and 10 largest cities in the XML format. The resulting file is also highly repetitive;

[1] `http://www.kernel.org/`

- wiki: a concatenation of English Wiki dumps (Wikipedia, Wikisource, Wikibooks, Wikinews, Wikiquote, Wikiversity, and Wikivoyage [2]) dated 20160203 in XML;
- dna: a collection of DNA reads (short fragments produced by a sequencing machine) from multiple human genomes[3] filtered from symbols other than $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}, \mathtt{N}\}$ and newline;
- debruijn: a binary De Bruijn sequence of order k is an artificial sequence of length $2^k + k - 1$ than contains all possible binary k-length substrings. A file of length n is obtained as a prefix of a De Bruijn sequence of order $\lceil \log n \rceil$. It contains nearly n irreducible lcps with total length of nearly $n \log n$ (see [16, Lemma 5]) which is the worst case for LCPscan, SE-SI and EM-SI algorithms.

Setup. We performed experiments on a machine equipped with two six-core 1.9 GHz Intel Xeon E5-2420 CPUs with 15 MiB L3 cache and 120 GiB of DDR3 RAM. For experiments we limited the RAM in the system (with the kernel boot flag) to 4 GiB and all algorithms were allowed to use 3.5 GiB. The machine had 6.8 TiB of free disk space striped with RAID0 across four identical local disks achieving a (combined) transfer rate of about 480 MiB/s.

The OS was Linux (Ubuntu 12.04, 64bit) running kernel 3.13.0. All programs were compiled using `g++` version 4.9.2 with `-O3 -DNDEBUG` options. All tested LCP array construction algorithms are sequential, i.e., only a single thread of execution was used for computation. In the last experiment we used parallel algorithms to compute SA and BWT in order to demonstrate the performance of currently fastest methods but those measurements have no bearing on the findings of this paper. All reported runtimes are wallclock (real) times.

Experiments. In the first experiment we compare the scalability of the new external-memory LCP array construction algorithms described in this paper (EM-SΦ, EM-SI) to LCPscan. We executed the algorithms on increasing length prefixes of testfiles using 3.5 GiB of RAM and measured the runtime and the I/O volume.

The results are presented in Figure 2. The performance of EM-SI, similarly to LCPscan, is related to the number of irreducible lcp values (see Table 3). However, avoiding the external-memory sorting gives EM-SI a consistent speed and I/O advantage (of about $60n$ bytes) over LCPscan. The EM-SI algorithm is at least two times faster than LCPscan and even more on highly repetitive data. The computation time can be further reduced by 30–65% if one stops at the $\mathrm{PLCP}_{\mathrm{succ}}$ array. Overall, if the BWT is given as input alongside the text and the suffix array, EM-SI is the fastest way to compute the LCP array. Even if we include the cost of the standalone construction of BWT from the suffix array, the algorithm still outperforms LCPscan.

The performance of EM-SΦ is also related to the number of irreducible lcp values though in a different way than EM-SI. Whenever during step 3.1 the exact value of $\mathrm{PLCP}[i]$ can be deduced from the lower/upper bounds on $\mathrm{PLCP}[i]$ (i.e., $\ell_{\min} = \ell_{\max}$), the algorithm does not write any data to disk. All other pairs are written to disk and processed in step 3.2, which usually dominates the runtime. From Lemma 4 we expect the number of skipped pairs to be high if the number of irreducible lcp values is low, and thus the algorithm runs faster and uses less I/O (by about $20n$ bytes) on kernel and geo testfiles. However, even on the non-repetitive data, EM-SΦ is still about two times faster than LCPscan. Furthermore, when BWT is not available it usually also outperforms EM-SI, making it the algorithm of choice in this case.

[2] `http://dumps.wikimedia.org/`
[3] `http://www.1000genomes.org/`

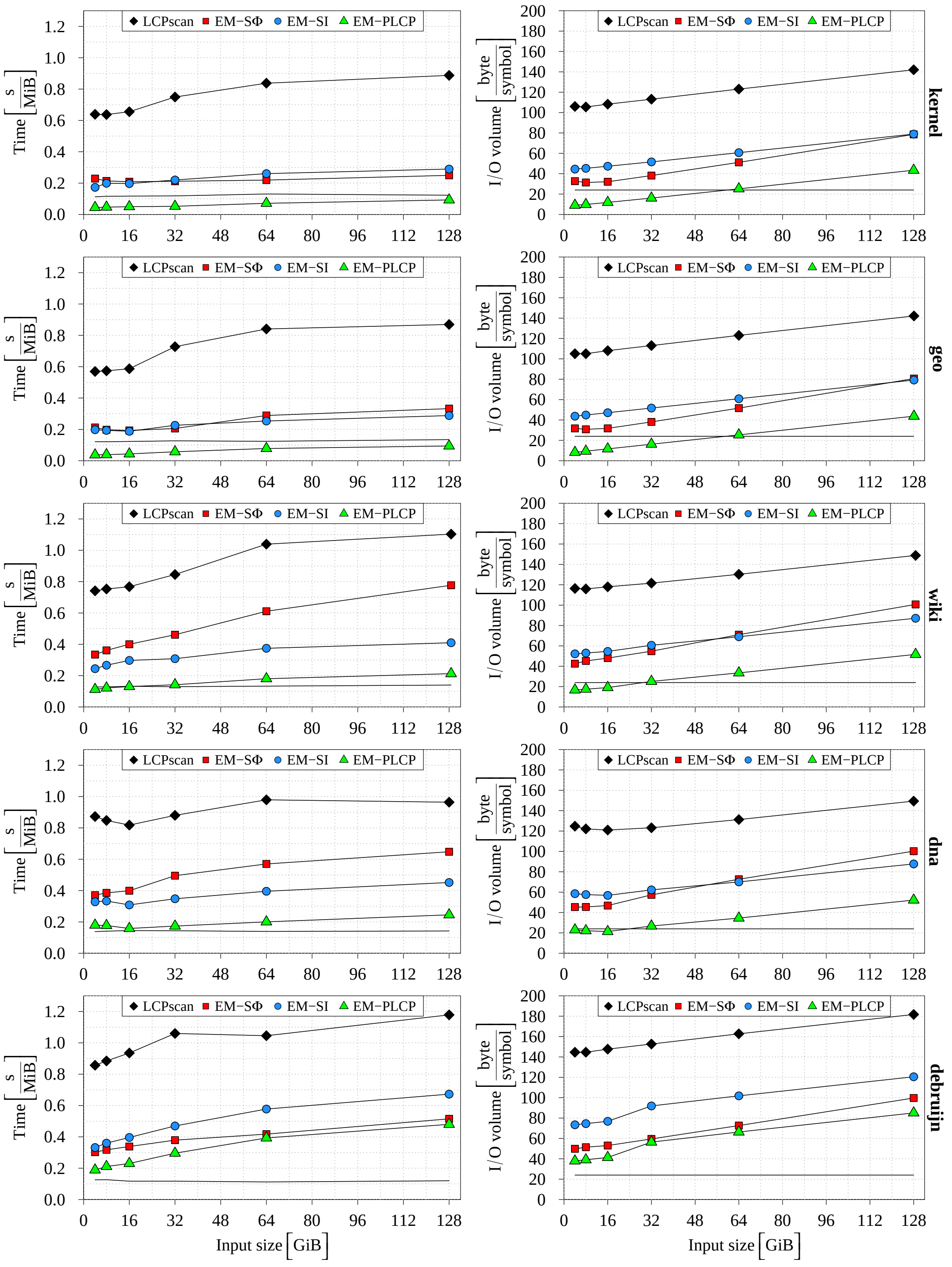


Figure 2 Comparison of the runtime (left; in seconds per MiB of input text) and I/O volume (right; in bytes per input symbol) of the new external-memory LCP array construction algorithms (EM-SΦ, EM-SI) to LCPscan. All algorithms were allowed to use 3.5 GiB of RAM. The unlabeled curve shows the runtime and I/O volume of the standalone external-memory construction of BWT from SA.

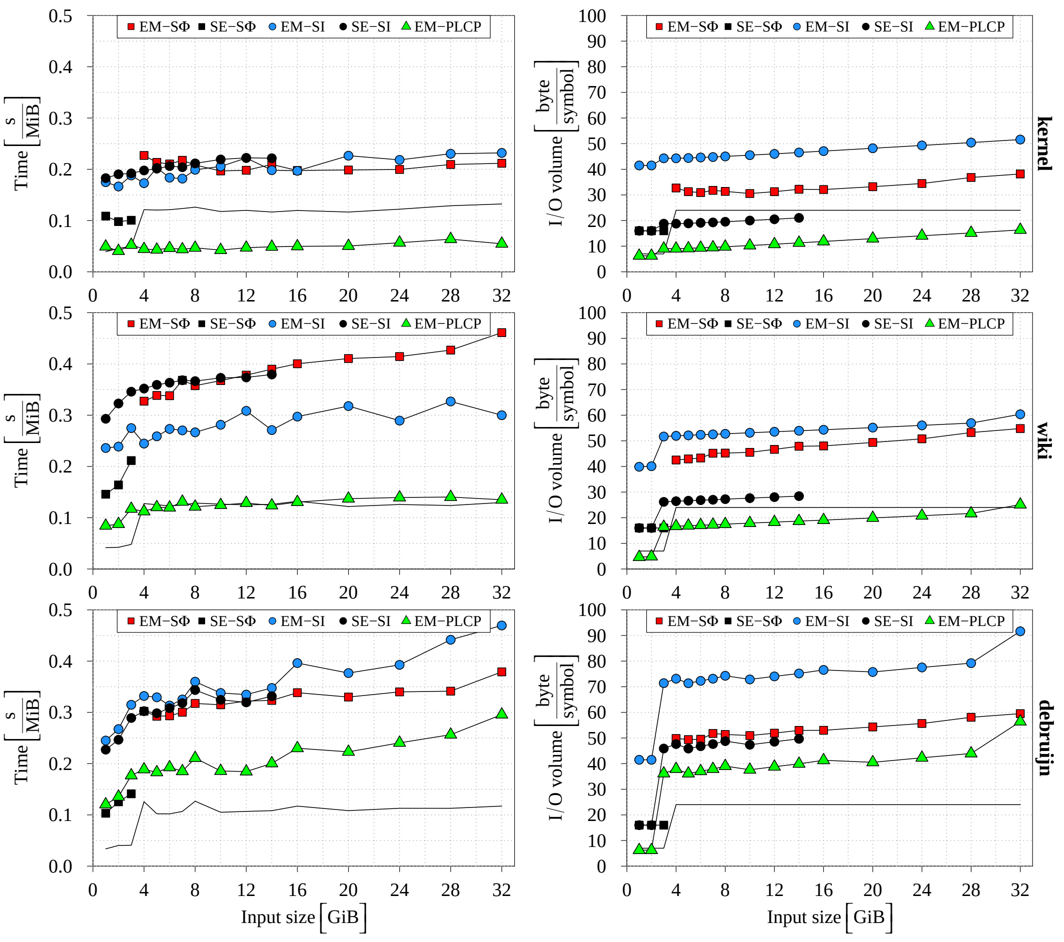


Figure 3 Comparison of the runtime and I/O volume of the new external-memory LCP array construction algorithms (EM-SΦ, EM-SI), to their original semi-external counterparts (SE-SΦ, SE-SI) [16]. The setup is analogous to Figure 2, i.e., all algorithms are using 3.5 GiB of RAM. The unlabeled curve shows the runtime and I/O volume of the standalone external-memory construction of BWT from SA (when the text fits in RAM, we run a semi-external version that needs less I/O).

In the second experiment we compare the EM-SΦ and EM-SI algorithms to their semi-external counterparts SE-SΦ and SE-SI. More precisely, we analyse the transition between semi-external and fully-external algorithms in terms of runtime and I/O volume.

The results are given in Figure 3. First, observe that at the point where we no longer can accommodate the text and bitvectors $\mathrm{PLCP}_{\mathrm{succ}}$ and R in RAM (i.e., between 2 and 3 GiB prefixes) the I/O volume of EM-PLCP (and thus also SE-SI and EM-SI) increases proportionally to the number of irreducible lcp values (up to $30n$ bytes for debruijn). The extra I/O accounts mostly for scanning and does not significantly affect the runtime. Second, note the difference of $25n$ bytes in the I/O volume of SE-SI and EM-SI. While SE-SI uses the original semi-external method that performs select-queries over $\mathrm{PLCP}_{\mathrm{succ}}$ bitvector (kept in RAM) to convert $\mathrm{PLCP}_{\mathrm{succ}}$ into LCP array [16], EM-SI uses the fully external-memory (and thus more I/O-demanding) method that does not require $\mathrm{PLCP}_{\mathrm{succ}}$ to fit in RAM, and furthermore, replaces the select-queries with simple lookups (see Section 5). The I/O increase is however compensated by faster computation and the fully-external method is either comparable (kernel, debruijn) or faster (wiki) than the semi-external method.

Table 4 Experimental results on the 512 GiB instance of dna testfile using 120 GiB of RAM. The disk usage column gives a peak disk space usage including the input and output of the given algorithm. pEM-BWT is a simple external-memory algorithm constructing BWT from SA. For comparison with pSAscan we also parallelized the computation in pEM-BWT.

Algorithm	Runtime	I/O volume	Disk usage
pSAscan	2.51 days	16.63 TiB	3.75 TiB
pEM-BWT	0.54 days	12.00 TiB	5.50 TiB
LCPscan	5.04 days	62.22 TiB	6.33 TiB
EM-SI	2.16 days	25.89 TiB	6.12 TiB
EM-SΦ	2.69 days	20.77 TiB	5.50 TiB
EM-PLCP	0.77 days	8.27 TiB	4.36 TiB

The transition between the SE-SΦ algorithm and EM-SΦ shows an increase in I/O by a factor 2–3 at the point where the text no longer fits in RAM. This is due to the fact that SE-SΦ reads the text once while EM-SΦ reads it in steps 2.2 and 3.2. Similarly, while SE-SΦ reads SA once in Step 3, EM-SΦ needs two scans (steps 3.1 and 3.3). Moreover EM-SΦ computes the values $\ell_{\min}$ and $\ell_{\max}$ twice for every processed pair (step 3.1 and 3.3), whereas SE-SΦ does it only once. This causes a slowdown by a factor 1.6–2.3 at the transition point, since these computations involve random accesses to the PLCP_q array and thus attract multiple cache misses. Note however that the increase in runtime and I/O volume is much bigger if we consider the transition from SE-SΦ to LCPscan.

In the last experiment we compare the performance of the new external-memory algorithms EM-SI and EM-SΦ to LCPscan on a full 512 GiB instance of the dna file using all 120 GiB of RAM available on our test machine. Unlike in previous experiments, here we use LCPscan with six rounds of processing since the four-round version ran out of disk space. The performance of both modes is very similar.

The results are given in Table 4. For comparison we also present the resources used by pSAscan [15] – currently the fastest practical way to construct suffix arrays in external memory, and pEM-BWT – a simple parallel BWT-from-SA construction. Note that in all previous experiments the algorithm for computing BWT from SA was sequential. Here we decided to use the parallel version to make it comparable to pSAscan. The results are consistent with previous experiments, i.e., both EM-SI and EM-SΦ are about two times faster than LCPscan. When BWT is available alongside the suffix array, the processing time is smaller for EM-SI. Otherwise, we need to additionally run a separate BWT construction and as a result EM-SΦ has a slight edge over EM-SI. Finally, if one wishes to only compute the PLCP array the processing time of EM-SI is reduced by about 65%.

Lastly, note that the new algorithms use a fairly moderate disk space. For EM-PLCP a peak disk space usage is either achieved after step 1.1 where in addition to input, we have $2r$ integers and the R bitvector stored on disk (recall that r is the number of irreducible lcp values) or after step 2.3 where in addition to input we store the $\mathrm{PLCP}_{\mathrm{succ}}$ bitvector on disk. The resulting disk usage is $7.125n + \max(10r, 0.125n)$ bytes, assuming 40-bit integers. Given that for the input instance we have $10r = 1.61n$ (see Table 3), the peak disk usage is $8.735n$, i.e., 4.36 TiB. For EM-SI we obtain the full LCP array in the last stage and thus the disk usage increases to $7.125n + \max(10r, 5.125n)$ bytes, i.e., $12.25n$ for the tested input. For EM-SΦ the disk usage is either maximized after step 3.1 or at the end of computation and is equal to at most $16n$ bytes (we ignore the space needed for PLCP_q). In practice it can be less due to skipped $(i, \Phi[i])$ pairs (see the discussion above) but it cannot be easily expressed in terms and n and r. In our experiment 55% of all pairs were skipped resulting in a peak disk usage of about $11n$ bytes or 5.5 TiB. We leave details for the full version of the paper.

7 Concluding Remarks

We have described two new external memory algorithms for LCP array construction. Our experiments show that the new algorithms are about two times faster than the state of the art. A common feature of the new algorithms is their avoidance of external-memory sorting.

One of the possible avenues for future work is reducing the disk space usage. In LCPscan this is accomplished by splitting and processing the input text in multiple parts. Similar partitioning techniques can be applied to reduce the disk space usage of the presented algorithms. Although the new algorithms are already quite disk space efficient, with partitioning their peak disk space usage can be guaranteed to be little more than what is needed for the input and the output.

Another possibility for improvement is to use parallelism. Both of the new algorithms are compute-bound rather than I/O-bound in some stages of the computation. Parallel computation in such stages can reduce the running time further.

References

1 M. I. Abouelhoda, S. Kurtz, and E. Ohlebusch. Replacing suffix trees with enhanced suffix arrays. *J. Discrete Algorithms*, 2(1):53–86, 2004. doi:10.1016/S1570-8667(03)00065-0.

2 M. J. Bauer, A. J. Cox, G. Rosone, and M. Sciortino. Lightweight LCP construction for next-generation sequencing datasets. In *Proceedings of the 12th Workshop on Algorithms in Bioinformatics (WABI 2012)*, volume 7534 of *LNCS*, pages 326–337. Springer, 2012. doi:10.1007/978-3-642-33122-0_26.

3 T. Beller, S. Gog, E. Ohlebusch, and T. Schnattinger. Computing the longest common prefix array based on the Burrows-Wheeler transform. *J. Discrete Algorithms*, 18:22–31, 2013. doi:10.1016/j.jda.2012.07.007.

4 T. Bingmann, J. Fischer, and V. Osipov. Inducing suffix and LCP arrays in external memory. In *Proceedings of the 2013 Workshop on Algorithm Engineering and Experiments (ALENEX 2013)*, pages 88–102. SIAM, 2013. doi:10.1137/1.9781611972931.8.

5 D. R. Clark. *Compact Pat Trees*. PhD thesis, University of Waterloo, 1998.

6 F. A. da Louza, G. P. Telles, and C. D. de Aguiar Ciferri. External memory generalized suffix and LCP arrays construction. In *Proceedings of the 24th Annual Symposium on Combinatorial Pattern Matching (CPM 2013)*, volume 7922 of *LNCS*, pages 201–210. Springer, 2013. doi:10.1007/978-3-642-38905-4_20.

7 R. Dementiev, J. Kärkkäinen, J. Mehnert, and P. Sanders. Better external memory suffix array construction. *ACM J. Exp. Algor.*, 12:3.4:1–3.4:24, August 2008. doi:10.1145/1227161.1402296.

8 M. Deo and S. Keely. Parallel suffix array and least common prefix for the GPU. In *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP 2013)*, pages 197–206. ACM, 2013. doi:10.1145/2442516.2442536.

9 S. Gog and E. Ohlebusch. Fast and lightweight LCP-array construction algorithms. In *Proceedings of the 2011 Workshop on Algorithm Engineering and Experiments (ALENEX 2011)*, pages 25–34. SIAM, 2011. doi:10.1137/1.9781611972917.3.

10 G. H. Gonnet, R. A. Baeza-Yates, and T. Snider. New indices for text: Pat trees and Pat arrays. In W. B. Frakes and R. Baeza-Yates, editors, *Information Retrieval: Data Structures & Algorithms*, pages 66–82. Prentice–Hall, 1992.

11 J. Kärkkäinen and D. Kempa. Engineering a lightweight external memory suffix array construction algorithm. In *Proceedings of the 2nd International Conference on Algorithms for Big Data (ICABD 2014)*, volume 1146 of *CEUR Workshop Proceedings*, pages 53–60. CEUR-WS.org, 2014.

12 J. Kärkkäinen and D. Kempa. LCP array construction in external memory. In *Proceedings of the 13th International Symposium on Experimental Algorithms (SEA 2014)*, volume 8504 of *LNCS*, pages 412–423. Springer, 2014. `doi:10.1007/978-3-319-07959-2_35`.

13 J. Kärkkäinen and D. Kempa. LCP array construction in external memory. *J. Exp. Algorithmics*, 21(1):1.7:1–1.7:22, April 2016. `doi:10.1145/2851491`.

14 J. Kärkkäinen, D. Kempa, and M. Piątkowski. Tighter bounds for the sum of irreducible LCP values. In *Proceedings of the 26th Annual Symposium on Combinatorial Pattern Matching (CPM 2015)*, volume 9133 of *LNCS*, pages 316–328. Springer, 2015. `doi:10.1007/978-3-319-19929-0_27`.

15 J. Kärkkäinen, D. Kempa, and S. J. Puglisi. Parallel external memory suffix sorting. In *Proceedings of the 26th Annual Symposium on Combinatorial Pattern Matching (CPM 2015)*, volume 9133 of *LNCS*, pages 329–342. Springer, 2015. `doi:10.1007/978-3-319-19929-0_28`.

16 J. Kärkkäinen, G. Manzini, and S. J. Puglisi. Permuted longest-common-prefix array. In *Proceedings of the 20th Annual Symposium on Combinatorial Pattern Matching (CPM 2009)*, volume 5577 of *LNCS*, pages 181–192. Springer, 2009. `doi:10.1007/978-3-642-02441-2_17`.

17 J. Kärkkäinen and P. Sanders. Simple linear work suffix array construction. In *Proceedings of the 30th International Colloquium on Automata, Languages and Programming (ICALP 2003)*, volume 2719 of *LNCS*, pages 943–955. Springer, 2003. `doi:10.1007/3-540-45061-0_73`.

18 T. Kasai, G. Lee, H. Arimura, S. Arikawa, and K. Park. Linear-time longest-common-prefix computation in suffix arrays and its applications. In *Proceedings of the 12th Annual Symposium on Combinatorial Pattern Matching (CPM 2001)*, volume 2089 of *LNCS*, pages 181–192. Springer, 2001. `doi:10.1007/3-540-48194-X_17`.

19 W. Liu, G. Nong, W. H. Chan, and Y. Wu. Induced sorting suffixes in external memory with better design and less space. In *Proceedings of the 22nd International Symposium on String Processing and Information Retrieval (SPIRE 2015)*, volume 9309 of *LNCS*, pages 83–94. Springer, 2015. `doi:10.1007/978-3-319-23826-5_9`.

20 V. Mäkinen. Compact suffix array – a space efficient full-text index. *Fund. Inform.*, 56(1–2):191–210, 2003.

21 V. Mäkinen, D. Belazzougui, F. Cunial, and A. I. Tomescu. *Genome-Scale Algorithm Design: Biological Sequence Analysis in the Era of High-Throughput Sequencing*. Cambridge University Press, 2015.

22 U. Manber and G. W. Myers. Suffix arrays: a new method for on-line string searches. *SIAM J. Comput.*, 22(5):935–948, 1993. `doi:10.1137/0222058`.

23 G. Manzini. Two space saving tricks for linear time LCP array computation. In *Proceedings of the 14th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2004)*, volume 3111 of *LNCS*, pages 372–383. Springer, 2004. `doi:10.1007/978-3-540-27810-8_32`.

24 I. Munro. Tables. In *Proceedings of the 16th Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 1996)*, volume 1180 of *LNCS*, pages 37–42. Springer, 1996. `doi:10.1007/3-540-62034-6_35`.

25 G. Navarro and V. Mäkinen. Compressed full-text indexes. *ACM Comput. Surv.*, 39(1):article 2, 2007. `doi:10.1145/1216370.1216372`.

26 G. Nong, W. H. Chan, S. Q. Hu, and Y. Wu. Induced sorting suffixes in external memory. *ACM Trans. Inf. Syst.*, 33(3), February 2015. doi:10.1145/2699665.

27 G. Nong, W. H. Chan, S. Zhang, and X. F. Guan. Suffix array construction in external memory using d-critical substrings. *ACM Trans. Inf. Syst.*, 32(1), January 2014. doi:10.1145/2518175.

28 E. Ohlebusch. *Bioinformatics Algorithms: Sequence Analysis, Genome Rearrangements, and Phylogenetic Reconstruction.* Oldenbusch Verlag, 2013.

29 D. Okanohara and K. Sadakane. Practical entropy-compressed rank/select dictionary. In *Proceedings of the 2007 Workshop on Algorithm Engineering and Experiments (ALENEX 2007)*. SIAM, 2007. doi:10.1137/1.9781611972870.6.

30 S. J. Puglisi and A. Turpin. Space-time tradeoffs for longest-common-prefix array computation. In *Proceedings of the 19th International Symposium on Algorithms and Computation (ISAAC 2008)*, volume 5369 of *LNCS*, pages 124–135. Springer, 2008. doi:10.1007/978-3-540-92182-0_14.

31 K. Sadakane. Succinct representations of lcp information and improvements in the compressed suffix arrays. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2002)*, pages 225–232. ACM/SIAM, 2002.

32 J. Shun. Fast parallel computation of longest common prefixes. In *Proceedings of the 2014 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2014)*, pages 387–398. IEEE, 2014. doi:10.1109/SC.2014.37.

33 J. Sirén. Sampled longest common prefix array. In *Proceedings of the 21st Annual Symposium on Combinatorial Pattern Matching (CPM 2010)*, volume 6129 of *LNCS*, pages 227–237. Springer, 2010. doi:10.1007/978-3-642-13509-5_21.

34 J. S. Vitter. Algorithms and data structures for external memory. *Found. Trends Theoretical Computer Science*, 2(4):305–474, 2006. doi:10.1561/0400000014.

Almost All Even Yao-Yao Graphs Are Spanners*

Jian Li[1] **and Wei Zhan**[2]

1 **Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China**
lijian83@mail.tsinghua.edu.cn

2 **Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China**
zhanw-13@mails.tsinghua.edu.cn

Abstract

It is an open problem whether Yao-Yao graphs YY_k (also known as sparse-Yao graphs) are all spanners when the integer parameter k is large enough. In this paper we show that, for any integer $k \geq 42$, the Yao-Yao graph YY_{2k} is a t_k-spanner, with stretch factor $t_k = 6.03 + O(k^{-1})$ when k tends to infinity. Our result generalizes the best known result which asserts that all YY_{6k} are spanners for $k \geq 6$ [Bauer and Damian, SODA'13]. Our proof is also somewhat simpler.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, I.3.5 Computational Geometry and Object Modeling

Keywords and phrases Yao-Yao graph, geometric spanner, curved trapezoid

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.62

1 Introduction

Let $\mathcal{P}$ be a set of points on the Euclidean plane $\mathbb{R}^2$. A simple undirected graph $G = (V, E)$ with vertex set $V = \mathcal{P}$ is called a *geometric t-spanner*, if for any pair of vertices (u, v) there is a path $uu_1 \dots u_s v$ in G such that the total length of the path $|uu_1| + |u_1u_2| + \dots + |u_s v|$ is at most t times the Euclidean distance $|uv|$, where t is a constant, called the *stretch factor*. The concept of geometric spanners was first proposed in [7], and can be considered as a special case of general graph spanners [21] if we consider the complete graph with edge lengths defined by the Euclidean metric.

Yao graph was first introduced by Andrew Yao in his seminal work on high-dimensional Euclidean minimum spanning trees [23]. Let $C_u(\gamma_1, \gamma_2)$ be the cone with apex u and consisting of the rays with polar angles in $[\gamma_1, \gamma_2)$. Given a positive integer parameter k, the construction of Yao graph Y_k is described in the following process.

```
Initially $\mathsf{Y}_k$ is an empty graph.
For each point $u$:
    For each $j = 0, \dots, k-1$:
        Let $C = C_u(2j\pi/k, 2(j+1)\pi/k)$;
        Select $v \in C \cap \mathcal{P}$ such that $|uv|$ is the shortest;
        Add edge $\overrightarrow{uv}$ into $\mathsf{Y}_k$.
```

* JL and WZ are supported in part by the National Basic Research Program of China grants 2015CB358700, 2011CBA00300, 2011CBA00301, and the National NSFC grants 61033001, 61361136003.

In the above process, ties are broken in an arbitrary but consistent manner. The above process is usually referred to as a “Yao step”. Note that the edges $\overrightarrow{uv}$ we added are directed edges (the directions are useful in the construction of Yao-Yao graphs).

Besides the usefulness in constructing minimum spanning trees, Yao graphs are also sparse graphs with surprisingly nice spanning properties. It is known that Yao graph Y_k has a stretch factor $(1-2\sin(\pi/k))^{-1}$ when $k \geq 6$ (see e.g., [18] and [5]). Later, the spanning properties of Y_4, Y_6 and Y_5 are proved in a series of work [10, 11, 2]. Due to the spanning properties and the simplicity of the construction, they have important applications in networking and wireless communication; we refer to [20] and [16] for more details.

One may notice that a Yao graph may not have a bounded degree. This is a serious drawback in certain wireless networking applications, since a wireless node can only communicate with a bounded number of neighbors. The issue was initially realized by Li et al. [18]. To address the issue, they proposed a modified construction with two Yao steps: the first Yao step produces a Yao graph Y_k, and the second step, called the “reverse Yao step”, eliminates a subset of edges of Y_k to ensure the maximum degree is bounded. The reverse Yao step can be described by the following procedure:

```
Initially $\mathsf{YY}_k$ is an empty graph.
For each point $u$:
    For each $j = 0, \dots, k-1$:
        $C = C_u(2j\pi/k, 2(j+1)\pi/k)$;
        Select $v \in C \cap \mathcal{P}$, $\overrightarrow{vu} \in \mathsf{Y}_k$ such that $|uv|$ is the shortest;
        Add edge $\overrightarrow{vu}$ into $\mathsf{YY}_k$.
```

The resulting graph, YY_k, is named as “Yao-Yao graph” or “Sparse-Yao graph” in the literature. The node degrees in YY_k are clearly upper-bounded by $2k$. It has long been conjectured that YY_k are also geometric spanners when k is larger than some constant threshold [18, 16, 3]:

► **Conjecture 1.** *There exists a constant k_0 such that for any integer $k > k_0$, YY_k is a geometric spanner.*

In sharp contrast to Yao graphs, our knowledge about the spanning properties of Yao-Yao graphs is still quite limited. Li et al. [18] proved that YY_k is connected for $k > 6$, and provided extensive experimental evidence suggesting YY_k are indeed spanners for larger k. Jia et al. [14] and Damian [9] showed that YY_k is spanner in certain special cases (the underlying point set satisfies certain restrictions). It is also known that for small constant k's, none of YY_k with $2 \leq k \leq 6$ admits a constant stretch factor [2, 10, 13]. Recently, a substantial progress was made by Bauer and Damian [3], who showed that YY_{6k} are spanners with stretch factor 11.76 for all integer $k \geq 6$ and the factor drops to 4.75 when $k \geq 8$. In fact, a closer examination of the proofs in their paper actually implies an asymptotic stretch factor of $2 + O(k^{-1})$. None of YY_k with other k values have been proved or disproved to be spanners. We note here that some of the aforementioned work [18, 9] focused on *UDGs (unit disk graphs).* But their arguments can be easily translated to general planar point sets as well. [1]

[1] Suppose a spanning property holds for any UDG. For a general set of points, we first scale it so that its diameter is less than 1. We can see that the UDG defined over the scaled set is a complete graph. So the spanning property also holds for the set of points.

In this paper, we improve the current knowledge of Yao-Yao graphs and make a step towards the resolution of Conjecture 1, by showing that almost all Yao-Yao graphs with even k are geometric spanners for k large enough. Formally, our results is as follows:

▶ **Theorem 2.** *For any $k \geq 42$, YY_{2k} is a t_k-spanner, where $t_k = 6.03 + O(k^{-1})$.*

Our Technique Our proof contains two major steps.

1. (Section 3) We first introduce two classes of intermediate graphs, called *overlapping Yao graphs* (denoted as OY_k) and *trapezoidal Yao graphs* (denoted as TY_k). The construction of overlapping Yao graphs is similar to that of Yao graphs, except that the cones incident on a vertex overlap with each other. We can easily show that OY_k is a geometric spanner. The definition of trapezoidal Yao graphs is also similar to Yao graphs, but takes advantage of a shape called *curved trapezoid* (defined in Section 2). We can show that OY_k is a subgraph of TY_k, implying TY_k is a geometric spanner as well.
2. (Section 4) In the second step, we show that YY_{2k} spans TY_{2k} (i.e., for any $u, v \in \mathcal{P}$, the shortest u-v path in YY_{2k} is at most a constant times longer than the shortest u-v path in TY_{2k}).

Our proof makes crucial use of the properties of curved trapezoids. Roughly speaking, curved trapezoids are more flexible than triangles which were used in [3]), which is the main reason for the improvement from YY_{6k} to more general YY_{2k}.

Related Works Replacing the Euclidean distance $|\cdot|$ by a power $|\cdot|^\kappa$ with a constant $\kappa \geq 2$ leads to the definition of *power spanners*. Since the power of length models the energy consumed in wireless transmissions, power spanners have important implications in wireless networking applications. In this setting, Yao-Yao graphs YY_k have been proved to be power spanners for any $\kappa \geq 2$ when $k > 6$ [14, 22]. It is clear that when a graph is a geometric spanner, it must also be a power spanner (See [17, Lemma 1]), however the reverse does not hold.

We also note that neither Yao graphs nor Yao-Yao graphs can be guaranteed to be planar graphs [22, 15], whereas Delaunay triangulation provides another type of spanner which is planar but without bounded degree [12, 4]. In order to facilitate network design in certain applications, some previous work [6, 19] made use of both Yao graphs and Delaunay graphs to produce degree-bounded and planar geometric spanners.

The paper is organized as follows. We introduce some standard notations and useful tools, including the shape of curved trapezoid, in Section 2. We introduce overlapping Yao graphs, trapezoidal Yao graphs, and prove their spanning properties in Section 3. We prove our main result in Section 4. Finally, we conclude with some future work in Section 5.

2 Preliminaries

$\mathcal{P}$ is the underlying set of points in $\mathbb{R}^2$. $D(a, \rho)$ denotes an open disk centered at point a with radius ρ. The boundary and closure of a region R are denoted by ∂R and $\overline{R}$, respectively. Let $S(a, \rho) = \partial D(a, \rho)$ be the circle centered at a with radius ρ. The length of shortest u-v path in a graph G is denoted by $d_G(u, v)$.

On $\mathbb{R}^2$, a point u with coordinates (x, y) is denoted by $u(x, y)$. Let $o(0, 0)$ be the origin of $\mathbb{R}^2$. The positive direction of x-axis is fixed as the polar axis throughout the construction and analysis. For a point $a \in \mathbb{R}^2$, we use x_a to denote its x-coordinate and y_a its y-coordinate. We use $\varphi(uv)$ to represent the polar angle of vector $\overrightarrow{uv}$. The angle computations are all under the modulo of 2π, and angle subtraction is regarded as the

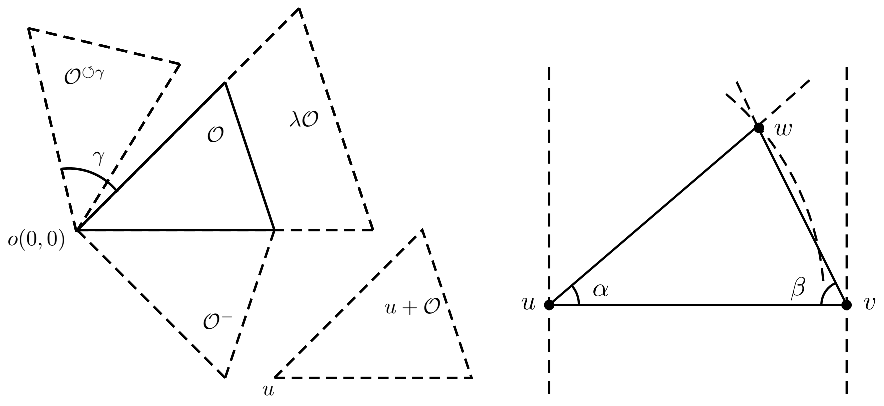


Figure 1 Left: Affine transformations. Right: Proof for Lemma 3.

counterclockwise difference. A cone between polar angles $\gamma_1 < \gamma_2$ with apex at the origin is denoted as $C(\gamma_1, \gamma_2) = \{u \mid \varphi(ou) \in [\gamma_1, \gamma_2)\}$.

It is also necessary to introduce some notations of standard affine transformations on a geometric object $\mathcal{O}$ in the plane:

- (Dilation) If $\mathcal{O}$ is uniformly scaled by factor λ with the origin as the center, the result is denoted by $\lambda\mathcal{O} = \{\lambda z \mid z \in \mathcal{O}\}$.
- (Translation) If $\mathcal{O}$ is translated so that the original point goes to point u, the result is denoted by $u + \mathcal{O} = \{u + z \mid z \in \mathcal{O}\}$.
- (Rotation) If $\mathcal{O}$ rotated an angle γ counterclockwise with respect to the origin, the result is denoted by $\mathcal{O}^{\circlearrowleft\gamma}$.
- (Reflection) If $\mathcal{O}$ is reflected through the x-axis, the result is denoted by $\mathcal{O}^-$.

By the above notations, we can denote the cone with apex u by $u + C(\gamma_1, \gamma_2)$ (abbreviated as $C_u(\gamma_1, \gamma_2)$).

Geometric Inequalities

In order to attain a constant stretch factor for Yao graphs and their variants, one often needs to bound a certain geometric ratio. Here we present a simple yet general lemma for this purpose. Let the *open strip* on the segment uv be the collection of points

▶ **Lemma 3.** *Suppose $\tau \geq 1$ is a constant. Let w be a point with $\angle wuv, \angle wvu \in [0, \pi/2)$ (see Figure 1). When $\tau|vw| < |uv|$, the following statements about the ratio $|uw|/(|uv| - \tau|vw|)$ hold:*

1. *If w is restricted within a compact segment of an arc centered at u, the ratio is maximized when $|vw|$ is the largest;*
2. *If w is restricted within a compact segment of a ray from v, the ratio is maximized when $|vw|$ is the largest;*
3. *If w is restricted within a compact segment of a ray originated from u, the maximum ratio is achieved when w makes $|uw|$ the largest or smallest;*

Proof. The first statement is straightforward since both $|uv|$ and $|uw|$ are fixed. Now, we show the last two properties. Let $\alpha = \angle wuv$ and $\beta = \angle wvu$. By the law of sines and the fact that $\sin(\alpha + \beta) = \sin\alpha\cos\beta + \cos\alpha\sin\beta$, we can see that

$$\frac{|uw|}{|uv| - \tau|vw|} = \frac{\sin\beta}{\sin(\alpha+\beta) - \tau\sin(\alpha)} = \frac{1}{\cos\alpha - \sin\alpha \cdot (\tau - \cos\beta)/\sin\beta}. \tag{1}$$

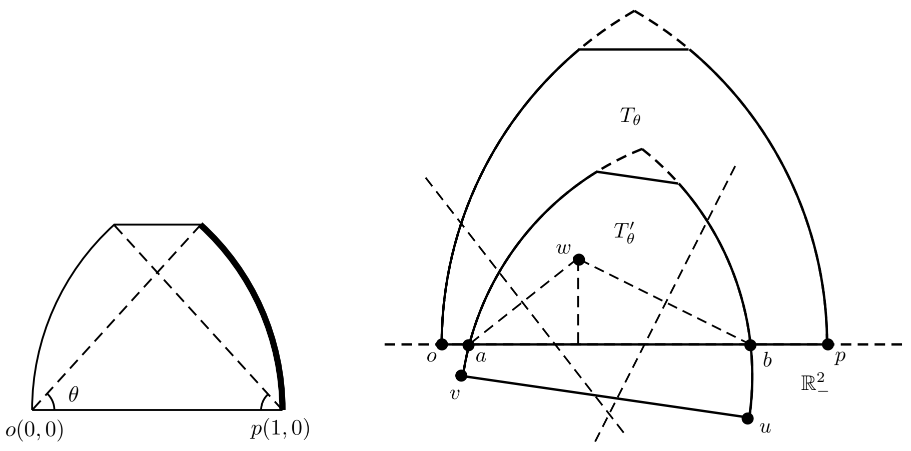


Figure 2 Left: The curved trapezoid. The critical arc is shown in bold. Right: An illustration of Lemma 6 with its proof.

One can see the ratio is maximized when α is maximized, which implies the second statement. Moreover, $(\tau - \cos\beta)/\sin\beta$ is a convex function of β for $\beta \in (0, \pi/2)$. Hence, it is maximized when β is maximized or minimized, which implies the last statement. ◀

Due to Lemma 3, to compute an upper bound for $|uw|/(|uv| - |vw|)$ in a region, it suffices to consider a small number of extreme positions. In particular, the following corollary is an immediately consequence, which is an improvement of a classical result mentioned in [8]:

▶ **Corollary 4.** *$R = \overline{D(u, |uv|) \cap C_u(\gamma_1, \gamma_2)}$ is a sector with apex u and with v on its arc. Suppose $\alpha = \max\{\varphi(uv) - \gamma_1, \gamma_2 - \varphi(uv)\} < \pi/3$. Then, for all $w \in R \setminus \{u\}$, it holds that*

$$\frac{|uw|}{|uv| - |vw|} \leq \left(1 - 2\sin\frac{\alpha}{2}\right)^{-1}.$$

Proof. Take $\tau = 1$ in Lemma 3. By considering the rays from u and the arcs centered at u, and applying the first and third cases in Lemma 3, we can see that the ratio can only be maximized when w is at the two corners of the sector with $\beta = (\pi - \alpha)/2$, or when w approaches u with $\beta = 0$. The two upper bounds in equality 1 in these cases are $(1 - 2\sin(\alpha/2))^{-1}$ and $(\cos\alpha)^{-1}$, and the former one is larger. ◀

Curved Trapezoid T_θ

To construct the trapezoidal Yao Graphs TY_k, we need to define an open shape, called *curved trapezoid*, as follows.

▶ **Definition 5** (Curved Trapezoid T_θ)**.** Fix two points $o(0, 0)$ and $p(1, 0)$. Given $\theta \in [\pi/4, \pi/3)$, we define the curved trapezoid T_θ as:

$$T_\theta = \{u(x, y) \mid 0 < x < 1, 0 < y < \sin\theta, |ou| < 1, |pu| < 1\}.$$

Intuitively, it is the convex hull of two sectors with apices at o and p respectively. We regard T_θ as a shape attached to the origin o, and the closed arc not incident on o is called the *critical arc*. See Figure 2 for an example.

We list some useful properties of T_θ here. It is straightforward to show that T_θ is symmetric with respect to the vertical line $x = \frac{1}{2}$. For any $u \in T_\theta$ it holds that $0 < \varphi(ou) < \pi/2$. If u is on the critical arc, it holds that $0 < \varphi(ou) < \theta$. The following crucial lemma, illustrated in Figure 2, is also an immediate consequence from the definition.

▶ **Lemma 6.** *Denote $\mathbb{R}^2_-$ as the lower half-plane $\{(x, y) \mid y \le 0\}$. Consider any two points $u, v \in \mathbb{R}^2_-$ satisfying $0 < x_u < 1$, $|\varphi(uv) - \pi| < \frac{1}{6}\pi$, and $|ou|, |pv| \in [|uv|, 1)$. We construct a similar shape $T'_\theta := u + |uv|(T^-_\theta)^{\circlearrowleft\varphi(uv)}$. We have that $T'_\theta - T_\theta$ is completely contained in lower half plane $\mathbb{R}^2_-$.*

Proof. We first consider the extreme case when $\theta = \pi/3$. Since $u, v \in \mathbb{R}^2_-$, if the two arcs of T'_θ do not intersect with the line $y = 0$, then T_θ is fully contained in $\mathbb{R}^2_-$ and the proof is done.

Now suppose the two intersections are a, b, illustrated in the RHS of Figure 2. Notice that $|ua| = |uv| \le |ou|$ and $|vb| = |uv| \le |pv|$. Combining with the condition that $0 < x_u < 1$, we can see that both a and b lie in the segment op. For any point $w \in T'_\theta - \mathbb{R}^2_-$, it suffices to prove $|ow|, |pw| < 1$ so that w is in T_θ. This is done by examining the perpendicular bisector of aw and bw: since $|ua| > |uw|$, u is in the upper half-plane of the bisector of aw, and so is p. Thus $|pw| < |pa| \le |op| = 1$. The same arguments hold for $|ow| < 1$.

The above shows that $T'_\theta - T_\theta \subset \mathbb{R}^2_-$ when $\theta = \pi/3$, and if θ is smaller, we only need to show that the distance $d(w, op)$ from w to line $y = 0$ is less than $\sin\theta$. This is done by the observation that $d(w, ab) \le d(w, uv)$ in T'_θ, and $d(w, uv) < |uv| \sin\theta < \sin\theta$. This completes the proof for Lemma 6. ◀

3 Overlapping Yao Graphs and Trapezoidal Yao Graphs

In this section we consider two variants of Yao graphs, overlapping Yao graphs and trapezoidal Yao graphs.

▶ **Definition 7** (Overlapping Yao Graph OY_k). Let $\gamma = \left\lceil \frac{k}{4} \right\rceil \frac{2\pi}{k}$. For every $u \in \mathcal{P}$ and $j = 0, \ldots, k-1$, select shortest $\overrightarrow{uv}$ with $v \in C_u(2j\pi/k, 2j\pi/k + \gamma)$. The chosen edges form the *overlapping Yao graph* $\mathsf{OY}_k(\mathcal{P})$.

The angle γ in the definition is actually the smallest multiple of $2\pi/k$ which is no less than $\pi/2$. We note here that the term “overlapping" comes from the fact that the cones with the same apex overlap with each other, while in the original Yao graphs they are disjoint. First we claim that OY_k is a spanner when k is large.

▶ **Lemma 8.** *If $k > 24$, then OY_k is a τ_k-spanner where $\tau_k = \left(1 - 2\sin\left(\frac{\pi}{k} + \frac{\pi}{8}\right)\right)^{-1}$.*

Proof. We prove $d_{\mathsf{OY}}(u, v) \le \tau_k |uv|$ by induction on the length $|uv|$ for $u, v \in \mathcal{P}$. Notice that it is always possible to find j so that $v \in C_u\left(2j\pi/k + \pi/4, 2(j+1)\pi/k + \pi/4\right)$. See Figure 3 for an illustration. This cone is contained in $C_u(\gamma_1, \gamma_2)$, where $\gamma_1 = 2j\pi/k$ and $\gamma_2 = 2j\pi/k + \gamma$, since $k > 24$ and thus $\gamma \ge \pi/2$. If $\overrightarrow{uv}$ is the shortest in $C_u(\gamma_1, \gamma_2)$, then $\overrightarrow{uv} \in \mathsf{OY}_k$ and we are done with the proof.

Now, suppose that in the construction of construction of OY_k, we choose $\overrightarrow{uw}$ for $C_u(\gamma_1, \gamma_2)$ with $w \ne v$. Since $k > 24$, one shall see that

$$\begin{aligned} \alpha \quad &:= \quad \max\{\varphi(uv) - \gamma_1, \gamma_2 - \varphi(uv)\} \\ &\le \quad \max\{2\pi/k + \pi/4, \gamma - \pi/4\} = 2\pi/k + \pi/4 < \pi/3. \end{aligned}$$

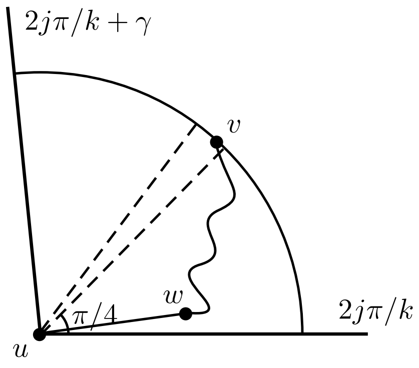


Figure 3 Illustration of the proof of Lemma 8. The outer cone is $C_u(\gamma_1,\gamma_2)$ where $\gamma_1 = 2j\pi/k$ and $\gamma_2 = 2j\pi/k+\gamma$ The cone defined by two dashed lines is $C_u\,(2j\pi/k+\pi/4, 2(j+1)\pi/k+\pi/4)$.

Hence, we can apply Corollary 4 on the sector $R = \overline{D(u,|uv|)\cap C_u(\gamma_1,\gamma_2)}$, which claims

$$\frac{|uw|}{|uv|-|vw|} \le \tau_k = \left(1-2\sin\frac{\alpha}{2}\right)^{-1}.$$

Since $\alpha < \pi/3$, we have $|vw| < |uv|$. By the induction hypothesis, we can see that

$$d_{\mathsf{OY}}(u,v) \le |uw| + d_{\mathsf{OY}}(v,w) \le |uw| + \tau_k|vw| \le \tau_k(|uv|-|vw|) + \tau_k|vw| = \tau_k|uv|,$$

which completes the proof. ◀

We also note that Barba et al. proposed the so-called *continuous Yao graphs* in [1], which play a similar role as OY_k. By adapting their method, it might be possible to prove a slightly smaller stretch factor for OY_k. However the constant is not our primary goal in this paper, and we leave it to the future work.

Now we define the trapezoidal Yao graphs based on the curved trapezoid T_θ (Definition 5).

▶ **Definition 9** (Trapezoidal Yao Graph TY_k). Let $\theta = \left\lceil \frac{k}{8} \right\rceil \frac{2\pi}{k}$. For every $u \in \mathcal{P}$ and $j = 0,\ldots,k-1$, define two curved trapezoids

$$\Gamma_{j1} = (T_\theta)^{\circlearrowleft 2j\pi/k} \qquad \text{and} \qquad \Gamma_{j2} = (T_\theta^-)^{\circlearrowleft 2j\pi/k}.$$

Note that their bottom sides lie on the ray of angle $2j\pi/k$. For each $i = 1,2$, we do the following: We grow a curved trapezoid $u + \lambda\Gamma_{ji}$ by gradually increasing λ (initially 0) until its boundary hits some point v. If the critical arc hits v, we select $\overrightarrow{uv}$. Otherwise, we select nothing. All the selected edges form the graph $\mathsf{TY}_k(\mathcal{P})$.

▶ **Lemma 10.** *For any integer $k > 24$, TY_k is a τ_k-spanner where* $\tau_k = \left(1-2\sin\left(\frac{\pi}{k}+\frac{\pi}{8}\right)\right)^{-1}$.

Proof. Note that in Definition 5, we require that $\theta \in [\pi/4, \pi/3)$. When $k > 24$, we can see that

$$\frac{\pi}{4} \le \theta = \left\lceil \frac{k}{8} \right\rceil \frac{2\pi}{k} \le \frac{k+7}{k}\cdot\frac{\pi}{4} < \frac{\pi}{3}.$$

Hence, the value θ in Definition 9 satisfies the requirement of θ in Definition 5.

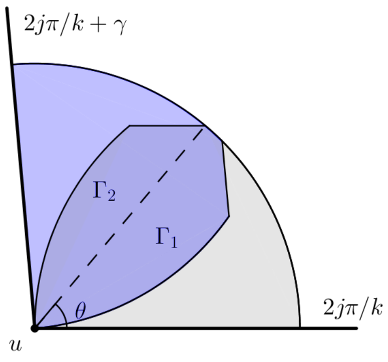


Figure 4 Illustration for Lemma 10: Two curved trapezoids cover a sector.

We show that the overlapping Yao graph OY_k is actually a subgraph of TY_k, from which the lemma is an immediate consequence. Given any j and one cone $C_u(2j\pi/k, 2j\pi/k+\gamma)$, we choose two copies of T_θ: $\Gamma_1 = (T_\theta)^{\circlearrowleft 2j\pi/k}$, and $\Gamma_2 = (T_\theta^-)^{\circlearrowleft 2j\pi/k+\gamma}$. See Figure 4. Since $2\theta \geq \gamma \geq \pi/2$, it is clear that Γ_1 and Γ_2 exactly cover the interior of sector $D(o,1) \cap C_o(2j\pi/k, 2j\pi/k+\gamma)$. Now, for any edge $\overrightarrow{uv} \in \mathsf{OY}_k$ selected within this cone, both $u+|uv|\Gamma_1$ and $u+|uv|\Gamma_2$ must have empty interior ($|uv|$ is the shortest within the cone). Therefore, if $\varphi(uv) - 2j\pi/k \leq \theta$, it should be selected into TY_k with respect to Γ_1. Otherwise it should be selected with respect to Γ_2. This implies that OY_k is a subgraph of TY_k, and by Lemma 8, TY_k is a τ_k-spanner as well. ◀

4 Yao-Yao Graphs YY_{2k} are Spanners

In this section, we prove our main result that YY_{2k} has a constant stretch factor for large k, by show that YY_{2k} spans TY_{2k}.

To begin with, we prove an important property of TY_{2k}, which will be useful later. The property can be seen as an analogue of the property of Θ_6 shown in [3, Lemma 2]. One of the reasons we can improve YY_{6k} to more general YY_{2k} is that we can take advantage of the curved trapezoid rather than the regular triangle. We state this property by assuming, that without loss of generality, a curved trapezoid T_θ is placed at its normal position, i.e., T_θ lies in the first quadrant with two vertices $o(0,0)$ and $p(1,0)$. From now on, we work with trapezoidal Yao graphs TY_{2k}. Hence, the associated parameter θ (see Definition 9) is $\theta = \left\lceil \frac{2k}{8} \right\rceil \frac{2\pi}{2k} = \left\lceil \frac{k}{4} \right\rceil \frac{\pi}{k}$.

▶ **Lemma 11.** *Suppose $o \in \mathcal{P}$ and T_θ has an empty interior. If there is a point $a \in \mathcal{P}$ such that $0 < x_a < 1$, $y_a \leq 0$ and $0 < \varphi(ap) < \pi/6$, then*

$$d_{\mathsf{TY}}(oa) \leq x_a + (2\tau_{2k}+1)|y_a|$$

where τ_k is as defined in Lemma 10, and there is a path in TY_{2k} from a to o where each edge is shorter than $|oa|$.

Proof. Note that simply applying Lemma 10 is insufficient to achieve the guarantee. Instead, we present an iterative algorithm for finding a path from a to o. The path found by the iterative algorithm is not necessarily a shortest path from a to o. Nevertheless, we can bound its length as in the lemma.

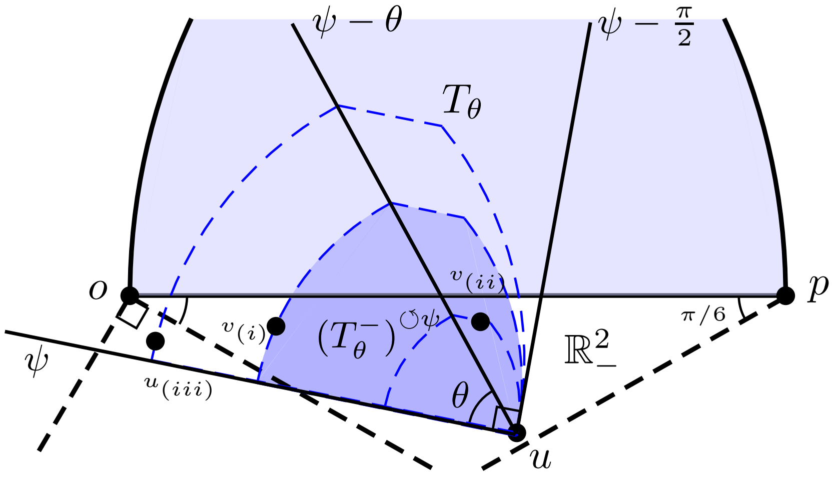


Figure 5 Illustration of three cases in the proof of Lemma 11.

```
Let $u$ be the current point in $\mathcal{P}$; Initially $u$ is set to be $a$.
While $u \neq o$ and $\varphi(ou) > -\frac{\pi}{6}$ do
    Let $\psi = \min\{j\pi/k \mid j\pi/k > \varphi(uo), j = 0, 1, \ldots, 2k-1\}$;
    Grow a curved trapezoid $u + \lambda(T_\theta^-)^{\circlearrowleft\psi}$ until its boundary hits some point $v$.
    If $v \in C_u(\psi - \frac{\pi}{2}, \psi - \theta)$:
        (i) $\mathsf{TY}_{2k}$ must contain the edge $\overrightarrow{uv}$. We add $\overrightarrow{uv}$ to our path;
    Otherwise:
        (ii) Add the path from $u$ to $v$ in $\mathsf{OY}_{2k}$ constructed in Lemma 8;
    Set the current point $u$ to be $v$ and proceed to the next iteration.
If now $u = o$, the path is already found;
(iii) Otherwise take the path from $u$ to $o$ in $\mathsf{OY}_{2k}$ constructed in Lemma 8.
```

See Figure 5 for an illustration of the three cases (i),(ii) and (iii). Let $l(u)$ be the length of the path from u to o generated by the algorithm above. To analyze its behaviour, we define a potential function

$$\Phi(u) = x_u + (2\tau_{2k} + 1)|y_u| - l(u).$$

We claim that the potential function never increases as the algorithm proceeds and is eventually 0. The potential only changes in the three labeled steps (i),(ii) and (iii). When executing step (i) and (ii), it is clear that $\varphi(uo) \in \left(\frac{5}{6}\pi, \pi\right]$, $\varphi(up) \in \left[0, \frac{1}{6}\pi\right)$. Therefore ψ also falls in the range $\left(\frac{5}{6}\pi, \pi\right]$. Moreover, v must be contained in $\mathbb{R}^2_-$ due to Lemma 6, that is, $y_v \leq 0$.

- In step (i), v is simply the nearest neighbor in the cone $C_u(\psi - \theta, \psi)$. Since $\varphi(up) < \pi/6$, we know $|uv| < |vp|$. Since $\psi - \theta > \pi/2$ and $\psi \leq \pi$, we can see that $x_v \leq x_u$ and $|y_v| < |y_u|$. Hence the change in potential is

$$\begin{aligned}\Delta\Phi = \Phi(v) - \Phi(u) &= |uv| - (x_u - x_v) - (2\tau_{2k} + 1)(|y_u| - |y_v|)\\ &\leq |uv| - (x_u - x_v) - (|y_u| - |y_v|) \leq 0.\end{aligned}$$

- In step (ii), since $\varphi(uv) \in [\psi - \frac{\pi}{2}, \psi - \theta)$, we know $|\varphi(uv) - \pi/2|$ is at most $\pi/4$, so $|y_v| < |y_u|$ and $|x_v - x_u| < |y_u| - |y_v|$. That further implies $|uv| < 2(|y_u| - |y_v|)$, and thus $d_{\mathsf{TY}}(uv) \le \tau_{2k}|uv| < 2\tau_{2k}(|y_u| - |y_v|)$. Therefore the change in potential is

$$\begin{aligned}\Delta\Phi = \Phi(v) - \Phi(u) &= d_{\mathsf{TY}}(uv) - (x_u - x_v) - (2\tau_{2k} + 1)(|y_u| - |y_v|) \\ &= (d_{\mathsf{TY}}(uv) - 2\tau_{2k}(|y_u| - |y_v|)) + (x_v - x_u - (|y_u| - |y_v|)) \le 0.\end{aligned}$$

- In step (iii), suppose the node arrived before u is u', with $\varphi(u'o) > \frac{5}{6}\pi$. Since $\overrightarrow{u'u}$ is chosen, it must be the case that $|u'u| \le |u'o|$. It is immediate that $\varphi(uo) \in \left(\frac{\pi}{3}, \frac{5}{6}\pi\right]$, and thus $|uo| \le 2|y_u|$, $-x_u \le |y_u|$. Therefore, we have that

$$\Delta\Phi = \Phi(o) - \Phi(u) = (d_{\mathsf{TY}}(uo) - 2\tau_{2k}|y_u|) + (-x_u - |y_u|) \le 0.$$

Now that $\Phi(o) = 0$, and the above arguments show that the potential cannot increase during the path from u to o, we can conclude that $\Phi(a) \ge 0$. That is, $l(a) \le x_a + (2\tau_{2k} + 1)|y_a|$.

Also notice that in steps (i) and (ii), it is ensured that $|uv| < |ou|$ and $|ov| < |ou|$ since v is contained in the curved trapezoid. On the other hand, the paths from OY_{2k} in steps (ii) and (iii), which are constructed according to Lemma 8, contains only edges shorter than the direct distance. Thus the edges we selected are all shorter than $|oa|$. ◀

▶ **Lemma 12.** *On a Yao-Yao graph $\mathsf{YY}_{2k}(\mathcal{P})$ with $k \ge 42$, if $\overrightarrow{uv} \in \mathsf{TY}_{2k}$, then $d_{\mathsf{YY}}(uv) \le \tau'_k|uv|$, where $\tau'_k = \sqrt{2} + O(k^{-1})$ is a constant depending only on k.*

Proof. Let $\tau'_k \ge 1$ be a constant to be fixed later. Our proof is by induction on the length $|uv|$. The base case is simple: the shortest edge is in both TY_{2k} and YY_{2k}. Now, consider an edge $\overrightarrow{uv} \in \mathsf{TY}_{2k}$. Without loss of generality, we assume that $|uv| = 1$. We further assume that T_θ, which generates $\overrightarrow{uv}$ in TY_{2k}, in its normalized position, i.e., u is $o(0,0)$ and another vertex is $p(1,0)$. See Figure 6. Noticing that θ is a multiple of π/k, we can see that v must be the nearest neighbor of u in its corresponding cone of YY_{2k}.

If $\overrightarrow{uv} \in \mathsf{YY}_{2k}$, then $d_{\mathsf{YY}}(uv) = |uv|$ and we are done with the proof. Otherwise, there must be another edge $\overrightarrow{wv} \in \mathsf{YY}_{2k}$ where u and w are in the same v-apex cone, and $|vw| \le |uv|$ (this is due to the construction of Yao-Yao graph). It follows that $|uw| < |uv|$ since $k \ge 42$.

Now, we prove by induction that $d_{\mathsf{YY}}(uv) \le \tau'_k|uv|$. Now noticing that T_θ has an empty interior, there are only two cases for w we need to consider. See Figure 6 for an example.

1. $y_w > 0$: See the left part of Figure 6. By Lemma 3, we can see that within all possible positions of w, $|vw|/(|uv| - \tau_{2k}|uw|)$ can only be maximized when $|vw| = |uv|$ or when w is on the left boundary of T_θ. Since when $k \ge 42$, it holds $2\tau_{2k}\sin\frac{\pi}{2k} < 1$, therefore

$$\frac{|vw|}{|uv| - \tau_{2k}|uw|} \le \frac{1}{1 - 2\tau_{2k}\sin\frac{\pi}{2k}}.$$

   Hence, if $\tau'_k \ge 1/(1 - 2\tau_{2k}\sin\frac{\pi}{2k})$, we have that $|vw| \le \tau'_k(|uv| - \tau_{2k}|uw|)$. Consider the shortest path from u to w in TY_{2k}. Notice that $d_{\mathsf{TY}}(uw) \le \tau_{2k}|uw| < |uv|$. Applying the induction hypothesis to every edge of the shortest path, we can see $d_{\mathsf{YY}}(uw) \le \tau'_k d_{\mathsf{TY}}(uw)$. Combining the above inequalities, we can finally obtain that

$$d_{\mathsf{YY}}(uv) \le d_{\mathsf{YY}}(uw) + |vw| \le \tau'_k(d_{\mathsf{TY}}(uw) + |uv| - \tau_{2k}|uw|) \le \tau'_k|uv|.$$

2. $y_w \le 0$: See the middle and right parts of Figure 6. Consider the two cases of angle $\varphi(uv)$:

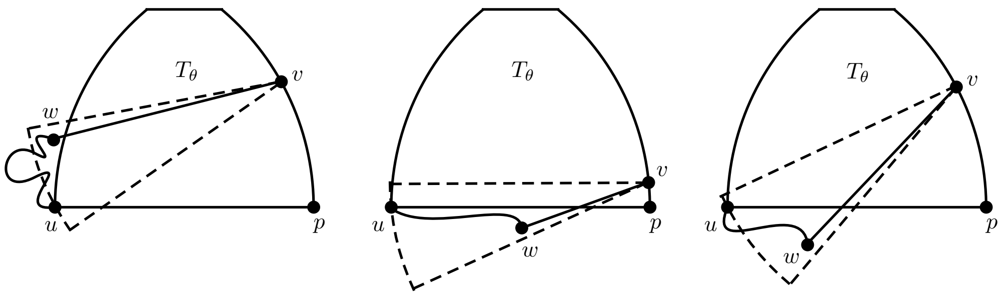


Figure 6 Proof for Lemma 12. Left: w is in the upper half-plane; Middle: w is in the lower half-plane and $\varphi(uv) \le \pi/k$; Right: w is in the lower half-plane and $\varphi(uv) > \pi/k$.

- If $0 \le \varphi(uv) \le \pi/k$, then $\varphi(wp) \le \varphi(wv) \le \pi/k$;
- If $\pi/k < \varphi(uv) \le \theta$, then $\angle upv \le \frac{(k-1)\pi}{2k} \le \angle uwv$. Thus p is outside the circumcircle of $\triangle uvw$, and $\varphi(wp) = \angle upw \le \angle uvw \le \pi/k$.

Therefore in both cases, we have $0 < x_w < x_p$ and $0 \le \varphi(wp) \le \pi/k$. We notice the following two inequalities:

$$|y_w| \le (x_p - x_w) \tan \varphi(wp) \le (|uv| - x_w) \tan \frac{\pi}{k},$$
$$|vw| \le (x_v - x_w) \sec \varphi(wv) \le (|uv| - x_w) \sec \left(\theta + \frac{\pi}{k}\right).$$

Consider the path from w to u as we constructed in Lemma 11 (w and u correspond to a and o in the lemma, respectively). Since all edges in the constructed path are shorter than $|wu|$ (thus shorter than $|uv|$ as well), we can apply the induction hypothesis to get $d_{\mathsf{YY}}(uw) \le \tau'_k d_{\mathsf{TY}}(uw)$. Applying Lemma 11, we have

$$\begin{aligned} d_{\mathsf{YY}}(uv) &\le d_{\mathsf{YY}}(uw) + |vw| \le \tau'_k d_{\mathsf{TY}}(uw) + |vw| \\ &\le \tau'_k (x_w + (2\tau_{2k} + 1)|y_w|) + |vw| \\ &\le \tau'_k x_w + \left(\tau'_k (2\tau_{2k} + 1) \tan \frac{\pi}{k} + \sec\left(\theta + \frac{\pi}{k}\right)\right)(|uv| - x_w). \end{aligned}$$

Suppose we choose τ'_k such that $\tau'_k(2\tau_{2k}+1)\tan\frac{\pi}{k} + \sec(\theta+\frac{\pi}{k}) \le \tau'_k$, (which is equivalent to $\tau' \ge ((1-(2\tau_{2k}+1)\tan\frac{\pi}{k})\cos(\theta+\frac{\pi}{k}))^{-1}$). Note that the above holds also because $(2\tau_{2k}+1)\tan\frac{\pi}{k} < 1$ when $k \ge 42$. Finally, we can conclude that

$$d_{\mathsf{YY}}(uv) \le \tau'_k |uv|.$$

We can conclude from the above two cases that τ'_k can be chosen to be $\sec\frac{\pi}{4} + O(k^{-1}) = \sqrt{2} + O(k^{-1})$. ◀

Combining Lemma 10 and 12, with the observation that $\tau_k = (1 - 2\sin(\pi/8))^{-1} + O(k^{-1})$, we can get our main result that almost all YY_{2k} are spanners.

▶ **Theorem 2** (restated). *For any integer $k \ge 42$, YY_{2k} is a t_k-spanner, where $t_k = \tau'_k \tau_{2k} = \sqrt{2}(1-2\sin(\pi/8))^{-1} + O(k^{-1}) = 6.03 + O(k^{-1})$.*

5 Conclusion and Future Work

In this paper we proved that Yao-Yao graphs YY_{2k} are geometric spanners for k large enough, making a positive progress to the long-standing Conjecture 1. For Yao-Yao graphs with odd parameters, the resolution of the conjecture is still elusive and seems to require additional new ideas.

We did not try very hard to optimize the constant stretch ratio for different k values. Hence, the constant claimed in Theorem 2 may well be further improved. One potential approach is to use techniques developed in [1]. Obtaining tighter bounds (and lower bounds as well!) is left as an interesting future work.

We propose some other potential future directions, which are interesting in their own rights and might lead to sparse spanners with bounded degrees and small stretch factors.

- Consider the variant of Yao-Yao graphs where the cones are not divided uniformly. One could expect that such variants are spanners if the apex angles of the cones are all small. It seems that our techniques in the paper can be adapted to this situation if the separated cones are *centrosymmetric*: every separation ray of polar angle φ has a corresponding one at polar angle $\pi + \varphi$. But the more general case is still open.
- Instead of insisting on one polar division for every node, we can let each point choose its polar axis individually by uniformly and independently randomizing a direction. We conjecture that such graphs are also spanners (in expectation or with high probability) for k large enough.

References

1 Luis Barba, Prosenjit Bose, Jean-Lou De Carufel, Mirela Damian, Rolf Fagerberg, André van Renssen, Perouz Taslakian, and Sander Verdonschot. Continuous Yao graphs. In *Proceedings of the 26th Canadian Conference on Computational Geometry, CCCG 2014, Halifax, Nova Scotia, Canada, 2014*, 2014.

2 Luis Barba, Prosenjit Bose, Mirela Damian, Rolf Fagerberg, Wah Loon Keng, Joseph O'Rourke, André van Renssen, Perouz Taslakian, Sander Verdonschot, and Ge Xia. New and improved spanning ratios for Yao graphs. *JoCG*, 6(2):19–53, 2015.

3 Matthew Bauer and Mirela Damian. An infinite class of sparse-Yao spanners. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 184–196. SIAM, 2013.

4 Prosenjit Bose, Paz Carmi, Sebastien Collette, and Michiel Smid. On the stretch factor of convex delaunay graphs. *Journal of computational geometry*, 1(1):41–56, 2010.

5 Prosenjit Bose, Mirela Damian, Karim Douïeb, Joseph O'rourke, Ben Seamone, Michiel Smid, and Stefanie Wuhrer. $\pi/2$-angle Yao graphs are spanners. *International Journal of Computational Geometry & Applications*, 22(01):61–82, 2012.

6 Prosenjit Bose, Joachim Gudmundsson, and Michiel H. M. Smid. Constructing plane spanners of bounded degree and low weight. *Algorithmica*, 42(3-4):249–264, 2005.

7 Paul Chew. There is a planar graph almost as good as the complete graph. In *Proceedings of the second annual symposium on Computational geometry*, pages 169–177. ACM, 1986.

8 Artur Czumaj and Hairong Zhao. Fault-tolerant geometric spanners. *Discrete & Computational Geometry*, 32(2):207–230, 2004.

9 Mirela Damian. A simple Yao-Yao-based spanner of bounded degree. *arXiv preprint arXiv:0802.4325*, 2008.

10 Mirela Damian, Nawar Molla, and Val Pinciu. Spanner properties of $\pi/2$-angle Yao graphs. In *Proc. of the 25th European Workshop on Computational Geometry*, pages 21–24. Citeseer, 2009.

11 Mirela Damian and Kristin Raudonis. Yao graphs span theta graphs. In *Combinatorial Optimization and Applications*, pages 181–194. Springer, 2010.

12 David P Dobkin, Steven J Friedman, and Kenneth J Supowit. Delaunay graphs are almost as good as complete graphs. In *Foundations of Computer Science, 1987., 28th Annual Symposium on*, pages 20–26. IEEE, 1987.

13 Nawar M El Molla. Yao spanners for wireless ad hoc networks. Master's thesis, Villanova University, 2009.

14 Lujun Jia, Rajmohan Rajaraman, and Christian Scheideler. On local algorithms for topology control and routing in ad hoc networks. In *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 220–229. ACM, 2003.

15 Iyad A Kanj and Ge Xia. On certain geometric properties of the Yao-Yao graphs. In *Combinatorial Optimization and Applications*, pages 223–233. Springer, 2012.

16 Xiang-Yang Li. *Wireless Ad Hoc and Sensor Networks: Theory and Applications*. Cambridge, 6 2008.

17 Xiang-Yang Li, Peng-Jun Wan, and Yu Wang. Power efficient and sparse spanner for wireless ad hoc networks. In *Computer Communications and Networks, 2001. Proceedings. Tenth International Conference on*, pages 564–567. IEEE, 2001.

18 Xiang-Yang Li, Peng-Jun Wan, Yu Wang, and Ophir Frieder. Sparse power efficient topology for wireless networks. In *System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on*, pages 3839–3848. IEEE, 2002.

19 Xiang-Yang Li and Yu Wang. Efficient construction of low weight bounded degree planar spanner. In *Computing and Combinatorics*, pages 374–384. Springer, 2003.

20 Giri Narasimhan and Michiel Smid. *Geometric spanner networks*. Cambridge University Press, 2007.

21 David Peleg and Alejandro A Schäffer. Graph spanners. *Journal of graph theory*, 13(1):99–116, 1989.

22 Christian Schindelhauer, Klaus Volbert, and Martin Ziegler. Geometric spanners with applications in wireless networks. *Computational Geometry*, 36(3):197–214, 2007.

23 Andrew Chi-Chih Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM Journal on Computing*, 11(4):721–736, 1982.

Online Non-Preemptive Scheduling in a Resource Augmentation Model Based on Duality

Giorgio Lucarelli[*1], **Nguyen Kim Thang**[†2], **Abhinav Srivastav**[‡3], **and Denis Trystram**[§4]

1 **LIG, Université Grenoble-Alpes, Saint-Martin-d'Hères, France**
2 **IBISC, Université d'Evry, Évry, France**
3 **LIG, Université Grenoble-Alpes, Saint-Martin-d'Hères, France; and Verimag, Université Grenoble-Alpes, Saint-Martin-d'Hères, France**
4 **LIG, Université Grenoble-Alpes, Saint-Martin-d'Hères, France**

Abstract

Resource augmentation is a well-established model for analyzing algorithms, particularly in the online setting. It has been successfully used for providing theoretical evidence for several heuristics in scheduling with good performance in practice. According to this model, the algorithm is applied to a more powerful environment than that of the adversary. Several types of resource augmentation for scheduling problems have been proposed up to now, including speed augmentation, machine augmentation and more recently rejection. In this paper, we present a framework that unifies the various types of resource augmentation. Moreover, it allows generalize the notion of resource augmentation for other types of resources. Our framework is based on mathematical programming and it consists of extending the domain of feasible solutions for the algorithm with respect to the domain of the adversary. This, in turn allows the natural concept of duality for mathematical programming to be used as a tool for the analysis of the algorithm's performance. As an illustration of the above ideas, we apply this framework and we propose a primal-dual algorithm for the online scheduling problem of minimizing the total weighted flow time of jobs on unrelated machines when the preemption of jobs is not allowed. This is a well representative problem for which no online algorithm with performance guarantee is known. Specifically, a strong lower bound of $\Omega(\sqrt{n})$ exists even for the offline unweighted version of the problem on a single machine. In this paper, we first show a strong negative result even when speed augmentation is used in the online setting. Then, using the generalized framework for resource augmentation and by combining speed augmentation and rejection, we present an $(1+\epsilon_s)$-speed $O(\frac{1}{\epsilon_s\epsilon_r})$-competitive algorithm if we are allowed to reject jobs whose total weight is an ϵ_r-fraction of the weights of all jobs, for any $\epsilon_s > 0$ and $\epsilon_r \in (0,1)$. Furthermore, we extend the idea for analysis of the above problem and we propose an $(1+\epsilon_s)$-speed ϵ_r-rejection $O\left(\frac{k^{(k+3)/k}}{\epsilon_r^{1/k}\epsilon_s^{(k+2)/k}}\right)$-competitive algorithm for the more general objective of minimizing the weighted ℓ_k-norm of the flow times of jobs.

1998 ACM Subject Classification F.2.2 [Nonnumerical Algorithms and Problems] Sequencing and scheduling

Keywords and phrases Online algorithms, Non-preemptive scheduling, Resource augmentation, Primal-dual

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.63

[*] Giorgio Lucarelli ais partially supported by the ANR projet Moebus n° ANR-13-INFR-0001.
[†] Nguyen Kim Thang is supported by the ANR project OATA n° ANR-15-CE40-0015-01.
[‡] Abhinav Srivastav is partially supported by the LabEx PERSYVAL Lab (ANR-11-LABX-0025-01) funded by the French program "Investissement d'avenir".
[§] Denis Trystram is partially supported by the ANR projet Moebus n° ANR-13-INFR-0001.

1 Introduction

A well-identified issue in algorithms and, in particular, in online computation is the weakness of the worst case paradigm. Summarizing an algorithm by a pathological worst case can underestimate its performance on most inputs. Many practically well-performed algorithms admit a mediocre theoretical guarantee whereas theoretically established algorithms behave poorly even on simple instances in practice. The need of more accurate models is crucial and is considered as an important question in algorithmic community. Several models have been proposed in this direction.

A *first* type of models study online problems assuming nice properties on the inputs. For example, several models, in which arrivals of requests are assumed to follow a given distribution, an unknown distribution, a Markov chain, a random order, etc, have been studied for fundamental online problems such as paging, k-server, matching, Steiner tree. Other models that assume properties on inputs include the access graph model [5], the diffuse adversary model [22], the statistical adversary model [25], the working set model [1, 10]. A *second* type of models consists of giving more power to online algorithms and compare the online algorithm (with additional power) to the offline optimum (without additional power). This class consists of the model with advice [12, 13] and the resource augmentation model [20, 24]. A *third* type of models aim at comparing an online algorithm to some benchmark different from the offline optimum. This class includes the comparative analysis [22], the bijective analysis [3], etc. Each model has successfully explained the performance of algorithms in certain contexts but it has limits against other classes of problems. The lack of appropriate tools is a primary obstacle for the advance of most of the above models.

In this paper, we are interested in studying the resource augmentation model that compares online algorithms to a weaker adversary. Kalyanasundaram and Pruhs [20] proposed a *speed augmentation* model, where an online algorithm is compared against an adversary with slower processing speed. Phillips et al. [24] proposed the *machine augmentation* model in which the algorithm has more machines than the adversary. Recently, Choudhury et al. [8] introduced the *rejection model* where an online algorithm is allowed to discard a small fraction of jobs. The power of these models lies in the fact that many natural scheduling algorithms can be analyzed with respect to them, as well as, they have successfully provided theoretical evidence for heuristics in scheduling with good performance in practice. Although the models give more power to online algorithms, the connection especially between the latter and the two formers is unclear and the disconnection is emphasized by the fact that some algorithms have good performance in a model but have moderate behavior in others (for example, the problem of minimizing maximum flow-time [8]).

1.1 Generalized Resource Augmentation and Approach

In this paper, we introduce a generalized resource augmentation model that unifies all the previous ones. We also consider an approach based on duality for the systematic study of algorithms in this new model. To see that the duality is particularly appropriate, we first explain the model and the approach intuitively.

The weak duality in mathematical programming can be interpreted as a game between an algorithm and an adversary (the primal program against the dual one). The game is $L(x, \lambda)$, the standard Lagrangian function completely defined for a given problem, in which x and λ are primal and dual variables, respectively. The primal and dual variables are controlled and correspond to the strategies of the adversary and the algorithm, respectively. The goal of the algorithm is to choose a strategy λ among its feasible sets so as to minimize $L(x, \lambda)$ for whatever feasible strategy x of the adversary. The algorithm is c-competitive if it can choose dual variables λ^* in such a way that whatever the strategy (choice on x) of the adversary, the value $\min_x L(x, \lambda^*)$ is always within a desirable factor c of the objective due to the algorithm.

The resource augmentation models [8, 20] consist in giving more power to the algorithm. This idea could be perfectly interpreted as a game between an algorithm and an adversary in which additional power for the algorithm is reflected by better choices over its feasible strategy set.

Concretely, let us illustrate this idea for the speed augmentation and the rejection models. In several scheduling problems, a constraint originally states that the speed of a given machine is at most one. In the speed augmentation model, this constraint is relaxed such that the algorithm executes jobs at higher speed than that of the adversary. On other hand, the relaxation is of a different nature in the rejection model. Specifically, there are usually constraints ensuring that all jobs should be completed. In the rejection model, the algorithm is allowed to systematically reject a fraction of constraints whereas adversary should satisfy all of them. In both models, the algorithm optimizes the objective over a feasible domain whereas the adversary optimizes the same objective over a sub-domain with respect to the algorithm. This naturally leads to a more general model of resource augmentation.

▶ **Definition 1** (Generalized Resource Augmentation)**.** Consider an optimization problem that can be formalized by a mathematical program. Let $\mathcal{P}$ be the set of feasible solutions of the program and let $\mathcal{Q}$ be a subset of $\mathcal{P}$. In *generalized resource augmentation*, the performance of an algorithm is measured by the worst ratio between its objective over $\mathcal{P}$ and that of a solution which is optimized over $\mathcal{Q}$.

Based on the above definition, the polytope of the adversary in speed augmentation model is a strict subset of the algorithm's polytope since the speed constraint for the adversary is tighter. In the rejection model, the polytope of the adversary is also a strict subset of the algorithm's one since it contains more constraints. In addition, the generalized model allows us to introduce different kind of relaxations to the set of feasible solutions – each corresponding to different type of augmentations.

Together with the generalized model, we consider the following duality-based approach for the systematic design and analysis of algorithms. Let $\mathcal{P}$ and $\mathcal{Q}$ be the sets of feasible solutions for the algorithm and the adversary, respectively. By resource augmentation, $\mathcal{Q} \subset \mathcal{P}$. In order to study the performance of an algorithm, we consider the dual of the mathematical program consisting of the objective function optimized over $\mathcal{Q}$. By weak duality, the dual is a lower bound for any solution. Then, we bound the algorithm's cost by that of this dual. We exploit the resource augmentation properties (relation between $\mathcal{P}$ and $\mathcal{Q}$) to derive effective bounds. Intuitively, one needs to take the advantage from resource augmentation so as to raise the dual as much as possible — an impossible procedure without resource augmentation. As it has been shown in previous works and as we will see below, the duality approach is particularly appropriate to study problems with resource augmentation.

1.2 Our Contributions

We illustrate the applicability of the generalized model and the duality-based approach through a scheduling problem, in which jobs arrive online and they have to be scheduled non-preemptively on a set of unrelated machines. The objective is to minimize the average weighted time a job remains in the system (average weighted flow-time), where the weights represent the importance of the jobs. This is a well representative hard problem since no online algorithm with performance guarantee is known. Specifically, a strong lower bound of

$\Omega(\sqrt{n})$ exists even for the offline unweighted version of the problem on a single machine [21], where n is the number of jobs. For the online setting, any algorithm without resource augmentation has at least $\Omega(n)$ competitive ratio, even for single machine (as mentioned in [7]). Moreover, in contrast to the preemptive case, our first result (Lemma 2) shows that no deterministic algorithm has bounded competitive ratio when preemptions are not allowed even if we consider a single machine which has arbitrary large speed augmentation. However, the non-preemptive scheduling is a natural setting and it is important to have algorithms with theoretical explanation on their performance or a mean to classify algorithms.

In this paper, we present a competitive algorithm in a model which combines speed augmentation and the rejection model. Specifically, for arbitrary $0 < \epsilon_r < 1$ and $\epsilon_s > 0$, there exists a $O(1/(\epsilon_r \cdot \epsilon_s))$-competitive algorithm that uses machines with speed $(1 + \epsilon_s)$ and rejects jobs with at most ϵ_r-fraction of the total weight of all jobs. The design and analysis of the algorithm follow the duality approach. At the release time of any job j, the algorithm defines the dual variables associated to the job and assigns the job to some machine based on this definition. The value of the dual variables associated to a job j are selected in order to satisfy two key properties: (i) comprise the marginal increase of the total weighted flow-time due to the arrival of the job — the property that has been observed [2, 26] and has become more and more popular in dual-fitting for online scheduling; and (ii) capture the information for a future decision of the algorithm whether job j will be completed or rejected — a *novel* point in the construction of dual variables to exploit the power of rejection. Informally, to fulfill the second property, we introduce *prediction* terms to dual variables that at some point in the future will indicate whether the corresponding job would be rejected. Moreover, these terms are chosen so as to stabilize the schedule such that the properties of the assignment policy are always preserved (even with job rejections in the future). This allows us to maintain a non-migratory schedule.

Our algorithm dispatches jobs immediately at their release time — a desired property in scheduling. Besides, the algorithm processes jobs in the highest density first manner and interrupts a job only if it is rejected. In other words, no completed job has been interrupted during its execution. The algorithm is relatively simple, particularly for a single machine setting as there is no assignment policy. Therefore, the analysis of the algorithm in the generalized resource augmentation could be considered as a first step toward the theoretical explanation for the well-known observation that simple scheduling algorithms usually behave well and are widely used in practice.

Finally, we extend the above ideas to the more general objective of minimizing the weighted ℓ_k-norm of flow-time of jobs on unrelated machines. The ℓ_k-norm captures the notion of fairness between jobs since it removes the extreme outliers and hence it is more appropriate to balance the difference among the flow-times of individual jobs than the average function, which corresponds to the ℓ_1-norm (see for example [23]). For the ℓ_k-norm objective, we propose a primal-dual algorithm which is $(1 + \epsilon_s)$-speed $O\left(\frac{k^{(k+3)/k}}{\epsilon_r^{1/k}\epsilon_s^{(k+2)/k}}\right)$-competitive and it rejects jobs of total weight at most ϵ_r-fraction of the total weight of all jobs. The analysis for this problem is more technical and it is given in the Appendix.

1.3 Related Work

Duality based techniques have been extensively developed in approximation algorithms [27] and in online algorithms [6]. Specifically, Buchbinder and Naor [6] gave a general framework for online covering/packing LPs that applies to several fundamental problems in online computation. However, this framework encounters different issues to design competitive algorithms for online scheduling problems. Recently, Anand et al. [2] have proposed the use of dual-fitting techniques to study scheduling problems in the speed augmentation model. After this seminal paper, the duality approaches in online scheduling have been extended to a variety of problems, and has rapidly become standard techniques. The duality approaches have also led to the development of newer techniques for analyzing algorithm (see for example [2, 11, 15, 16, 18, 17, 26]). Informally, in the approach proposed in [2], the key step relies on the construction of a dual feasible solution in such a way that its dual objective is up to some bounded factor from that of the algorithm. In [2], the dual variables are carefully designed in order to encode the power of speed augmentation. Later on, Nguyen [26] explicitly formalized the comparison through the mean of Lagrangian functions between the algorithm and the adversary, with a tighter feasible domain due to speed augmentation. That point of view makes the framework in [2] effective to study non-convex formulations.

For the online non-preemptive scheduling problem of minimizing total weighted flow-time, no competitive algorithm for unrelated machines even with resource augmentation is known; that is in contrast to the preemptive version which has been well studied [2, 11, 16, 18, 17, 26]. For identical machines, Phillips et al. [24] gave a constant competitive algorithm that uses $m \log P$ machines (recall that the adversary uses m machines), where P is the ratio of the largest to the smallest processing time. Moreover, an $O(\log n)$-machine $O(1)$-speed algorithm that returns the optimal schedule has been presented in [24] for the unweighted flow-time objective. Epstein and van Stee [14] proposed an ℓ-machines $O(\min\{\sqrt[\ell]{P}, \sqrt[\ell]{n}\})$-competitive algorithm for the unweighted case on a single machine. This algorithm is optimal up to a constant factor for constant ℓ. For the offline non-preemptive single machine setting, Bansal et al. [4] gave a 12-speed $(2+\epsilon)$-approximation polynomial time algorithm. Recently, Im et al. [19] gave a $(1+\epsilon)$-speed $(1+\epsilon)$-approximation quasi-polynomial time algorithm for the setting of identical machines.

For the online non-preemptive scheduling problem of minimizing the weighted ℓ_k-norm of flow-time, to the best of our knowledge, no competitive algorithm is known. However, the problem in the preemptive setting has been widely studied. With speed augmentation, Anand et al. [2] gave a $(1+\epsilon)$-speed, $O(k/\epsilon^{2+1/k})$-competitive algorithm but the algorithm needs to know the speed $(1+\epsilon)$ in advance. Later on, Nguyen [26] derived an improved $(1+\epsilon)$-speed, $k/\epsilon^{1+1/k}$-competitive algorithm which does not need information on ϵ a priori. Recently, Choudhury et al. [9] have considered the (preemptive) problem in the restricted assignment setting in the rejection model. They have presented a $1/\epsilon^{O(1)}$-competitive algorithm that rejects at most ϵ fraction of the total job weight.

2 Problem Definition and Notation

We are given a set $\mathcal{M}$ of m unrelated machines. The jobs arrive *online*, that is we learn about the existence and the characteristics of a job only after its release. Let $\mathcal{J}$ denote the set of all jobs of our instance, which is not known a priori. Each job $j \in \mathcal{J}$ is characterized by its *release time* r_j, its *weight* w_j and if job j is executed on machine $i \in \mathcal{M}$ then it has a *processing time* p_{ij}. We study the *non-preemptive* setting, meaning that a job is considered to be completed only if it is fully processed in one machine without interruption during its execution. This definition allows the interruption of jobs. However, if the execution of a job is interrupted then it has to be processed entirely later on in order to be considered as completed. In this paper, we consider a stronger non-preemptive model according to which we are only allowed to interrupt a job if we reject it, i.e., we do not permit restarts. Moreover, each job has to be dispatched to one machine at its arrival and migration is not allowed. Given a schedule $\mathcal{S}$, we denote by C_j the *completion time* of the job j. Then, its *flow-time* is defined as $F_j = C_j - r_j$, that is the total time that j remains in the system. Our objective is to create a non-preemptive schedule that minimizes the total weighted flow-times of all jobs, i.e., $\sum_{j \in \mathcal{J}} w_j F_j$. A more general objective that implies fairness between jobs is the minimization of the weighted ℓ_k-norm of the flow-time of all jobs, i.e., $(\sum_{j \in \mathcal{J}} w_j F_j^k)^{1/k}$, where $k \geq 1$.

Let $\delta_{ij} = \frac{w_j}{p_{ij}}$ be the *density* of a job $j \in \mathcal{J}$ on machine $i \in \mathcal{M}$. Moreover, let $q_{ij}(t)$ be the remaining processing time at time t of a job $j \in \mathcal{J}$ which is dispatched at machine $i \in \mathcal{M}$. A job $j \in \mathcal{J}$ is called *pending* at time t, if it is already released at t but not yet completed, i.e., $r_j \leq t < C_j$. Finally, let $P = \max_{j,j' \in \mathcal{J}} \{p_j / p_{j'}\}$ and $W = \max_{j,j' \in \mathcal{J}} \{w_j / w_{j'}\}$.

3 Scheduling to Minimize Total Weighted Flow-time

In this section, we describe our primal-dual method for the online non-preemptive scheduling problem of minimizing the total weighted flow-time on unrelated machines. This problem admits no competitive algorithm even with speed augmentation as shown by the following lemma.

▶ **Lemma 2.** *For any speed augmentation $s \leq P^{1/10}$ or $s < W^{1/6}$, every deterministic algorithm has competitive ratio at least $\Omega(P^{1/10})$ or $\Omega(W^{1/6})$, respectively, even for the single machine problem.*

Proof. Let $s > 1$ be the speed of the machine; without loss of generality we assume that the machine speed for the adversary is 1. Let $R > s^2$ be an arbitrary (large) constant and fix an algorithm.

We consider the following instance. At time 1, a *long* job of processing time $2sR^3$ and weight 1 is released. After that, the phase 1 starts: at any time aR^3, starting with $a = 1$, a *short* job of processing time 1 and weight R is released. If the algorithm processes the long job during the whole interval $[aR^3, (a+1)R^3]$, then the adversary stops releasing jobs and the instance halts. Otherwise, the adversary will release a new short job at time $(a+1)R^3$ and so on, until $a = 2s - 1$. Then, the phase 2 begins immediately after phase 1: at any time aR^3 for $a \geq 2s$, a *small* job of processing time 1 and weight R^2 is released. Similarly, if the algorithm keeps processes the long job during the whole interval $[aR^3, (a+1)R^3]$, the instance halts. Otherwise, the adversary will release a new small job at time $(a+1)R^3$, until $a = 2sR^2$.

In the instance, we have at most $2s$ short jobs and $2sR^2$ small jobs. Observe that by using speed s, the algorithm cannot complete the long job between two consecutive release times of short or small jobs. We analyze the performance of the algorithm by considering different cases.

Case 1: the instance halts during phase 1. In this case, there is a $a \in \{1, 2, \ldots, 2s-1\}$ for which the algorithm keeps processing the long job during the whole interval $[aR^3, (a+1)R^3]$ and hence the short job released at aR^3 is not processed during that time interval. Thus, the weighted flow-time of the short job is at least $R \cdot R^3$. However, the adversary can execute immediately all short jobs at their release times and process the long job in the end. The total weighted flow-time of all short jobs is at most $2sR$. The long job would be started no later than the time where phase 1 terminates, which is $(2s-1)R^3 + 1$. So the weighted flow-time of the long job is at most $4sR^3$. Therefore, the competitive ratio is at least $\Omega(R/s)$.

Case 2: the instance halts during phase 2. In this case, there is a $a \in \{2s, 2s+1, \ldots, 2sR^2\}$ for which the algorithm keeps processing the long job during the whole interval $[aR^3, (a+1)R^3]$ and hence the small job released at aR^3 is not processed during that time interval. We proceed similarly as in the previous case. The weighted flow-time of this small job is at least $R^2 \cdot R^3$. Nevertheless, the adversary can process the long job during $[1, 2sR^3 + 1]$, execute small jobs at their release time (except the first one which starts 1 unit of time after its release time) and execute all short jobs during the interval $[2sR^3 + 2, 5sR^3]$ whenever a small job is not executed. This is a feasible schedule since the number of short jobs is $(2s - 1) < 3R^3 - 5$ (note that there are 2 small jobs released during $[2sR^3 + 2, 5sR^3]$). By this strategy, the weighted flow-time of the long job is $2sR^3 + 1$. The total weighted flow-time of small jobs is at most $2sR^2 \cdot R^2$. The total weighted flow-time of short jobs is at most $2s \cdot R \cdot 5sR^3$. Hence, the cost of the adversary is at most $14s^2R^4$ and the competitive ratio is at least $\Omega(R/s^2)$.

Case 3: the instance halts at the end of phase 2. The algorithm executes the long job after the end of phase 2 and hence this job is completed at later than $2sR^5$; so its weighted flow-time is at least $2sR^5$. The adversary can apply the same strategy as in Case 2 with total cost $14s^2R^4$. Therefore, the competitive ratio is at least $\Omega(R/s)$.

In summary, the competitive ratio is at least $\Omega(R/s^2)$. Recall that P and W are the largest ratio between processing times and that between weights, respectively. In this instance, $P = 2sR^3$ and $W = R^2$ respectively. By a simple estimation (setting $R = s^3$), for any speed $s \leq P^{1/10}$ the competitive ratio is at least $\Omega(P^{1/10})$; and for $s \leq W^{1/6}$, the competitive ratio is at least $\Omega(W^{1/6})$. ◀

In the following, we study the problem in the resource augmentation model with speed augmentation and rejection.

3.1 Linear Programming Formulation

For each machine $i \in \mathcal{M}$, job $j \in \mathcal{J}$ and time $t \geq r_j$, we introduce a binary variable $x_{ij}(t)$ which indicates if j is processed on i at time t. We consider the following linear programming formulation. Note that the objective value of this linear program is at most twice that of the optimal preemptive schedule.

$$\min \sum_{i\in\mathcal{M}} \sum_{j\in\mathcal{J}} \int_{r_j}^{\infty} \delta_{ij}\,(t - r_j + p_{ij}) x_{ij}(t)dt$$

$$\sum_{i\in\mathcal{M}} \int_{r_j}^{\infty} \frac{x_{ij}(t)}{p_{ij}} dt \geq 1 \quad \forall j \in \mathcal{J} \tag{1}$$

$$\sum_{j\in\mathcal{J}} x_{ij}(t) \leq 1 \quad \forall i \in \mathcal{M}, t \tag{2}$$

$$x_{ij}(t) \in \{0,1\} \quad \forall i \in \mathcal{M}, j \in \mathcal{J}, t \geq r_j$$

After relaxing the integrality constraints, we get the following dual program.

$$\max \sum_{j\in\mathcal{J}} \lambda_j - \sum_{i\in\mathcal{M}} \int_0^{\infty} \gamma_i(t)dt$$

$$\frac{\lambda_j}{p_{ij}} - \gamma_i(t) \leq \delta_{ij}\,(t - r_j + p_{ij}) \quad \forall i \in \mathcal{M}, j \in \mathcal{J}, t \geq r_j \tag{3}$$

We will interpret the resource augmentation models in the above primal and dual programs as follows. In the speed augmentation model, we assume that all machines in the schedule of our algorithm run with speed 1, while in adversary's schedule they run at a speed $a < 1$. This can be interpreted in the primal linear program by modifying the constraint (2) to be $\sum_{j\in\mathcal{J}} x_{ij}(t) \leq a$. Intuitively, each machine in the adversary's schedule can execute jobs with speed at most a at each time t. The above modification in the primal program reflects to the objective of the dual program which becomes $\sum_{j\in\mathcal{J}} \lambda_j - a\sum_{i\in\mathcal{M}} \int_0^\infty \gamma_i(t)dt$. In the rejection model, we assume that the algorithm is allowed to reject some jobs. This can be interpreted in the primal linear program by summing up only on the set of the non rejected jobs, i.e., the algorithm does not have to satisfy the constraint (1) for rejected jobs. Hence the objective becomes $\sum_{i\in\mathcal{M}}\sum_{j\in\mathcal{J}\setminus\mathcal{R}} \int_{r_j}^\infty \delta_{ij}\left(t - r_j + p_{ij}\right)dt$. Concluding, our algorithm rejects a set $\mathcal{R}$ of jobs, uses machines with speed $1/a$ times faster than that of the adversary and, by using weak duality, has a competitive ratio at most

$$\frac{\sum_{i\in\mathcal{M}}\sum_{j\in\mathcal{J}\setminus\mathcal{R}} \int_{r_j}^\infty \delta_{ij}(t - r_j + p_{ij})dt}{\sum_{j\in\mathcal{J}} \lambda_j - a\sum_{i\in\mathcal{M}} \int_0^\infty \gamma_i(t)dt}.$$

3.2 Algorithm and Dual Variables

We describe next the scheduling, the rejection and the dispatching policies of our algorithm which we denote by $\mathcal{A}$. In parallel, we give the intuition about the definition of the dual variables in a primal-dual way. Let $\epsilon_s > 0$ and $0 < \epsilon_r < 1$ be constants arbitrarily small. Intuitively, ϵ_s and ϵ_r stand for the speed augmentation and the rejection fraction of our algorithm, respectively. In what follows, we assume that in the schedule created by $\mathcal{A}$ all machines run with speed 1, while in the adversary's schedule they run by speed $\frac{1}{1+\epsilon_s}$.

Each job is immediately dispatched to a machine upon its arrival. We denote by $Q_i(t)$ the set of pending jobs at time t dispatched to machine $i \in \mathcal{M}$, i.e., the set of jobs dispatched to i that have been released but not yet completed and have not been rejected at t. Our *scheduling policy* for each machine $i \in \mathcal{M}$ is the following: at each time t when the machine i becomes idle or has just completed or interrupted some job, we start executing on i the job $j \in Q_i(t)$ such that j has the largest density in $Q_i(t)$, i.e., $j = \text{argmax}_{j'\in Q_i(t)}\{\delta_{ij'}\}$. In case of ties, we select the job that arrived earliest.

When a machine $i \in \mathcal{M}$ starts executing a job $k \in \mathcal{J}$, we introduce a counter v_k (associated to job k) which is initialized to zero. Each time when a job $j \in \mathcal{J}$ with $\delta_{ij} > \delta_{ik}$ is released during the execution of k and j is dispatched to i, we increase v_k by w_j. Then, the *rejection policy* is the following: we interrupt the execution of the job k and we reject it the first time where $v_k > \frac{w_k}{\epsilon_r}$.

Let Δ_{ij} be the increase in the total weighted flow-time occurred in the schedule of our algorithm if we assign a new job $j \in \mathcal{J}$ to machine i, following the above scheduling and rejection policies. Assuming that the job $k \in \mathcal{J}$ is executed on i at time r_j, we have that

$$\Delta_{ij} = \begin{cases} w_j\left(q_{ik}(r_j) + \displaystyle\sum_{\substack{\ell\in Q_i(r_j)\setminus\{k\}:\\ \delta_{i\ell}\geq\delta_{ij}}} p_{i\ell}\right) + p_{ij}\displaystyle\sum_{\substack{\ell\in Q_i(r_j)\setminus\{k\}:\\ \delta_{i\ell}<\delta_{ij}}} w_\ell & \text{if } v_k + w_j \leq \frac{w_k}{\epsilon_r}, \\ w_j\displaystyle\sum_{\substack{\ell\in Q_i(r_j):\\ \delta_{i\ell}\geq\delta_{ij}}} p_{i\ell} + \left(p_{ij}\displaystyle\sum_{\substack{\ell\in Q_i(r_j):\\ \delta_{i\ell}<\delta_{ij}}} w_\ell - q_{ik}(r_j)\displaystyle\sum_{\substack{\ell\in Q_i(r_j)\cup\{k\}:\\ \ell\neq j}} w_\ell\right) & \text{otherwise.} \end{cases}$$

where, in both cases, the first term corresponds to the weighted flow-time of the job j if it is dispatched to i and the second term corresponds to the change of the weighted flow-time for the jobs in $Q_i(r_j)$. Note that, the second case corresponds to the rejection of k and hence we remove the term $w_j q_{ik}(r_j)$ in the weighted flow-time of j, while the flow-time of each pending job is reduced by $q_{ik}(r_j)$.

In the definition of the dual variables, we aim to charge to job j the increase Δ_{ij} in the total weighted flow-time occurred by the dispatching of j in machine i, except from the quantity $w_j q_{ik}(r_j)$ which will be charged to job k, if $\delta_{ij} > \delta_{ik}$. However, we will use the dual variables (in the primal-dual sense) to guide the assignment policy. Hence the charges have to be distributed in a consistent manner to the assignment decisions of jobs to machines in the past. So in order to do the charging, we introduce a *prediction term*: at the arrival of each job j we charge to it an additional quantity of $\frac{w_j}{\epsilon_r} p_{ij}$. By doing this, the consistency is maintained by the rejection policy: if the charge from future jobs exceeds the prediction term of some job then the latter will be rejected.

Based on the above, we define

$$\lambda_{ij} = \begin{cases} \frac{w_j}{\epsilon_r} p_{ij} + w_j \sum\limits_{\ell \in Q_i(r_j):\delta_{i\ell} \geq \delta_{ij}} p_{i\ell} + p_{ij} \sum\limits_{\ell \in Q_i(r_j)\setminus\{k\}:\delta_{i\ell} < \delta_{ij}} w_\ell & \text{if } \delta_{ij} > \delta_{ik} \\ \frac{w_j}{\epsilon_r} p_{ij} + w_j \left(q_{ik}(r_j) + \sum\limits_{\ell \in Q_i(r_j)\setminus\{k\}:\delta_{i\ell} \geq \delta_{ij}} p_{i\ell} \right) + p_{ij} \sum\limits_{\ell \in Q_i(r_j):\delta_{i\ell} < \delta_{ij}} w_\ell & \text{otherwise} \end{cases}$$

which represents the total charge for job j if it is dispatched to machine i. Note that the only difference in the two cases of the definition of λ_{ij} is that we charge the quantity $w_j q_{ik}(r_j)$ to j only if $\delta_{ij} \leq \delta_{ik}$. Moreover, we do not consider the negative quantity that appears in the second case of Δ_{ij}. Intuitively, we do not decrease our estimation for the completion times of pending jobs when a job is rejected. The *dispatching policy* is the following: dispatch j to the machine $i^* = \text{argmin}_{i \in \mathcal{M}}\{\lambda_{ij}\}$. Intuitively, a part of Δ_{ij} may be charged to job k, and more specifically to the prediction part of λ_{ik}. However, we do not allow to exceed this prediction by applying rejection. In other words, the rejection policy can be re-stated informally as: we reject k just before we exceed the prediction charging part in λ_{ik}.

In order to keep track of the negative terms in Δ_{ij}, for each job $j \in \mathcal{J}$ we denote by D_j the set of jobs that are rejected by the algorithm after the release time of j and before its completion or rejection (including j in case it is rejected), that is the jobs that cause a decrease to the flow time of j. Moreover, we denote by j_k the job released at the moment we reject a job $k \in \mathcal{R}$. Then, we say that a job $j \in \mathcal{J}$ which is dispatched to machine $i \in \mathcal{M}$ is *definitively finished* $\sum_{k \in D_j} q_{ik}(r_{j_k})$ time after its completion or rejection. Let $U_i(t)$ be the set of jobs that are dispatched to machine $i \in \mathcal{M}$, they are already completed or rejected at a time before t, but they are not yet definitively finished at t.

It remains to formally define the dual variables. At the arrival of a job $j \in \mathcal{J}$, we set $\lambda_j = \frac{\epsilon_r}{1+\epsilon_r} \min_{i \in \mathcal{M}}\{\lambda_{ij}\}$ and we never change λ_j again. Let $W_i(t)$ be the total weight of jobs dispatched to machine $i \in \mathcal{M}$ in the schedule of $\mathcal{A}$, and either they are pending at t or they are not yet definitively finished at t, i.e., $W_i(t) = \sum_{\ell \in Q_i(t) \cup U_i(t)} w_\ell$. Then, we define $\gamma_i(t) = \frac{\epsilon_r}{1+\epsilon_r} W_i(t)$. Note that $\gamma_i(t)$ is updated during the execution of $\mathcal{A}$. Specifically, given any fixed time t, $\gamma_i(t)$ may increase if a new job j' arrives at any time $r_{j'} \in [r_j, t)$. However, $\gamma_i(t)$ does never decrease in the case of rejection since the jobs remain in $U_i(t)$ for a sufficient time after their completion or rejection.

3.3 Analysis

We first prove the following lemma which guarantees the feasibility of the dual constraint using the above definition of the dual variables.

▶ **Lemma 3.** *For every machine $i \in \mathcal{M}$, job $j \in \mathcal{J}$ and time $t \geq r_j$, the dual constraint is feasible, that is*

$$\frac{\lambda_j}{p_{ij}} - \gamma_i(t) - \delta_{ij}\left(t - r_j + p_{ij}\right) \leq 0\,.$$

Proof. Fix a machine i. We have observed above that, for any fixed time $t \geq r_j$, as long as new jobs arrive, the value of $\gamma_i(t)$ may only increase. Then, it is sufficient to prove the dual constraints for the job j using the values of $\gamma_i(t)$, $Q_i(t)$, $U_i(t)$ and $W_i(t)$ as these are defined at time r_j. Let k be the job executed in machine i at r_j. We have the following cases.

Case 1: $\delta_{ij} > \delta_{ik}$

In this case we may have rejected the job k at r_j. By the definitions of λ_j and λ_{ij}, we have

$$\begin{aligned}\frac{\lambda_j}{p_{ij}} &\leq \frac{\epsilon_r}{(1+\epsilon_r)}\frac{\lambda_{ij}}{p_{ij}} = \frac{\epsilon_r}{1+\epsilon_r}\left(\frac{w_j}{\epsilon_r} + \delta_{ij}\sum_{\ell\in Q_i(r_j):\delta_{i\ell}\geq\delta_{ij}} p_{i\ell} + \sum_{\ell\in Q_i(r_j)\setminus\{k\}:\delta_{i\ell}<\delta_{ij}} w_\ell\right)\\ &= \frac{\epsilon_r}{1+\epsilon_r}\left(\frac{w_j}{\epsilon_r} + \delta_{ij}\sum_{\ell\in Q_i(r_j)\setminus\{j\}:\delta_{i\ell}\geq\delta_{ij}} p_{i\ell} + w_j + \sum_{\ell\in Q_i(r_j)\setminus\{k\}:\delta_{i\ell}<\delta_{ij}} w_\ell\right)\\ &= w_j + \frac{\epsilon_r}{1+\epsilon_r}\left(\delta_{ij}\sum_{\ell\in Q_i(r_j)\setminus\{j\}:\delta_{i\ell}\geq\delta_{ij}} p_{i\ell} + \sum_{\ell\in Q_i(r_j)\setminus\{k\}:\delta_{i\ell}<\delta_{ij}} w_\ell\right)\end{aligned}$$

By the definition of $\gamma_i(t)$ we get

$$\begin{aligned}\gamma_i(t) + \delta_{ij}(t - r_j + p_{ij}) &= \frac{\epsilon_r}{1+\epsilon_r}\sum_{\ell\in Q_i(t)\cup U_i(t)} w_\ell + \delta_{ij}(t-r_j) + w_j\\ &\geq \frac{\epsilon_r}{1+\epsilon_r}\left(\sum_{\ell\in Q_i(t)\cup U_i(t)} w_\ell + \delta_{ij}(t-r_j)\right) + w_j\end{aligned}$$

Thus, it remains to show that

$$\delta_{ij}\cdot\sum_{\substack{\ell\in Q_i(r_j)\setminus\{j\}:\\ \delta_{i\ell}\geq\delta_{ij}}} p_{i\ell} + \sum_{\substack{\ell\in Q_i(r_j)\setminus\{k\}:\\ \delta_{i\ell}<\delta_{ij}}} w_\ell \leq \sum_{\ell\in Q_i(t)\cup U_i(t)} w_\ell + \delta_{ij}(t-r_j) \qquad (4)$$

Let $\tilde{C}_j = r_j + \sum_{\ell\in Q_i(r_j):\delta_{i\ell}\geq\delta_{ij}} p_{i\ell}$ (if k is rejected) or $\tilde{C}_j = r_j + q_{ik}(r_j) + \sum_{\ell\in Q_i(r_j):\delta_{i\ell}\geq\delta_{ij}} p_{i\ell}$ (otherwise) be the estimated completion time of j at time r_j if it is dispatched to machine i.

Case 1.1: $t \leq \tilde{C}_j$. By the definition of $U_i(t)$, all jobs in $Q_i(r_j)$ with $\delta_{i\ell} < \delta_{ij}$ still exist in $Q_i(t)\cup U_i(t)$. Moreover, for every job $\ell \in Q_i(r_j)\setminus(Q_i(t)\cup U_i(t)\cup\{k\})$ it holds that $\delta_{i\ell}\geq\delta_{ij}$, since ℓ is processed before j by the algorithm. Then, by splitting the first term of the left-hand side of (4) we get

$$\begin{aligned}
&\delta_{ij} \cdot \sum_{\ell \in Q_i(r_j)\setminus\{j\}:\delta_{i\ell}\geq\delta_{ij}} p_{i\ell} + \sum_{\ell\in Q_i(r_j)\setminus\{k\}:\delta_{i\ell}<\delta_{ij}} w_\ell \\
&\quad = \delta_{ij} \sum_{\ell\in Q_i(r_j)\setminus(Q_i(t)\cup U_i(t)\cup\{k\})} p_{i\ell} + \delta_{ij} \sum_{\substack{\ell\in(Q_i(r_j)\cap(Q_i(t)\cup U_i(t)))\setminus\{j\}:\\ \delta_{i\ell}\geq\delta_{ij}}} p_{i\ell} \\
&\qquad\qquad + \sum_{\substack{\ell\in(Q_i(r_j)\cap(Q_i(t)\cup U_i(t)))\setminus\{k\}:\\ \delta_{i\ell}<\delta_{ij}}} w_\ell \\
&\quad \leq \delta_{ij} \sum_{\ell\in Q_i(r_j)\setminus(Q_i(t)\cup U_i(t)\cup\{k\})} p_{i\ell} + \sum_{\substack{\ell\in(Q_i(t)\cup U_i(t))\setminus\{j\}:\\ \delta_{i\ell}\geq\delta_{ij}}} w_\ell + \sum_{\substack{\ell\in(Q_i(t)\cup U_i(t))\setminus\{k\}:\\ \delta_{i\ell}<\delta_{ij}}} w_\ell \\
&\quad \leq \delta_{ij}(t-r_j) + \sum_{\ell\in Q_i(t)\cup U_i(t)} w_\ell
\end{aligned}$$

where the first inequality is due to $\delta_{ij}p_{i\ell} \leq w_\ell$ for each $\ell \in Q_i(t)\cup U_i(t)$ with $\delta_{i\ell}\geq\delta_{ij}$, while the latter one holds since the set of jobs $Q_i(r_j)\setminus(Q_i(t)\cup U_i(t)\cup\{k\})$ corresponds to the set of pending jobs at r_j that start their execution after r_j and are definitively finished before t.

Case 1.2: $t > \tilde{C}_j$. By splitting the second term of the left-hand side of (4) we get

$$\begin{aligned}
&\delta_{ij} \cdot \sum_{\ell \in Q_i(r_j)\setminus\{j\}:\delta_{i\ell}\geq\delta_{ij}} p_{i\ell} + \sum_{\ell\in Q_i(r_j)\setminus\{k\}:\delta_{i\ell}<\delta_{ij}} w_\ell \\
&\quad = \delta_{ij} \sum_{\substack{\ell\in Q_i(r_j)\setminus\{j\}:\\ \delta_{i\ell}\geq\delta_{ij}}} p_{i\ell} + \sum_{\substack{\ell\in Q_i(r_j)\setminus(Q_i(t)\cup U_i(t)\cup\{k\}):\\ \delta_{i\ell}<\delta_{ij}}} w_\ell + \sum_{\substack{\ell\in Q_i(r_j)\cap(Q_i(t)\cup U_i(t)):\\ \delta_{i\ell}<\delta_{ij}}} w_\ell \\
&\quad \leq \delta_{ij}(\tilde{C}_j - r_j) + \delta_{ij} \sum_{\ell\in Q_i(r_j)\setminus(Q_i(t)\cup U_i(t)\cup\{k\}):\delta_{i\ell}<\delta_{ij}} p_{i\ell} + \sum_{\ell\in Q_i(t)\cup U_i(t)} w_\ell \\
&\quad \leq \delta_{ij}(\tilde{C}_j - r_j) + \delta_{ij}(t-\tilde{C}_j) + \sum_{\ell\in Q_i(t)\cup U_i(t)} w_\ell
\end{aligned}$$

where the first inequality follows by the definition of $\tilde{C}_j$ and since $w_\ell < \delta_{ij}p_{i\ell}$ for each $\ell \in Q_i(r_j)$ with $\delta_{i\ell} < \delta_{ij}$, while the second inequality follows since the set of jobs in $Q_i(r_j)\setminus(Q_i(t)\cup U_i(t)\cup\{k\})$ with $\delta_{i\ell}<\delta_{ij}$ corresponds to the pending jobs at r_j which at time r_j have been scheduled to be executed during the interval $[\tilde{C}_j, t)$.

Case 2: $\delta_{ij} \leq \delta_{ik}$

In this case the job k is not rejected at the arrival of job j. By using the same arguments as in Case 1, we have

$$\frac{\lambda_j}{p_{ij}} \leq w_j + \frac{\epsilon_r}{1+\epsilon_r}\left(\delta_{ij}q_{ik}(r_j) + \delta_{ij}\sum_{\ell\in Q_i(r_j)\setminus\{k,j\}:\delta_{i\ell}\geq\delta_{ij}} p_{i\ell} + \sum_{\ell\in Q_i(r_j):\delta_{i\ell}<\delta_{ij}} w_\ell\right)$$

Let $\tilde{C}_k = r_j + q_{ik}(r_j)$ be the estimated completion time of k at time r_j. We consider different scenarios.

Case 2.1: $t \le \tilde{C}_k$. In this case, it holds that $w_k \ge \delta_{ij} p_k \ge \delta_{ij} q_{ik}(r_j)$. Then,

$$\gamma_i(t) + \delta_{ij}\left(t - r_j + p_{ij}\right) \ge \frac{\epsilon_r}{1+\epsilon_r} \sum_{\ell \in Q_i(t) \cup U_i(t)} w_\ell + w_j \ge \frac{\epsilon_r}{1+\epsilon_r} \sum_{\ell \in Q_i(r_j)} w_\ell + w_j$$

$$\ge \frac{\epsilon_r}{1+\epsilon_r} \left(\sum_{\ell \in Q_i(r_j) \setminus \{k\}} w_\ell + w_k \right) + w_j \ge \frac{\epsilon_r}{1+\epsilon_r} \left(\sum_{\ell \in Q_i(r_j) \setminus \{k\}} w_\ell + \delta_{ij} q_{ik}(r_j) \right) + w_j$$

Hence, it remains to show

$$\delta_{ij} \sum_{\ell \in Q_i(r_j) \setminus \{k,j\} : \delta_{i\ell} \ge \delta_{ij}} p_{i\ell} + \sum_{\ell \in Q_i(r_j) : \delta_{i\ell} < \delta_{ij}} w_\ell - \sum_{\ell \in Q_i(r_j) \setminus \{k\}} w_\ell \le 0$$

which directly holds as $\delta_{ij} p_{i\ell} \le w_\ell$ for any job $j \in Q_i(r_j)$ with $\delta_{i\ell} \ge \delta_{ij}$.

Case 2.2: $t > \tilde{C}_k$. By the definition of $\gamma_i(t)$ we get

$$\gamma_i(t) + \delta_{ij}\left(t - r_j + p_{ij}\right) \ge \frac{\epsilon_r}{1+\epsilon_r} \left(\sum_{\ell \in Q_i(t) \cup U_i(t)} w_\ell + \delta_{ij} q_{ik}(r_j) + \delta_{ij}(t - r_j) \right) + w_j$$

Hence it suffices again to prove (4), which has been proved previously. ◀

The following lemma guarantees that, by the rejection policy, the algorithm rejects at most a small fraction of the total job weight.

▶ **Lemma 4.** *For the set $\mathcal{R}$ of jobs rejected by the algorithm $\mathcal{A}$ it holds that $\sum_{j \in \mathcal{R}} w_j \le \epsilon_r \sum_{j \in \mathcal{J}} w_j$.*

Proof. Each job $j \in \mathcal{J}$ dispatched to machine $i \in \mathcal{M}$ may increase only the counter v_k of the job $k \in \mathcal{J}$ that was executed on i at r_j. In other words, each job j may be charged to at most one other job. Besides, we reject a job k the first time where $v_k > \frac{w_k}{\epsilon_r}$, meaning that the total weight of jobs charged to k is at least $\frac{w_k}{\epsilon_r}$. Hence, the total weight of rejected jobs is at most an ϵ_r-fraction of the total weight of all jobs in the instance. ◀

▶ **Theorem 5.** *Given any $\epsilon_s > 0$ and $\epsilon_r \in (0,1)$, $\mathcal{A}$ is a $(1+\epsilon_s)$-speed $\frac{2(1+\epsilon_r)(1+\epsilon_s)}{\epsilon_r \epsilon_s}$-competitive algorithm that rejects jobs of total weight at most $\epsilon_r \sum_{j \in \mathcal{J}} w_j$.*

Proof. By Lemma 3, the proposed dual variables constitute a feasible solution for the dual program. By definition, the algorithm $\mathcal{A}$ uses for any machine at any time a factor of $1+\epsilon_s$ higher speed than that of the adversary. By Lemma 4, $\mathcal{A}$ rejects jobs of total weight at most $\epsilon_r \sum_{j \in \mathcal{J}} w_j$. Hence, it remains to give a lower bound for the dual objective based on the proposed dual variables.

We denote by $F_j^{\mathcal{A}}$ the flow-time of a job $j \in \mathcal{J} \setminus \mathcal{R}$ in the schedule of $\mathcal{A}$. By slightly abusing the notation, for a job $k \in \mathcal{R}$, we will also use $F_k^{\mathcal{A}}$ to denote the total time passed after r_k until deciding to reject a job k, that is, if k is rejected at the release of the job $j \in \mathcal{J}$ then $F_k^{\mathcal{A}} = r_j - r_k$. Denote by j_k the job released at the moment we decided to reject k, i.e., for the counter v_k before the arrival of job j_k we have that $w_k/\epsilon_r - w_{j_k} < v_k < w_k/\epsilon_r$.

Let Δ_j be the total increase in the flow-time caused by the arrival of the job $j \in \mathcal{J}$, i.e., $\Delta_j = \Delta_{ij}$, where $i \in \mathcal{M}$ is the machine to which j is dispatched by $\mathcal{A}$. By the definition of

λ_j's, we have

$$\begin{aligned}\sum_{j\in\mathcal{J}}\lambda_j &\geq \frac{\epsilon_r}{1+\epsilon_r}\left(\sum_{j\in\mathcal{J}}\Delta_j + \sum_{k\in\mathcal{R}}\left(q_{ik}(r_{j_k})\sum_{\ell\in Q_i(r_{j_k})\cup\{k\}:\ell\neq j_k} w_\ell\right)\right)\\ &= \frac{\epsilon_r}{1+\epsilon_r}\left(\sum_{j\in\mathcal{J}} w_jF_j^{\mathcal{A}} + \sum_{j\in\mathcal{J}}\left(w_j\sum_{k\in D_j} q_{ik}(r_{j_k})\right)\right)\end{aligned}$$

where the inequality comes from the fact that if $\delta_{ij} > \delta_{ik}$ then in the prediction part of the running job k at r_j we charge the quantity w_jp_k instead of $w_jq_k(r_j)$ which is the real contribution of k to the weighted flow-time of job j. By the definition of $\gamma_i(t)$'s, we have

$$\begin{aligned}\sum_{i\in\mathcal{M}}\int_0^\infty \gamma_i(t)dt &= \frac{\epsilon_r}{1+\epsilon_r}\left(\sum_{i\in\mathcal{M}}\int_0^\infty\sum_{\ell\in Q_i(t)} w_\ell dt + \sum_{i\in\mathcal{M}}\int_0^\infty\sum_{\ell\in U_i(t)} w_\ell dt\right)\\ &= \frac{\epsilon_r}{1+\epsilon_r}\left(\sum_{j\in\mathcal{J}} w_jF_j^{\mathcal{A}} + \sum_{j\in\mathcal{J}}\left(w_j\sum_{k\in D_j} q_{ik}(r_{j_k})\right)\right)\end{aligned}$$

since the set $Q_i(t)$ contains the pending jobs at time t dispatched on machine i, while each job $j\in\mathcal{J}$ appears by definition in $U_i(t)$ for $\sum_{k\in D_j} q_{ik}(r_{j_k})$ time after its completion or rejection.

Therefore, the proposed assignment for the dual variables leads to the following value of the dual objective

$$\begin{aligned}&\sum_{j\in\mathcal{J}}\lambda_j - \frac{1}{1+\epsilon_s}\sum_{i\in\mathcal{M}}\int_0^\infty\gamma_i(t)dt\\ &\geq \frac{\epsilon_r\epsilon_s}{(1+\epsilon_r)(1+\epsilon_s)}\left(\sum_{j\in\mathcal{J}} w_jF_j^{\mathcal{A}} + \sum_{j\in\mathcal{J}}\left(w_j\sum_{k\in D_j} q_{ik}(r_{j_k})\right)\right)\\ &\geq \frac{\epsilon_r\epsilon_s}{(1+\epsilon_r)(1+\epsilon_s)}\sum_{j\in\mathcal{J}} w_jF_j^{\mathcal{A}} \geq \frac{\epsilon_r\epsilon_s}{(1+\epsilon_r)(1+\epsilon_s)}\sum_{j\in\mathcal{J}\setminus\mathcal{R}} w_jF_j^{\mathcal{A}}\end{aligned}$$

Since the objective value of our linear program is at most twice the value of an optimal non-preemptive schedule, the theorem follows. ◀

4 ℓ_k-norm on Unrelated Machines

In this section, we study the objective of minimizing the weighted ℓ_k-norm of flow-times. Let $\epsilon_s > 0$ and $0 < \epsilon_r < 1$ be the speed augmentation and the rejection fraction of our algorithm, respectively. For each machine $i\in\mathcal{M}$, job $j\in\mathcal{J}$ and time $t\geq r_j$, we introduce a binary variable $x_{ij}(t)$ which indicates if j is processed on i at time t. We consider the following linear programming formulation. Note that the optimal objective value of this linear program is at most $\frac{4(20k)^{k+3}}{\epsilon_s^{k+1}}$ times the total weighted k-th power of flow-time of jobs in an optimal preemptive schedule.

$$\min \sum_{i\in\mathcal{M}}\sum_{j\in\mathcal{J}}\int_{r_j}^{\infty} \frac{2(20k)^{k+3}}{\epsilon_s^{k+1}}\delta_{ij}\left[(t-r_j)^k + p_{ij}^k\right]x_{ij}(t)dt$$
$$\sum_{i\in\mathcal{M}}\int_{r_j}^{\infty}\frac{x_{ij}(t)}{p_{ij}}dt \geq 1 \quad \forall j\in\mathcal{J}$$
$$\sum_{j\in\mathcal{J}} x_{ij}(t) \leq 1 \quad \forall i\in\mathcal{M}, t$$
$$x_{ij}(t)\in\{0,1\} \quad \forall i\in\mathcal{M}, j\in\mathcal{J}, t\geq r_j$$

After relaxing the integrality constraints, we get the following dual program.

$$\max \sum_{j\in\mathcal{J}}\lambda_j - \sum_{i\in\mathcal{M}}\int_0^{\infty}\gamma_i(t)dt$$
$$\frac{\lambda_j}{p_{ij}} - \gamma_i(t) \leq \frac{2(20k)^{k+3}}{\epsilon_s^{k+1}}\delta_{ij}\left[(t-r_j)^k + p_{ij}^k\right] \quad \forall i\in\mathcal{M}, j\in\mathcal{J}, t\geq r_j$$
$$\lambda_j, \gamma_i(t) \geq 0 \quad \forall i\in\mathcal{M}, j\in\mathcal{J}, t$$

The algorithm follows the same ideas as the one in the previous section for the objective of minimizing the total weighted flow-time. Each job is immediately dispatched to a machine upon its arrival. Recall that $Q_i(t)$ is the set of pending jobs at time t dispatched to machine $i\in\mathcal{M}$, i.e., the set of jobs dispatched to i that have been released but not yet completed and have not been rejected at t. Our *scheduling policy* for each machine $i\in\mathcal{M}$ is the same as the previous one: at each time t when the machine i becomes idle or has just completed or interrupted some job, we start executing on i the job $j\in Q_i(t)$ of largest density, i.e., $j = \text{argmax}_{j'\in Q_i(t)}\{\delta_{ij'}\}$. In case of ties, we select the job that arrived the earliest.

When a machine $i\in\mathcal{M}$ starts executing a job $u\in\mathcal{J}$, a counter v_u associated to job u is initialized to zero. Each time when a job $j\in\mathcal{J}$ with $\delta_{ij}>\delta_{iu}$ is released during the execution of u and j is dispatched to i, we increase v_u by w_j. Then, the *rejection policy* is the following: we interrupt the execution of the job k and we reject it the first time where $v_u > \frac{w_u}{\epsilon_r}$. As before we define the set of rejected jobs D_j which causes a decrease to the flow time of j and we say that j is *definitively finished* $\sum_{u\in D_j} q_{iu}(r_{j_u})$ time after its completion or rejection. However, j does not appear to the set of pending jobs $Q_i(t)$ for any t after its completion or rejection. Recall that $U_i(t)$ is the set of jobs that have been marked finished at a time before t in machine i but they have not yet been definitively finished at t. For a job $j\in Q_i(t)\cup U_i(t)$, let $F_j(t)$ be the *remaining time* of j from t to to the moment it is definitively finished.

Let Δ_{ij} be the increase in the total weighted k-th power of flow-time occurred in the schedule of our algorithm if we assign a new job $j\in\mathcal{J}$ to machine i, following the above scheduling and rejection policies. Assuming that the job $u\in\mathcal{J}$ is executed on i at time r_j, we have that, if $v_u + w_j \leq \frac{w_u}{\epsilon_r}$ then

$$\Delta_{ij} = w_j\Bigg(q_{iu}(r_j) + \sum_{\substack{a\in Q_i(r_j)\cup\{j\}\setminus\{u\}:\\ \delta_{ia}\geq\delta_{ij}}} p_{ia}\Bigg)^k + \sum_{\substack{a\in Q_i(r_j)\setminus\{u\}:\\ \delta_{ia}<\delta_{ij}}} w_a\left[\big(F_a(r_j)+p_{ij}\big)^k - F_a(r_j)^k\right],$$

otherwise,

$$\Delta_{ij} = w_j\Bigg(\sum_{\substack{a\in Q_i(r_j)\cup\{j\}:\\ \delta_{ia}\geq\delta_{ij}}} p_{ia}\Bigg)^k + \sum_{\substack{a\in Q_i(r_j)\setminus\{u\}:\\ \delta_{ia}<\delta_{ij}}} w_a\left[\big(F_a(r_j)+p_{ij}-q_{iu}(r_j)\big)^k - F_a(r_j)^k\right],$$

where, in both cases, the first term corresponds to the weighted k-th power of the flow-time of job j if it is dispatched to i and the second term corresponds to the change of the weighted k-th power of flow-time for the jobs in $Q_i(r_j)$. Note that, the second case corresponds to the rejection of u and hence we do not have the term $q_{iu}(r_j)$ in the weighted flow-time of j, while the flow-time of each pending job is reduced by $q_{iu}(r_j)$.

Based on the above, we define λ_{ij} as follows. If $\delta_{ij} > \delta_{iu}$ then λ_{ij} equals to

$$\frac{2^k(10k)^k}{\epsilon_s^k}\frac{1+\epsilon_r}{\epsilon_r}w_j p_{ij}^k + \left(1+\frac{\epsilon_s}{5}\right)w_j\Bigg(\sum_{\substack{a\in Q_i(r_j)\cup\{j\}\setminus\{u\}:\\ \delta_{ia}\geq\delta_{ij}}} p_{ia}\Bigg)^k + \sum_{\substack{a\in Q_i(r_j)\setminus\{u\}:\\ \delta_{ia}<\delta_{ij}}} w_a\left[\big(F_a(r_j)+p_{ij}\big)^k - F_a(r_j)^k\right],$$

otherwise, λ_{ij} equals to

$$\frac{2^k(10k)^k}{\epsilon_s^k}\frac{1+\epsilon_r}{\epsilon_r}w_j p_{ij}^k + \left(1+\frac{\epsilon_s}{5}\right)w_j\Bigg(q_{iu}(r_j) + \sum_{\substack{a\in Q_i(r_j)\cup\{j\}\setminus\{u\}:\\ \delta_{ia}\geq\delta_{ij}}} p_{ia}\Bigg)^k + \sum_{\substack{a\in Q_i(r_j)\setminus\{u\}:\\ \delta_{ia}<\delta_{ij}}} w_a\left[\big(F_a(r_j)+p_{ij}\big)^k - F_a(r_j)^k\right].$$

Intuitively, the value of λ_{ij}'s captures the marginal increase of the total weighted k-th power of flow-times due to the arrival of job j and additionally a prediction term. Note that we do not consider the negative quantity $q_{iu(r_j)}$ that appears in the second case of Δ_{ij}.

The *dispatching policy* of the algorithm is the following: dispatch j to the machine $i^* = \operatorname{argmin}_{i\in\mathcal{M}}\{\lambda_{ij}\}$.

It remains to formally define the dual variables. At the arrival of a job $j \in \mathcal{J}$, we set $\lambda_j = \frac{\epsilon_r}{1+\epsilon_r}\min_{i\in\mathcal{M}}\{\lambda_{ij}\}$ and we will never change the value of λ_j again. Define $\gamma_i(t)$ as

$$\gamma_i(t) = \frac{\epsilon_r}{1+\epsilon_r}\left(1+\frac{\epsilon_s}{2}\right)\left(1+\frac{\epsilon_s}{5}\right)\cdot k \sum_{a\in Q_i(t)\cup U_i(t)} w_a F_a(t)^{k-1}$$

Note that $\gamma_i(t)$ is updated during the execution of $\mathcal{A}$. More specifically, given any fixed time t, $\gamma_i(t)$ may increase if a new job j' arrives at any time $r_{j'} \in [r_j, t)$. However, $\gamma_i(t)$ does never decrease in the case of rejection since the jobs remain in $U_i(t)$ for a sufficient time after their completion or rejection.

Using the above definition of the dual variables, the following theorem holds by a quite more technical analysis than that of the previous section for the total weighted flow-time objective.

▶ **Theorem 6.** *Given any $\epsilon_s > 0$ and $\epsilon_r \in (0,1)$, there is a $(1+\epsilon_s)$-speed $O\left(\frac{k^{(k+3)/k}}{\epsilon_r^{1/k}\epsilon_s^{(k+2)/k}}\right)$-competitive algorithm that rejects jobs of total weight at most $\epsilon_r\sum_{j\in\mathcal{J}} w_j$.*

5 Conclusion

In this paper, we presented a generalized model of resource augmentation through the lens of the duality in mathematical programming. The model unifies previous ones and opens up possibilities for different types of resource augmentation. As shown in the paper, the generalized model can be used to explain the competitiveness of algorithms for certain problems that currently admit no algorithm with performance guarantee even in the resource augmentation context. Besides, an advantage in studying problems in the generalized model is that one can benefit the power of duality-based techniques which have been widely developed for the analysis of approximation and online algorithms. In this context, it would be interesting to consider different problems under the new model. Another interesting question is whether rejection is more powerful than speed. Or, more specifically, can we eliminate the speed augmentation or replace it by a new rejection rule in the presented results? Note that, the power of speed augmentation is that it affects proportionally all jobs, while the difficulty in the rejection case consists in deciding which jobs to reject and how to charge parts of the objective of the non-rejected jobs to the rejected ones.

References

1 Susanne Albers, Lene M. Favrholdt, and Oliver Giel. On paging with locality of reference. *J. Comput. Syst. Sci.*, 70(2):145–175, 2005.

2 S. Anand, Naveen Garg, and Amit Kumar. Resource augmentation for weighted flow-time explained by dual fitting. In *Symposium on Discrete Algorithms*, pages 1228–1241, 2012.

3 Spyros Angelopoulos, Reza Dorrigiv, and Alejandro López-Ortiz. On the separation and equivalence of paging strategies. In *Proc. Symposium on Discrete Algorithms*, pages 229–237, 2007.

4 Nikhil Bansal, Ho-Leung Chan, Rohit Khandekar, Kirk Pruhs, B Schicber, and Cliff Stein. Non-preemptive min-sum scheduling with resource augmentation. In *Proc. 48th Symposium on Foundations of Computer Science*, pages 614–624, 2007.

5 Allan Borodin, Sandy Irani, Prabhakar Raghavan, and Baruch Schieber. Competitive paging with locality of reference. *J. Comput. Syst. Sci.*, 50(2):244–258, 1995.

6 Niv Buchbinder and Joseph Naor. The design of competitive online algorithms via a primal-dual approach. *Foundations and Trends in Theoretical Computer Science*, 3(2-3):93–263, 2009.

7 Chandra Chekuri, Sanjeev Khanna, and An Zhu. Algorithms for minimizing weighted flow time. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 84–93, 2001.

8 Anamitra Roy Choudhury, Syamantak Das, Naveen Garg, and Amit Kumar. Rejecting jobs to minimize load and maximum flow-time. In *Proc. Symposium on Discrete Algorithms*, pages 1114–1133, 2015.

9 Anamitra Roy Choudhury, Syamantak Das, and Amit Kumar. Minimizing weighted ℓ_p-norm of flow-time in the rejection model. In *Proc. 35th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015)*, volume 45, pages 25–37, 2015.

10 Peter J. Denning. The working set model for program behavior. *Commun. ACM*, 11(5):323–333, 1968.

11 Nikhil R. Devanur and Zhiyi Huang. Primal dual gives almost optimal energy efficient online algorithms. In *Proc. 25th ACM-SIAM Symposium on Discrete Algorithms*, 2014.

12 Stefan Dobrev, Rastislav Kralovic, and Dana Pardubská. How much information about the future is needed? In *Proc. 34th Conference on Current Trends in Theory and Practice of Computer Science*, pages 247–258, 2008.

13 Yuval Emek, Pierre Fraigniaud, Amos Korman, and Adi Rosén. Online computation with advice. *Theor. Comput. Sci.*, 412(24):2642–2656, 2011.

14 Leah Epstein and Rob van Stee. Optimal on-line flow time with resource augmentation. *Discrete Applied Mathematics*, 154(4):611–621, 2006.

15 Anupam Gupta, Ravishankar Krishnaswamy, and Kirk Pruhs. Online primal-dual for non-linear optimization with applications to speed scaling. In *Proc. 10th Workshop on Approximation and Online Algorithms*, pages 173–186, 2012.

16 Sungjin Im, Janardhan Kulkarni, and Kamesh Munagala. Competitive algorithms from competitive equilibria: Non-clairvoyant scheduling under polyhedral constraints. In *STOC*, 2014.

17 Sungjin Im, Janardhan Kulkarni, and Kamesh Munagala. Competitive flow time algorithms for polyhedral scheduling. In *Proc. 56th Symposium on Foundations of Computer Science*, pages 506–524, 2015.

18 Sungjin Im, Janardhan Kulkarni, Kamesh Munagala, and Kirk Pruhs. Selfishmigrate: A scalable algorithm for non-clairvoyantly scheduling heterogeneous processors. In *Proc. 55th Symposium on Foundations of Computer Science*, 2014.

19 Sungjin Im, Shi Li, Benjamin Moseley, and Eric Torng. A dynamic programming framework for non-preemptive scheduling problems on multiple machines [extended abstract]. In *Proc. 26th ACM-SIAM Symposium on Discrete Algorithms*, pages 1070–1086, 2015.

20 Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, 2000.

21 Hans Kellerer, Thomas Tautenhahn, and Gerhard J. Woeginger. Approximability and nonapproximability results for minimizing total flow time on a single machine. *SIAM J. Comput.*, 28(4):1155–1166, 1999.

22 Elias Koutsoupias and Christos H. Papadimitriou. Beyond competitive analysis. *SIAM J. Comput.*, 30(1):300–317, 2000.

23 Benjamin Moseley, Kirk Pruhs, and Cliff Stein. The complexity of scheduling for p-norms of flow and stretch - (extended abstract). In *Proc. Integer Programming and Combinatorial Optimization*, pages 278–289, 2013.

24 Cynthia A Phillips, Clifford Stein, Eric Torng, and Joel Wein. Optimal time-critical scheduling via resource augmentation. *Algorithmica*, 32(2):163–200, 2002.

25 Prabhakar Raghavan. A statistical adversary for on-line algorithms. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 7:79–83, 1992.

26 Nguyen Kim Thang. Lagrangian duality in online scheduling with resource augmentation and speed scaling. In *Proc. 21st European Symposium on Algorithms*, pages 755–766, 2013.

27 David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge University Press, 2011.

Admissible Colourings of 3-Manifold Triangulations for Turaev-Viro Type Invariants*

Clément Maria[1] and Jonathan Spreer[2]

1 The University of Queensland, Brisbane, Australia
c.maria@uq.edu.au
2 The University of Queensland, Brisbane, Australia
j.spreer@uq.edu.au

Abstract

Turaev-Viro invariants are amongst the most powerful tools to distinguish 3-manifolds. They are invaluable for mathematical software, but current algorithms to compute them rely on the enumeration of an extremely large set of combinatorial data defined on the triangulation, regardless of the underlying topology of the manifold.

In the article, we propose a finer study of these combinatorial data, called *admissible colourings*, in relation with the cohomology of the manifold. We prove that the set of admissible colourings to be considered is substantially smaller than previously known, by furnishing new upper bounds on its size that are aware of the topology of the manifold. Moreover, we deduce new topology-sensitive enumeration algorithms based on these bounds.

The paper provides a theoretical analysis, as well as a detailed experimental study of the approach. We give strong experimental evidence on large manifold censuses that our upper bounds are tighter than the previously known ones, and that our algorithms outperform significantly state of the art implementations to compute Turaev-Viro invariants.

1998 ACM Subject Classification G.2.1 [Combinatorics] Combinatorial algorithms, F.2.2 [Non-numerical Algorithms and Problems] Computations on discrete structures

Keywords and phrases low-dimensional topology, triangulations of 3-manifolds, cohomology theory, Turaev-Viro invariants, combinatorial algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.64

1 Introduction

In geometric topology, testing if two manifolds are equivalent is one of the most fundamental algorithmic problems. In fact, the task of comparing the topology of two given manifolds often stands at the very beginning of a question, and solving it is essential for conducting research in the field. In the active field of research of 3-manifold topology, this task is remarkably difficult. As a result, practitioners in computational topology rely on simpler invariants – properties of a topological space that can tell different spaces apart.

In the discrete setting, among the most useful invariants for 3-manifolds are the *Turaev-Viro invariants* [16]. They derive from quantum field theory but can be computed by purely combinatorial means – much like the famous Jones polynomial for knots. They are implemented in the major software packages *Regina* [4] and the *Manifold Recogniser* [12, 13],

* This work is supported by the Australian Research Council (DP140104246) and the Department of Industry and Science, Australia under the Australia-India Strategic Research Fund (AISRF06660).

and they play a key role in developing census databases, which are analogous to the well-known dictionaries of knots [1, 12]. Their main difficulty is that they are slow to compute: the best implementations rely on the enumeration of exponentially large sets of combinatorial data defined on a triangulation.

The Turaev-Viro invariants are a family of invariants (TV_r) indexed by an integer $r \geq 3$. For a triangulation $\mathfrak{T}$, the Turaev-Viro invariant is based on *colourings* of the triangulation $\mathfrak{T}$, which are assignements of one of $r-1$ distinct colours to each of the edges of $\mathfrak{T}$. Only a subset of colourings satisfy *admissibility constraints* – which are of combinatorial nature –, and each *admissible colouring* defines a *weight*. The Turaev-Viro invariant $\mathrm{TV}_r(\mathfrak{T})$ is equal to the sum of these weights over all admissible colourings.

For any $r \geq 3$, a naive algorithm to compute $\mathrm{TV}_r(\mathfrak{T})$ on a triangulation $\mathfrak{T}$, with m edges, consists of a simple backtracking procedure enumerating all of the $(r-1)^m$ edge colourings, checking each of them for admissibility and summing the weights, resulting in a memory efficient but very slow implementation. More recently, Burton and the authors introduced a *fixed parameter tractable (FPT)* algorithm which is *linear* in the size of the input, and only singly exponential in the treewidth of the dual graph of $\mathfrak{T}$ [5]. This is possible by using the structure of the input to process large groups of admissible colourings simultaneously. Despite good performance in practice, this approach requires exponential memory and the running time is very sensitive to the combinatorial structure of the input triangulation, as opposed to the topology of the underlying manifold.

Algorithmic results exist for specific values of r. For $r = 3$, the Turaev-Viro invariant $\mathrm{TV}_3(\mathfrak{T})$ can be interpreted in terms of the cohomology of the manifold, which results in a polynomial time algorithm [5, 12]. For $r = 4$ however, the computation of the invariant is known to be hard for the *counting complexity class* #P [5, 11]. This gives evidence that a general efficient solution (for example polynomial) for computing TV_r is unlikely to exist.

In this article, we elaborate on the cohomology interpretation of the Turaev-Viro invariants, successful for the case $r = 3$, to design more efficient implementations for TV_r relying on an optimised enumeration of admissible colourings. More precisely, we use the admissibility constraints to connect colourings and cohomology classes of the manifold, and reduce a priori the number of colourings to be considered algorithmically in order to find all admissible colourings.

Using this technique, we study the structure of the set of admissible colourings for $r = 3$ and $r = 4$. We design new sharper upper bounds on the number of admissible colourings of a triangulation for $r = 4$, and deduce an algorithm to compute TV_4 which is linear in these new bounds. This is of particular interest considering the #P-hardness of this computation. We give experimental evidence on large censuses of triangulations that these upper bounds are sharp in many cases and significantly better than the naive ones.

We then study in more details admissible colourings that reduce to the trivial cohomology class. This is a special case of particular importance, as it allows the study of homology spheres – manifolds involved in the 3-sphere recognition problem – and later becomes a key ingredient for an improved algorithm to compute TV_r, with r odd, on any manifold. We deduce new sharp upper bounds on the number of colourings of homology spheres for $r \leq 7$.

Finally, building on this study at the trivial cohomology class, and work by Kirby and Melvin [10] and Matveev [12], we introduce an improved algorithm to compute the Turaev-Viro invariants for odd values of r. By embedding it within existing algorithms, our method allows a significant exponential speed-up on both backtracking algorithm and FPT algorithm to compute Turaev-Viro invariants. We provide large scale experiments to show the interest of the method. In particular, our new enumeration of colourings, combined with the FPT algorithm to compute Turaev-Viro, performs up to two orders of magnitude faster than state of the art implementation, hence opening notably the range of possible practical computations in 3-manifold topology.

These implementations will appear as features in the 3-manifold software `Regina` [4].

2 Background

Manifolds and generalised triangulations: Let M be a closed 3-manifold. A *generalised triangulation* $\mathfrak{T}$ of M is a collection of n abstract tetrahedra $\Delta_1, \ldots, \Delta_n$ together with $2n$ *gluing maps* identifying their $4n$ triangular faces in pairs, such that the underlying topological space is homeomorphic to M.

As a consequence of the gluings, vertices, edges or triangles of the same tetrahedron may be identified. It follows from an Euler characteristic argument, that any n-tetrahedra v-vertex triangulation of a closed 3-manifold must have $2n$ triangles and $n+v$ edges. It is common in practical applications to have a *one-vertex triangulation*, in which all vertices of all tetrahedra are identified to a single point. We refer to an equivalence class defined by the gluing maps as a single *face of the triangulation*. The number of tetrahedra n of $\mathfrak{T}$ is often referred to as the *size* of the triangulation. We denote by V, E, F and T the vertices, edges, triangles and tetrahedra, respectively, of a generalised triangulation.

Generalised triangulations are widely used in 3-manifold topology. They are more general than simplicial complexes, and can encode a wide range of manifolds, and very complex topologies, with very few tetrahedra. For instance, one can build 13 400 *distinct* prime manifolds with less than 11 tetrahedra [12], and the number of distinct manifolds represented by generalised triangulations with less than n tetrahedra grows super-exponentially with n.

We refer to [9] for more details on generalised triangulations.

Homology and cohomology: In the following section we give a very brief introduction to (co)homology theory. For more details see [7].

Let $\mathfrak{T}$ be a generalised 3-manifold triangulation. For the field of coefficients $\mathbb{Z}_2 := \mathbb{Z}/2\mathbb{Z}$, the *group of p-chains*, $0 \le p \le 3$, denoted $\mathbf{C}_p(\mathfrak{T}, \mathbb{Z}_2)$, of $\mathfrak{T}$ is the group of formal sums of p-faces with $\mathbb{Z}_2$ coefficients. The *boundary operator* is a linear operator $\partial_p : \mathbf{C}_p(\mathfrak{T}, \mathbb{Z}_2) \to \mathbf{C}_{p-1}(\mathfrak{T}, \mathbb{Z}_2)$ such that $\partial_p \sigma = \partial_p \{v_0, \cdots, v_p\} = \sum_{j=0}^{p} \{v_0, \cdots, \widehat{v_j}, \cdots, v_p\}$, where σ is a face of $\mathfrak{T}$, $\{v_0, \ldots, v_p\}$ represents σ as a face of a tetrahedron of $\mathfrak{T}$ in local vertices $v_0, \ldots, v_p$, and $\widehat{v_j}$ means v_j is deleted from the list. Denote by $\mathbf{Z}_p(\mathfrak{T}, \mathbb{Z}_2)$ and $\mathbf{B}_{p-1}(\mathfrak{T}, \mathbb{Z}_2)$ the kernel and the image of ∂_p respectively. Observing $\partial_p \circ \partial_{p+1} = 0$, we define the *$p$-th homology group* $\mathbf{H}_p(\mathfrak{T}, \mathbb{Z}_2)$ of $\mathfrak{T}$ by the quotient $\mathbf{H}_p(\mathfrak{T}, \mathbb{Z}_2) = \mathbf{Z}_p(\mathfrak{T}, \mathbb{Z}_2)/\mathbf{B}_p(\mathfrak{T}, \mathbb{Z}_2)$. These structures are vector spaces.

The concept of *cohomology* is in many ways dual to homology, but more abstract and endowed with more algebraic structure. It is defined in the following way: The *group of p-cochains* $\mathbf{C}^p(\mathfrak{T}, \mathbb{Z}_2)$ is the formal sum of linear maps of p-faces of $\mathfrak{T}$ into $\mathbb{Z}_2$. The *coboundary operator* is a linear operator $\delta^p : \mathbf{C}^{p-1}(\mathfrak{T}, \mathbb{Z}_2) \to \mathbf{C}^p(\mathfrak{T}, \mathbb{Z}_2)$ such that for all $\phi \in \mathbf{C}^{p-1}(\mathfrak{T}, \mathbb{Z}_2)$ we have $\delta^p(\phi) = \phi \circ \partial_p$. As above, *$p$-cocycles* are the elements in the kernel of δ^{p+1}, *p-coboundaries* are elements in the image of δ^p, and the *p-th cohomology group* $\mathbf{H}^p(\mathfrak{T}, \mathbb{Z}_2)$ is defined as the p-cocycles factored by the d-coboundaries.

We denote by $\beta_1(\mathfrak{T}, \mathbb{Z}_2)$ the dimension of $\mathbf{H}_1(\mathfrak{T}, \mathbb{Z}_2)$, called the first *Betti number* of the manifold. By duality, this is also the dimension of homology and cohomology groups of dimension $p \in \{1, 2\}$, with $\mathbb{Z}_2$ coefficients.

In Section 3.1 we discuss how 1-cocycles correspond to (sets of) *admissible colourings* of the edges of $\mathfrak{T}$ used in the definition of Turaev-Viro invariants.

Turaev-Viro invariants: In this section we briefly describe *invariants of Turaev-Viro type* TV_r, parameterised by an integer $r \geq 3$. We then have a closer look at the more specialised original *Turaev-Viro invariants* $\mathrm{TV}_{r,q}$, which also depend on a second integer $0 < q < 2r$.

Let $\mathfrak{T}$ be a generalised triangulation of a closed 3-manifold M, and let $r \geq 3$, be an integer. Let V, E, F and T denote the set of vertices, edges, triangles and tetrahedra of the triangulation $\mathfrak{T}$ respectively. Let $I = \{0, 1/2, 1, 3/2, \ldots, (r-2)/2\}$ be the set of the first $r-1$ non-negative half-integers. A *colouring* of $\mathfrak{T}$ is defined to be a map $\theta : E \to I$; that is, θ "colours" each edge of $\mathfrak{T}$ with an element of I. A colouring θ is *admissible* if, for each triangle of $\mathfrak{T}$, the three edges e_1, e_2, and e_3 bounding the triangle satisfy the

- *parity condition* $\theta(e_1) + \theta(e_2) + \theta(e_3) \in \mathbb{Z}$;
- *triangle inequalities* $\theta(e_i) \leq \theta(e_j) + \theta(e_k)$, $\{i,j,k\} = \{1,2,3\}$; and
- *upper bound constraint* $\theta(e_1) + \theta(e_2) + \theta(e_3) \leq r-2$.

For a triangulation $\mathfrak{T}$ and $r \geq 3$, its set of admissible colourings is denoted by $\mathrm{Adm}(\mathfrak{T}, r)$.

For each admissible colouring θ and for each vertex $w \in V$, edge $e \in E$, triangle $f \in F$ or tetrahedron $t \in T$ we define *weights* $|w|_\theta, |e|_\theta, |f|_\theta, |t|_\theta \in \mathbb{C}$. The weights of vertices are constant, and the weights of edges, triangles and tetrahedra only depend on the colours of edges they are incident to. Using these weights, we define the *weight of the colouring* to be

$$|\mathfrak{T}|_\theta = \prod_{w \in V} |w|_\theta \times \prod_{e \in E} |e|_\theta \times \prod_{f \in F} |f|_\theta \times \prod_{t \in T} |t|_\theta, \tag{1}$$

Invariants of Turaev-Viro types of $\mathfrak{T}$ are defined as sums of the weights of all admissible colourings of $\mathfrak{T}$, that is $\mathrm{TV}_r(\mathfrak{T}) = \sum_{\theta \in \mathrm{Adm}(\mathfrak{T},r)} |\mathfrak{T}|_\theta$.

In [16], Turaev and Viro show that, when the weighting system satisfies some identities, $\mathrm{TV}_r(\mathfrak{T})$ is indeed an invariant of the manifold; that is, if $\mathfrak{T}$ and $\mathfrak{T}'$ are generalised triangulations of the same closed 3-manifold M, then $\mathrm{TV}_r(\mathfrak{T}) = \mathrm{TV}_r(\mathfrak{T}')$ for all r. We thus sometimes abuse notation and write $\mathrm{TV}_r(M)$, meaning the Turaev-Viro type invariant computed for a triangulation of M.

We refer to [5] for a precise definition of the weights of the original Turaev-Viro invariant at $sl_2(\mathbb{C})$, which not only depend on r but also on a second integer $0 < q < 2r$. The exact definition of these weights is rather involved, but not at all important in order to understand the findings presented in this article, we thus continue to denote these weights by $|\cdot|_\theta$ despite the fact that they not only depend on θ, but also on r and q. We use these weights in our experiments in Section 4.

For an n-tetrahedra triangulation $\mathfrak{T}$ with v vertices there is a simple backtracking algorithm to compute $\mathrm{TV}_{r,q}(\mathfrak{T})$ by testing the $(r-1)^{v+n}$ possible colourings for admissibility and computing their weights. The case $r = 3$ can however be computed in polynomial time, due to a connection between $\mathrm{Adm}(\mathfrak{T}, 3)$ and cohomology, see Section 3.1 and [5, 12].

Classical results about Turaev-Viro invariants: Note that the Turaev-Viro invariants $\mathrm{TV}_{r,q}$ are closely related to the more general invariant of Witten and Reshetikhin-Turaev $\tau_{r,q}$ ($\in \mathbb{C}$), due to the following result.

▶ **Theorem 1** (Turaev [15], Roberts [14]). *For the invariants of Witten and Reshetikhin-Turaev $\tau_{r,q}$, and the Turaev-Viro invariants, the following equality holds*

$$\mathrm{TV}_{r,q} = | \tau_{r,q} |^2 \ .$$

Theorem 1 enables us to translate a number of key results about the Witten and Reshetikhin-Turaev invariants in terms of Turaev-Viro invariants. Namely, the following statement holds.

▶ **Theorem 2** (Based on Kirby and Melvin [10]). *Let M and N be closed compact 3-manifolds, and let $r \geq 3$, $1 \leq q \leq r-1$. Then there exist $\gamma_r \in \mathbb{C}$, such that for $\mathrm{TV}'_{r,1} = \gamma_r \mathrm{TV}_{r,1}$ we have*

$$\mathrm{TV}'_{r,1}(M \# N) = \mathrm{TV}'_{r,1}(M) \cdot \mathrm{TV}'_{r,1}(N).$$

Additionally, when a manifold M is represented by a triangulation with n tetrahedra, the normalising factor γ_r can be computed in polynomial time in n.

Using Turaev-Viro invariants at the trivial cohomology class we have the following identity for odd degree r.

▶ **Theorem 3** (Based on Kirby and Melvin [10]). *Let M be a closed compact 3-manifold, and let $r \geq 3$ be an odd integer. Then*

$$\mathrm{TV}_{r,1}(M) = \mathrm{TV}_{3,1}(M) \cdot \mathrm{TV}_{r,1}(M, [0]).$$

3 Reduction of colourings at cohomology classes

Let $\mathfrak{T}$ be a 3-manifold triangulation with v vertices, $n+v$ edges, $2n$ triangles and n tetrahedra. Following the definitions in Section 2 above, there are at most $(r-1)^{n+v}$ admissible colourings for $\mathrm{Adm}(\mathfrak{T}, r)$. Due to the admissibility constraints for colourings, as described in Section 2, this bound is usually far from being sharp. However, current enumeration algorithms for admissible colourings do not try to capitalise on this fact (including the parameterised algorithm from [5]).

In this section we discuss methods that incorporate these constraints in a controlled fashion when enumerating admissible colourings. More precisely, we present improved upper bounds on the number of admissible colourings in important special cases (thus reducing a priori the number of options an enumeration algorithm needs to consider). Moreover, we give a number of examples where these new upper bounds are actually attained. The bounds are then used to construct a structure sensitive algorithm to enumerate $\mathrm{Adm}(\mathfrak{T}, 4)$, and to achieve a significant exponential speed-up for the computation of the Turaev-Viro invariants $\mathrm{TV}_{r,1}$ where r is odd.

3.1 Turaev-Viro invariants for r=3 and cohomology

There is a close connection between the first cohomology group of a 3-manifold triangulation $\mathfrak{T}$ and the admissible colourings of the Turaev-Viro invariants for $r = 3$. We discuss this connection under the viewpoint of triangulations which helps setting the scene for improved bounds on the number of admissible colourings for higher values of r, as presented in Sections 3.2 and 3.4 below.

▶ **Proposition 4.** *Let $\mathfrak{T}$ be a 3-manifold triangulation with v vertices. Then there is a bijection between $\mathrm{Adm}(\mathfrak{T}, 3)$ and the 1-cocycles of $\mathfrak{T}$, and we have $|\mathrm{Adm}(\mathfrak{T}, 3)| = 2^{v+\beta_1(\mathfrak{T},\mathbb{Z}_2)-1}$.*

Proof. An edge colouring $\theta : E \to \{0, 1/2\}$ defines a 1-cochain α_θ with coefficients in $\mathbb{Z}_2$ evaluating to 1 on edges coloured 1/2 and to 0 otherwise. The parity condition on θ is then equivalent to the boundary of α_θ (which is a 2-chain) vanishing over $\mathbb{Z}_2$. Moreover, note that every colouring $\theta : E \to \{0, 1/2\}$ satisfying the parity condition is admissible for $r = 3$. Thus θ is admissible if and only if α_θ is a cocycle. This proves the first statement. The second statement follows from the observation that $\mathfrak{T}$ has exactly $2^{v+\beta_1(\mathfrak{T},\mathbb{Z}_2)-1}$ cocycles. ◀

Let $\mathbf{H}^1(\mathfrak{T},\mathbb{Z}_2) = (\mathbb{Z}_2)^{\beta_1(\mathfrak{T},\mathbb{Z}_2)}$ be the first cohomology group of $\mathfrak{T}$. Since every 1-cocycle α in $\mathfrak{T}$ is a representative of a cohomology class $[\alpha] \in \mathbf{H}^1(\mathfrak{T},\mathbb{Z}_2)$, every admissible colouring in $\mathrm{Adm}(\mathfrak{T},3)$ can be associated to a cohomology class. This correspondence can be generalised to arbitrary $r \geq 3$ with the help of the following observation.

▶ **Proposition 5.** *Let $\mathfrak{T}$ be a 3-manifold triangulation with edge set E, $r \geq 3$, and $\theta \in \mathrm{Adm}(\mathfrak{T},r)$. Then the* reduction of θ*, defined by $\theta' : E \to \{0, 1/2\}$; $e \mapsto \theta(e) - \lfloor \theta(e) \rfloor$, is an admissible colouring in $\mathrm{Adm}(\mathfrak{T},3)$.*

Proof. Let f be a triangle of $\mathfrak{T}$ with edges e_1, e_2, and e_3. Since $\theta \in \mathrm{Adm}(\mathfrak{T},r)$ is admissible, we have $\theta(e_1)+\theta(e_2)+\theta(e_3) \in \mathbb{Z}$. Thus, there are either no or two half-integers amongst the colours of the edges of f and $\theta' \in \mathrm{Adm}(\mathfrak{T},3)$. ◀

We have seen that every colouring $\theta \in \mathrm{Adm}(\mathfrak{T},r)$ can be associated to a 1-cohomology class of $\mathfrak{T}$ via its reduction $\theta' \in \mathrm{Adm}(\mathfrak{T},3)$ and Proposition 4. We know from [12, 16] that this construction can be used to split $\mathrm{TV}_r(\mathfrak{T})$ (and thus also $\mathrm{TV}_{r,q}(\mathfrak{T})$) into simpler invariants indexed by the elements of $\mathbf{H}^1(\mathfrak{T},\mathbb{Z}_2)$. More precisely, let $[\alpha] \in \mathbf{H}^1(\mathfrak{T},\mathbb{Z}_2)$ be a cohomology class, then

$$\mathrm{TV}_r(\mathfrak{T},[\alpha]) = \sum_{\substack{\theta \in \mathrm{Adm}(\mathfrak{T},r) \\ \theta \bmod 2 \in [\alpha]}} |\mathfrak{T}|_\theta,$$

where θ mod 2 denotes the reduction of θ, is an invariant of $\mathfrak{T}$. The special case $\mathrm{TV}_r(\mathfrak{T},[0])$ is of particular importance for computations as explained in further detail in Section 3.4.

3.2 Admissible colourings for r=4

We have seen in Proposition 4 that admissible colourings for $r = 3$ are in one-to-one correspondence to the 1-cocycles of a triangulated 3-manifold $\mathfrak{T}$. This basic but very useful observation has consequences for the structure of $\mathrm{Adm}(\mathfrak{T},4)$. This is particularly interesting as computing $\mathrm{TV}_{4,1}$ is known to be $\#P$-hard [5, 11]. More precisely, the following statement holds.

▶ **Theorem 6.** *Let $\mathfrak{T}$ be an n-tetrahedron 3-manifold triangulation with v vertices, and let $\theta \in \mathrm{Adm}(\mathfrak{T},3)$. Furthermore, let $\ker_\theta$ be the number of edges coloured 0 by θ. Then*

$$|\mathrm{Adm}(\mathfrak{T},4)| \leq \left(\Sigma_{\theta\in\mathrm{Adm}(\mathfrak{T},3)\setminus\{\mathbf{0}\}} 2^{\ker_\theta}\right) + 2^{v+\beta_1(\mathfrak{T},\mathbb{Z}_2)-1} \quad (2)$$

$$\leq (|\mathrm{Adm}(\mathfrak{T},3)|-1)(2^{n+v-1}+1)+1, \quad (3)$$

where **0** *denotes the all zero colouring. Moreover, both bounds are sharp.*

Proof. Let $\theta \in \mathrm{Adm}(\mathfrak{T},4)$, and let θ' be its reduction, as defined in Proposition 5. If θ' is the trivial colouring (that is, if no colour of θ is coloured by $1/2$) the colouring $\theta/2$, obtained by dividing all of the colours of θ by two, must be in $\mathrm{Adm}(\mathfrak{T},3)$. It follows from Proposition 4 that exactly $2^{v+\beta_1(\mathfrak{T},\mathbb{Z}_2)-1}$ colourings in $\mathrm{Adm}(\mathfrak{T},4)$ reduce to the trivial colouring.

If θ' is not the trivial colouring then θ colours some edges by $1/2$. In particular it is not the trivial colouring. Since the only colours in θ are 0, $1/2$, and 1, all edges coloured by $1/2$

in θ are coloured by $1/2$ in θ' and vice versa. Thus, $\ker_{\theta'}$ denotes all edges coloured by 0 or 1 in θ. Naturally, there are at most $2^{\ker_{\theta'}}$ such colourings. The result now follows by adding these upper bounds $2^{\ker_{\theta'}}$ over all non-trivial reductions $\theta' \in \mathrm{Adm}(\mathfrak{T}, 3)$, and adding the $2^{v+\beta_1(\mathfrak{T},\mathbb{Z}_2)-1}$ extra colourings with trivial reduction.

For Equation (3) note that each non-trivial colouring in $\mathrm{Adm}(\mathfrak{T}, 3)$ has at least one edge coloured $1/2$ and thus $\ker_{\theta'}$ is at most the number of edges minus one.

It follows that for $\beta_1(\mathfrak{T}, \mathbb{Z}_2)$ or v sufficiently large this bound cannot be tight. For 1-vertex triangulations $\mathfrak{T}$ with $\beta_1(\mathfrak{T}, \mathbb{Z}_2) = 0$ this bound is sharp as explained in Proposition 7. Looking at all 1-vertex triangulations with $\beta_1(\mathfrak{T}, \mathbb{Z}_2) = 1$ up to six tetrahedra, the cases of equality in Inequality (3) are summarised in Table 2. See Table 1 for a large number of cases of equality for Inequality (2). ◀

3.3 A structure-sensitive algorithm to compute $\mathrm{Adm}(\mathfrak{T}, 4)$

In this section we describe an algorithm to compute $\mathrm{TV}_{4,q}$ – a problem known to be $\#P$-hard – exploiting the combinatorial structure of the input triangulation. The algorithm is a direct consequence of the proof of Theorem 6.

Input. A v-vertex n-tetrahedra triangulation of a closed 3-manifold $\mathfrak{T}$ with set of edges E

1. Compute $\mathrm{Adm}(\mathfrak{T}, 3)$. Following the proof of Proposition 4, it is enough to compute a basis of the 1-cohomology of $\mathfrak{T}$ with coefficients in the field with two elements $\mathbb{Z}_2$. Then every cocycle naturally defines an admissible colouring and vice versa. This can be done in polynomial time by solving a linear system of equations. $\mathrm{Adm}(\mathfrak{T}, 3)$ can then be enumerated using the cohomology basis.
2. For all $\theta \in \mathrm{Adm}(\mathfrak{T}, 3)$, enumerate the set of edges $\ker_\theta \subset E$ of $\mathfrak{T}$ coloured zero in θ.
3. For each non-trivial $\theta \in \mathrm{Adm}(\mathfrak{T}, 3)$, for each subset $A \subseteq \ker_\theta$: Let θ' be the edge colouring that colours (i) all edges in A by 1, (ii) all edges in $(E \setminus \ker_\theta)$ by $1/2$, and (iii) all edges in $(\ker_\theta \setminus A)$ by 0. For each non-trivial θ, set up a backtracking procedure to check all such θ' for admissibility. Add the admissible colourings θ' to $\mathrm{Adm}(\mathfrak{T}, 4)$.
4. For all colourings $\theta \in \mathrm{Adm}(\mathfrak{T}, 3)$, double all colours of θ and add the result to $\mathrm{Adm}(\mathfrak{T}, 4)$.

Correctness of the algorithm and running time. Due to Theorem 6 we know that the above procedure enumerates all colourings in $\mathrm{Adm}(\mathfrak{T}, 4)$. Computing $\mathrm{TV}_{4,q}(\mathfrak{T})$ thus runs in

$$O\left(\left(\Sigma_{\theta\in\mathrm{Adm}(\mathfrak{T},3)\setminus\{\mathbf{0}\ \}}\ 2^{\ker_\theta}\right) \cdot n + 2^{v+\beta_1(\mathfrak{T},\mathbb{Z}_2)-1}\right)$$

arithmetic operations. This upper bound is much smaller than the worst case running time $(r-1)^{n+v}$ of the naive backtracking procedure.

In Section 4.3, we provide experimental evidence on a large census of triangulations that the new upper bounds on the number of admissible colourings from Theorem 6 are tight in many cases and close to being tight in average, and that our new algorithm to enumerate the colourings of $\mathrm{Adm}(\mathfrak{T}, 4)$ experimentally exhibits an output-sensitive nature.

3.4 Computing Turaev-Viro invariants at the zero cohomology class

Following Proposition 4 the complexity of enumerating admissible colourings of a 3-manifold triangulation $\mathfrak{T}$ not only depends on the size n of $\mathfrak{T}$, but also on (i) the number of vertices, and (ii) the first Betti number of $\mathfrak{T}$.

Regarding (i) we show in Section 3.5 that, given $\mathfrak{T}$, we can efficiently find a triangulation $\mathfrak{T}'$ of the same 3-manifold of same or smaller size with only one vertex. Regarding (ii) the first Betti number of $\mathfrak{T}$ is a topological invariant and hence an unchangeable part of the input. Thus, when computing $\mathrm{TV}_{r,q}(\mathfrak{T})$ by enumerating colourings, this layer of complexity can not be avoided. However, this observation does not hold for the invariant $\mathrm{TV}_{r,q}(\mathfrak{T},[0])$ which is a useful tool for various reasons.

1. First of all and most prominently, in order to compute $\mathrm{TV}_{r,q}(\mathfrak{T},[0])$ we only need to consider admissible colourings which correspond to the zero cohomology class. Following Proposition 4 for a one-vertex triangulation $\mathfrak{T}$, the colourings corresponding to the zero cohomology class are precisely the ones which reduce to the all zero colouring and thus can only have integer colours. A similar statement for the case of special spines can be found in [12, Remark 8.1.2.2].
2. One of the most important tasks of 3-manifold invariants is to distinguish between a 3-manifold triangulation $\mathfrak{T}$ and the 3-sphere (this task is known as the 3-*sphere recognition problem*). Whenever the homology groups of $\mathfrak{T}$ and the 3-sphere are different, this distinction can efficiently be made (i.e., in polynomial time). Hence, 3-sphere recognition is most interesting when homology fails, that is, when $\mathfrak{T}$ has the (trivial) homology of the 3-sphere $\mathbf{H}^1(\mathfrak{T},\mathbb{Z}_2)=\{[0]\}$. In this important case we have $\mathrm{TV}_{r,q}(\mathfrak{T})=\mathrm{TV}_{r,q}(\mathfrak{T},[0])$.
3. There are several non-trivial further cases when $\mathrm{TV}_{r,q}(\mathfrak{T})$ can be obtained from $\mathrm{TV}_{r,q}(\mathfrak{T},[0])$ in polynomial time, see Section 3.5 for details.

For the remainder of this section, instead of considering $\mathrm{TV}_{r,q}(\cdot,[0])$, we follow the related approach of considering $\mathrm{TV}_{r,q}(\cdot)$ and triangulations with vanishing first Betti number. We will use this study in the next section to derive a faster algorithm to compute $\mathrm{TV}_{r,q}(\cdot)$ on *all* manifold triangulations for r odd and $q=1$. The following facts follow from the observations made in Sections 3.1 and 3.2.

▶ **Proposition 7.** *Let $\mathfrak{T}$ be a 1-vertex triangulation such that $\beta_1(\mathfrak{T},\mathbb{Z}_2)=0$. Then*
(i) $|\mathrm{Adm}(\mathfrak{T},r)|=1$ *for $r\le 4$;*
(ii) *Let $\theta\in\mathrm{Adm}(\mathfrak{T},r)$, then all colours in θ must be integers;*
(iii) $|\mathrm{Adm}(\mathfrak{T},r)|\le\lfloor\frac{r}{2}\rfloor^{n+1}$.

In particular, $\mathrm{TV}_{r,q}(\mathfrak{T})$, $r\le 4$, must be trivial, and manifolds with trivial $\mathbb{Z}_2$-cohomology (a large group of 3-manifolds) can never be distinguished from the 3-sphere by $\mathrm{TV}_{r,q}$, $r\le 4$.

Proof.
(i) It follows from Proposition 4 that $\mathrm{Adm}(\mathfrak{T},3)=\{\mathbf{0}\}$ and the statement follows from Theorem 6.
(ii) Since $\mathrm{Adm}(\mathfrak{T},3)=\{\mathbf{0}\}$ all colourings must reduce to the all zero colouring.
(iii) Since all colours in θ must be (a) integers, (b) sum to at most $r-2$ on each triangle, and (c) satisfy the triangle inequality. It follows that all colours must be integers between 0 and $\lfloor\frac{r-2}{2}\rfloor$. The statement now follows from the fact that $\mathfrak{T}$ has $n+1$ edges. ◀

The bound from Proposition 7 cannot be sharp since not all triangle colourings $(a,b,c)\in\{0,1,\dots,\lfloor\frac{r-2}{2}\rfloor\}^3$ are admissible. For $5\le r\le 7$ we have the following situation.

▶ **Theorem 8.** *Let $\mathfrak{T}$ be a 1-vertex n-tetrahedron triangulation such that $\beta_1(\mathfrak{T},\mathbb{Z}_2)=0$, then*

$$|\mathrm{Adm}(\mathfrak{T},5)|\le 2^n+1;\qquad |\mathrm{Adm}(\mathfrak{T},6)|\le 3^n+1;\qquad |\mathrm{Adm}(\mathfrak{T},7)|\le 3^n+1.$$

Moreover, all these upper bounds are sharp.

Proof. For $r = 5$ the admissible triangle colourings are $(0, 0, 0)$, $(1/2, 1/2, 0)$, $(1, 1, 0)$, $(1, 1/2, 1/2)$, $(1, 1, 1)$, $(3/2, 3/2, 0)$, $(3/2, 1, 1/2)$, up to permutations. By Proposition 5, no colouring in $\mathrm{Adm}(\mathfrak{T}, 5)$ can contain an edge colour $1/2$ or $3/2$: Otherwise the reduction of such a colouring would be a non-trivial colouring in $\mathrm{Adm}(\mathfrak{T}, 3)$, which does not exist (cf. Proposition 4 and Corollary 7 with $v = 1$ and $\beta_1(\mathfrak{T}, \mathbb{Z}_2) = 0$). Hence, all edge colours must be 0 or 1, leaving triangle colourings $(0, 0, 0)$, $(1, 1, 0)$, and $(1, 1, 1)$.

By an Euler characteristic argument, a 1-vertex n-tetrahedron 3-manifold has $n+1$ edges. Hence the number of colourings of $\mathrm{TV}_{5,q}$ is trivially bounded above by 2^{n+1}. Furthermore, let $\theta \in \mathrm{Adm}(\mathfrak{T}, 5)$, then either θ is constant 0 on the edges, constant 1 on the edges, or θ contains a triangle coloured $(1, 1, 0)$. In the last case, the complementary colouring θ', obtained by flipping the colour on all the edges, contains a triangle coloured $(0, 0, 1)$ and thus $\theta' \notin \mathrm{Adm}(\mathfrak{T}, 5)$. It follows that $|\mathrm{Adm}(\mathfrak{T}, 5)| \leq 2^n + 1$.

For $r = 6$ the admissible triangle colourings are the ones from the case $r = 5$ above plus $(3/2, 3/2, 1)$, $(2, 1, 1)$, $(2, 2, 0)$, $(2, 3/2, 1/2)$. Again, due to Proposition 5, no half-integers can occur in any colouring. Thus, the only admissible triangle colourings are $(0, 0, 0)$, $(1, 1, 0)$, $(1, 1, 1)$, $(2, 1, 1)$, and $(2, 2, 0)$.

We trivially have $|\mathrm{Adm}(\mathfrak{T}, 6)| \leq 3^{n+1}$. Let $\theta \in \mathrm{Adm}(\mathfrak{T}, 6)$. We want to show, that at most a third of all non-constant assignment of colours 0, 1, 2 to the edges of $\mathfrak{T}$ can be admissible. For this, let $\theta \in \mathrm{Adm}(\mathfrak{T}, 6)$ and let θ' be defined by adding 1 (mod 3) to every edge colour. For θ' to be admissible, all triangles of θ must be of type $(0, 0, 0)$ and $(2, 1, 1)$. If at least one triangle has colouring $(0, 0, 0)$, θ must be the trivial colouring. Hence, all triangles are of type $(2, 1, 1)$ in θ. Replacing 2 by 0 and 1 by $1/2$ in θ yields a non-trivial admissible colouring in $\mathrm{Adm}(\mathfrak{T}, 3)$, a contradiction by Corollary 7. Hence, for every non-trivial admissible colouring θ, the colouring θ' cannot be admissible.

Analogously, let θ'' be defined by adding 2 (mod 3) to every edge colour of θ. For θ'' to be admissible, all triangles of θ must be of the type $(1, 1, 1)$, or $(2, 2, 0)$. A single triangle of type $(1, 1, 1)$ in θ forces θ to be constant. Hence, all triangles must be of type $(2, 2, 0)$. Dividing θ by four defines a non-trivial colouring in $\mathrm{Adm}(\mathfrak{T}, 3)$, a contradiction.

Combining these observations, at most every third non-trivial assignment of colours 0, 1, 2 to the edges of θ can be admissible. Adding the two admissible constant colourings yields $|\mathrm{Adm}(\mathfrak{T}, 6)| \leq 3^n + 1$.

The proof for $r = 7$ follows from a slight adjustment of the proof for $r = 6$. Admissible triangle colourings for colourings in $\mathrm{Adm}(\mathfrak{T}, 7)$ are the ones from $r = 6$ plus $(2, 2, 1)$. Again, we want to show that at most every third non-trivial assignment of colours 0, 1, 2 to the edges of $\mathfrak{T}$ can be admissible. For this let $\theta \in \mathrm{Adm}(\mathfrak{T}, 7)$ and let θ' and θ'' be defined as above. For θ' to be admissible θ must consist of triangle colourings of type $(0, 0, 0)$, $(1, 1, 0)$ and $(2, 1, 1)$. Whenever θ is non-constant replacing 2 by 0, and 1 by $1/2$ yields a non-trivial colouring in $\mathrm{Adm}(\mathfrak{T}, 3)$ which is not possible. The argument for θ'' is the same as in the case $r = 6$. It follows that $|\mathrm{Adm}(\mathfrak{T}, 7)| \leq 3^n + 1$.

All of the above bounds are attained by a number of small 3-sphere triangulations. See Table 2 for more details about 1-vertex triangulations $\mathfrak{T}$ with $\beta_1(\mathfrak{T}, \mathbb{Z}_2) = 0$ with up to six tetrahedra and their average number of admissible colourings $|\mathrm{Adm}(\mathfrak{T}, r)|$, $5 \leq r \leq 7$. ◀

There are 27,202 1-vertex triangulations with vanishing first Betti number and up to 6 tetrahedra. Exactly 142 of them attain equality in all three bounds. For more details about these cases of equality and the average number of colourings for $5 \leq r \leq 7$ in the census, see Table 2.

Note that the sharp bounds from Theorem 8 suggest that the over count of the general bound from Proposition 7(iii) is only linear in r.

3.5 An algorithm to compute $\mathrm{TV}_{r,1}$, r odd

In this section we describe a significant exponential speed-up for computing $\mathrm{TV}_{r,1}(\mathfrak{T})$ in the case where r is odd and $\mathfrak{T}$ does not contain any two-sided projective planes[1]. Note that the case of r odd is of importance for 3-sphere recognition problem. The main ingredients for this speed-up are:

- The *crushing and expanding procedure* for closed 3-manifolds as described by Burton, and Burton and Ozlen, which turns an arbitrary v-vertex triangulation into a number of smaller 1-vertex triangulations in polynomial time [3, 6];
- A classical result about Turaev-Viro invariants due to Turaev [15], Roberts [14], and Kirby and Melvin [10] stating that there exists a scaled version $\mathrm{TV}'_{r,1} = \gamma_r \mathrm{TV}_{r,1}$ which is multiplicative under taking *connected sums*[2], i.e., $\mathrm{TV}'_{r,1}(M\#N) = \mathrm{TV}'_{r,1}(M)\mathrm{TV}'_{r,1}(N)$ (see Theorem 2);
- Another classical result due to the same authors and publications stating that, for r odd, we have

$$\mathrm{TV}_{r,1}(\mathfrak{T}) = \mathrm{TV}_{3,1}(\mathfrak{T}) \cdot \mathrm{TV}_{r,1}(\mathfrak{T}, [0]),$$

  and thus $\mathrm{TV}_{r,1}(\mathfrak{T}, [0])$ and $\mathrm{TV}_{3,1}(\mathfrak{T})$ are sufficient to compute $\mathrm{TV}_{r,1}(\mathfrak{T})$ (see Theorem 3);
- Proposition 7(ii) stating that computing $\mathrm{TV}_{r,1}(\mathfrak{T}, [0])$ of a 1-vertex closed 3-manifold triangulation can be done by only enumerating colourings with all integer colours.

Input. A v-vertex n-tetrahedra triangulation of a closed 3-manifold $\mathfrak{T}$

1. If $\mathfrak{T}$ has more than one vertex, apply the crushing and expanding procedure to $\mathfrak{T}$ as described in [3] and [6] respectively. It is not necessary to understand this procedure in detail. We only need this step to efficiently transform an arbitrary v-vertex n-tetrahedra triangulation $\mathfrak{T}$ into a number of triangulations $\mathfrak{T}_i$, $1 \le i \le s$, such that the following properties hold.
   - For $m \le s$, every triangulation $\mathfrak{T}_i$, $1 \le i \le m$, is a 1-vertex n_i-tetrahedron triangulation;
   - For $m < \ell \le s$, the topological type of every triangulation $\mathfrak{T}_\ell$ can be detected in polynomial time, and must be one of only three types (for which Turaev-Viro invariants can be pre-computed in constant time);
   - We have $(s-m) + \sum_{i=1}^{m} n_i \le n$;
   - We have $\mathfrak{T} \cong \mathfrak{T}_1 \# \ldots \# \mathfrak{T}_s$, i.e., $\mathfrak{T}$ is the connected sum of the $\mathfrak{T}_i$, $1 \le i \le s$.

   If $\mathfrak{T}$ contains a two-sided projective plane the crushing procedure will detect this fact and the computation is cancelled. The total running time of this step is polynomial.
2. For $1 \le i \le m$, compute $\mathrm{TV}_{r,1}(\mathfrak{T}_i, [0])$. This is the only step of this algorithm with an exponential running time. All other steps can be completed in polynomial time.
3. For all $\mathfrak{T}_i$, compute $\mathrm{TV}_{3,1}(\mathfrak{T}_i)$ – a polynomial time procedure, due to the one-to-one correspondence between admissible colourings in $\mathrm{Adm}(\mathfrak{T}, 3)$ and 1-cocycles of $\mathfrak{T}$.
4. Use Theorem 3 (for r odd we have $\mathrm{TV}_{r,1}(\cdot) = \mathrm{TV}_{3,1}(\cdot)\ \mathrm{TV}_{r,1}(\cdot, [0])$) to obtain $\mathrm{TV}_{r,1}(\mathfrak{T}_i)$.
5. Scale all values from the previous step to $\mathrm{TV}'_{r,1}$, multiply them and re-scale the product. The result equals $\mathrm{TV}_{r,1}(\mathfrak{T})$, by Theorem 2.

[1] This is a technical pre-condition for the procedure to succeed. Triangulations not satisfying this pre-condition are extremely rare.

[2] Building the connected sum $M\#N$ of two manifolds M and N consists of removing a small ball from M and N respectively, and glue them together along their newly created boundaries.

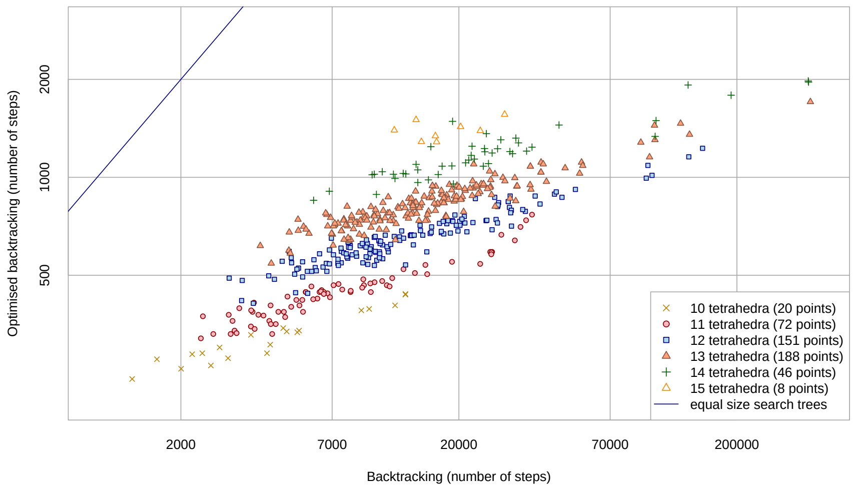


Figure 1 Number of nodes in the search tree visited by the naive algorithm and the optimised backtracking procedure for the 500 first 1-vertex triangulations of the Hodgson-Weeks census.

Running time of the proposed algorithm. The crushing and expanding procedure, computing $\mathrm{TV}_{3,1}(\mathfrak{T})_i$, $1 \leq i \leq s$, computing $\mathrm{TV}_{r,1}(\mathfrak{T}_i, [0])$, $m < i \leq s$, and scaling and multiplying the invariants are all polynomial time procedures [3, 6]. Following Proposition 7(iii) the running time to compute $\mathrm{TV}_{r,1}(\mathfrak{T}_i, [0])$, $1 \leq i \leq m$, is $O(\lfloor r/2 \rfloor^{n_i+1})$ (remember, $\mathfrak{T}_i$ is a 1-vertex triangulation). The overall running time is thus $O(\lfloor r/2 \rfloor^{n+1})$. The same procedure can be applied to improve the fixed parameter tractable algorithm as presented in [5] – which is also based on enumerating colourings – to get the running time $O(n \lfloor r/2 \rfloor^{6(k+1)} k^2 \log r)$, where k is the treewidth of the dual graph of $\mathfrak{T}$.

In Sections 4.1 and 4.2 we show that this improvement has also strong practical implications. In particular, the proposed algorithms allow computations up to several orders of magnitude faster than state of the art procedures to compute Turaev-Viro invariants.

4 Experiments

Here we run large scale experiments to illustrate the interest and performance of the methods introduced above. Implementations will appear within the 3-manifold software `Regina` [4].

We use two data sets for our experiments, both taken from large "census databases" of 3-manifolds to ensure that the experiments are comprehensive and not cherry-picked. The first census contains all 50 817 closed minimal triangulations that can be formed from $n \leq 11$ tetrahedra [2, 12]. This simulates "real-world" computation – the Turaev-Viro invariants were used to build this census. The second data set contains the triangulations from the Hodgson-Weeks census of closed hyperbolic manifolds [8]. This shows performance on larger triangulations, with n ranging from 9 to 20.

The admissible colouring weights may be computed symbolically or numerically, which acts substantially on running times. In the following, we either avoid this difficulty by measuring "discrete data" (like size of search spaces) to represent performance, or we indicate which weight representation we use.

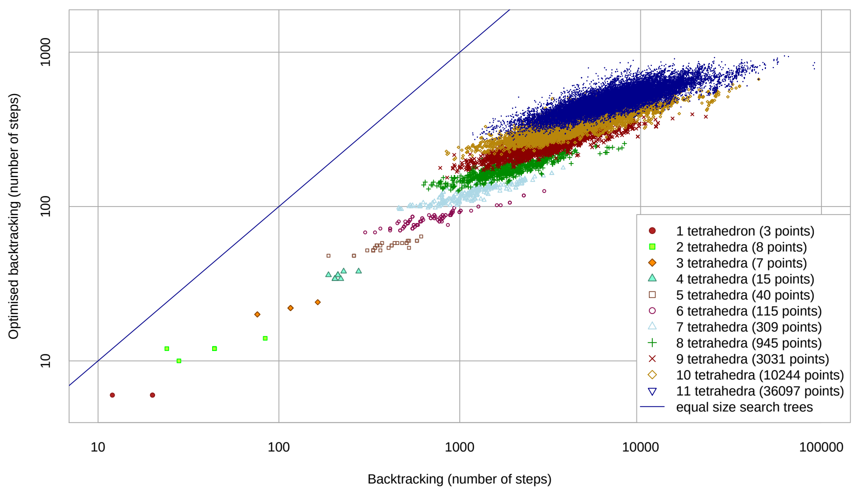


Figure 2 Number of nodes in the search tree visited by the optimised backtracking procedure over the naive algorithm for the 1-vertex minimal closed triangulations.

4.1 Computing $\mathrm{TV}_{r,1}$ with the backtracking method, r odd

We compare experimentally the performance of the naive backtracking algorithm with our proposed backtracking algorithm (Section 3.5) enumerating only colouring at the cohomology class [0]. To do so, we count the number of nodes in the backtracking search tree visited by both algorithms for computing $\mathrm{TV}_{5,1}$ (i) for all triangulations with ≤ 11 tetrahedra in the census of closed minimal triangulations [2] (see Figure 2), and (ii) for the first 500 triangulations of the Hodgson-Weeks census, with $10 \leq n \leq 15$, [8] (see Figure 1). These triangulations all have one vertex, and the improvement is solely due to the reduction of the space of colourings studied above (in particular, the crushing step is not applied).

Because a colouring may be declared non-admissible before colouring all edges of the triangulation, the standard backtracking algorithm visits generally fewer nodes than the $O((r-1)^{n+1})$, for a triangulation with n tetrahedra, predicted by the worst case complexity analysis. Despite this fact, the improvements of our algorithm for the minimal triangulations census range from factors 2 to 117. Improvements in the Hodgson-Weeks census, which contains much larger triangulations, range from factors 5.6 to 215. On both data sets, the range of improvements rapidly grows larger as the size of the triangulations increase. We confirm this observation below.

4.2 Computing $\mathrm{TV}_{r,1}$ with the FPT algorithm, r odd

As demonstrated in [5], the fixed parameter tractable (FPT) algorithm is the most efficient procedure to compute Turaev-Viro invariants experimentally. Improving the running time of this implementation is thus highly significant in practice.

We compare the running times of the FPT algorithm from [5] with the optimised FPT algorithm relying on the enumeration of colourings at the trivial cohomology class, as presented in Section 3.5. Here, the enumeration of colourings is done within the bags of the tree decomposition of the dual graph of the triangulation [5]. Turaev-Viro invariants are computed with floating point arithmetic. Figure 4 represents the running time of both algorithms on the census of closed minimal triangulations of up to 11 tetrahedra, for $r = 5$. All triangulations only have one vertex. We have removed from the timings triangulations with

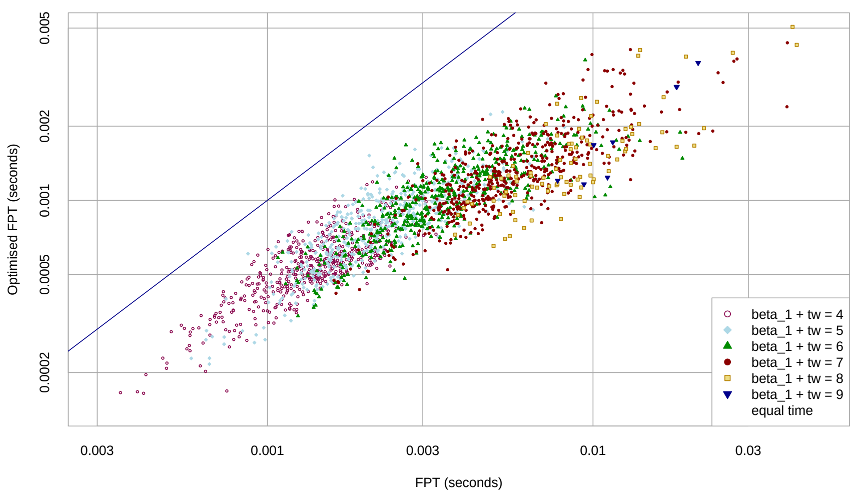


Figure 3 Comparison of the running times of the FPT algorithm presented in [5] and the FPT algorithm with the enumeration procedure introduced in Section 3.5, to compute $\mathrm{TV}_{5,1}$ on the Hodgson-Weeks census with $n \leq 20$ tetrahedra (with one vertex and $\mathrm{TV}_{3,1} \neq 0$). We use the parameter $\mathrm{tw} + \beta_1$ as a measure of "difficulty" for the computation. For readability, the plot presents a sparsified cloud (500 uniform random samples for each parameter value).

$\mathrm{TV}_{3,1} = 0$, as we can conclude in polynomial time, in our implementation, that $\mathrm{TV}_{r,1} = 0$ using the formula involving $\mathrm{TV}_{3,1}$ in Section 3.5. Consequently, Figure 4 illustrates the improvement solely due to the enumeration of colourings at the trivial cohomology class. For our implementation of the FPT algorithm, we include the timings of *all* steps of the algorithm presented in Section 3.5; the dominating step is naturally the computation of $\mathrm{TV}_{r,1}(\mathfrak{T}, [0])$ (step 2), which is the only exponential step of the procedure.

Figure 4 shows a clear improvement of the running time of our algorithm. Most interestingly, this range of improvement seems to increase (points getting further away from the diagonal) for triangulations on which the standard FPT algorithm is slower.

To confirm this tendency on larger scales, we run the computational power-intensive computation of $\mathrm{TV}_{11,1}$ on the first 1000 triangulations of the Hodgson-Weeks census (Figure 3), with triangulations with up to 20 tetrahedra. We observe that 40% of the total running time of the standard FPT algorithm over the 1000 triangulations is spent on only 10 of them. On these 10 inputs, our implementation is up to 130 times faster, and 29 times faster in average, reducing the total running time for these "hardest" 10 triangulations from several hours to a few minutes of computation.

4.3 Experiments for computing $\mathrm{Adm}(\mathfrak{T}, 4)$

In this section, we study experimentally the bounds on the number of admissible colourings for $r = 4$, and the efficiency of the algorithm for $\mathrm{TV}_{4,1}$, introduced in Sections 3.2 and 3.3, depending on them. Table 1 gives details on the bounds given by Theorem 6, and hence the worst case number of steps of our algorithm to compute $\mathrm{TV}_{4,1}$, and the average number of steps the backtracking algorithm requires to compute $\mathrm{TV}_{4,1}$. We run the experiments on all minimal triangulations with up to 6 tetrahedra, sorted by Betti number β_1.

We note that the actual number of nodes visited by the backtracking algorithm is smaller than the worst case bound, but it is significantly larger than the upper bound of Equation (2) in Theorem 6. Additionally, the bound given by Equation (2) is very close

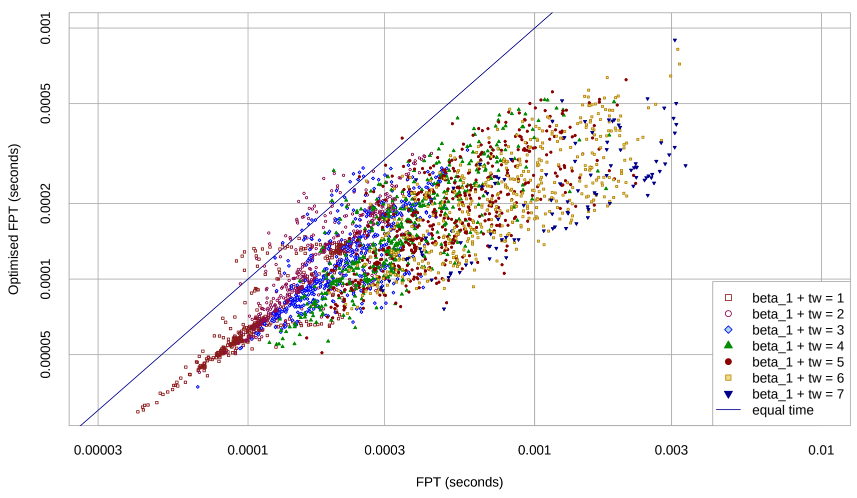


Figure 4 Comparison of the running times of the FPT algorithm from [5] and the FPT algorithm with the enumeration procedure introduced in Section 3.5, to compute $\mathrm{TV}_{5,1}$ on the census of minimal triangulations with $n \leq 11$ tetrahedra (with one vertex and $\mathrm{TV}_{3,1} \neq 0$). We use the parameter $\mathrm{tw} + \beta_1$ as a measure of "difficulty" for the computation. For readability, the plot presents sparsified cloud (500 uniform random samples for each parameter value).

Table 1 "$\#\mathfrak{T}$" lists the number of triangulations contained in the n-tetrahedra census of minimal triangulations with first Betti number $\beta_1(\mathfrak{T}, \mathbb{Z}_2)$, "# eq. (2)" lists the number of triangulations satisfying equality in Inequality (2). Below, the average number of nodes of the search tree visited by the backtracking algorithm ("# tree"), the bound "Eqn. (2)" given by Inequality (2), and the average number "Av." of admissible colourings in $\mathrm{Adm}(\mathfrak{T}, 4)$ are listed.

(n, β_1)	$(1,1)$	$(2,1)$	$(2,2)$	$(3,1)$	$(3,2)$	$(4,1)$	$(4,2)$	$(5,1)$	$(5,2)$	$(6,1)$	$(6,2)$	$(6,3)$
$\#\mathfrak{T}$	1	5	1	27	3	205	19	1858	184	21459	2516	34
#eq. (2)	1	5	1	14	1	67	4	261	10	1574	47	0
#tree	12.0	33.0	39.0	46.4	69.0	75.2	110.1	93.1	159.2	120.4	214.5	413.2
Eqn. (2)	4.0	6.0	10.0	7.7	14.7	13.1	21.8	20.4	35.8	34.6	58.0	94.5
Av.	4.0	6.0	10.0	6.3	11.3	8.7	15.3	9.3	18.6	10.7	22.0	41.4

to the average number of admissible colourings of the triangulations, which underlines the fact that our algorithm for $\mathrm{TV}_{4,1}$ has a practical output-sensitive behaviour in the number of admissible colourings. Finally, our bounds on $|\mathrm{Adm}(\mathfrak{T}, 4)|$ are sharp on 1985 out of the 26,312 triangulations of the experiment.

5 Experiments on triangulations with vanishing β_1

In this section we provide experimental details on the number of admissible colourings of triangulations with $\beta_1 = 0$, and the ability of $\mathrm{TV}_{r,1}$ to distinguish between these manifolds and the 3-sphere, which is of particular importance in 3-manifold topology.

In Table 2 we provide details on the number of admissible colourings of triangulations with up to 6 tetrahedra and $\beta_1 = 0$. In particular, the table shows that the bounds from Proposition 8 are tight, and much finer in general than the naive bound $(r-1)^{n+v}$.

As evidence for the interest of the algorithm to compute $\mathrm{TV}_{r,1}$, r odd, introduced in Section 3.5, we analyse the ability of $\mathrm{TV}_{r,1}$, $r \in \{3,5,7,9\}$, to distinguish 3-manifolds

Table 2 Number "# trigs." of 1-vertex triangulations $\mathfrak{T}$ of manifolds with $\beta_1(\mathfrak{T}, \mathbb{Z}_2) = 0$ and n tetrahedra, $1 \leq n \leq 6$. Number "#sharp" of such triangulations satisfying equality in all bounds from Theorem 8 (third column), and the average number "$\overline{|\mathrm{Adm}(\mathfrak{T}, r)|}$" of admissible colourings in $\mathrm{Adm}(\mathfrak{T}, r)$, $5 \leq r \leq 7$ (columns 6, 9, and 12), compared to the naive upper bound "$(r-1)^{v+n}$" (columns 4, 7, and 10) and the new upper bounds given by Theorem 8 (columns 5, 8, and 11).

n	#trig.	#sharp	$(5-1)^{n+v}$	2^n+1	$\overline{\lvert\mathrm{Adm}(\mathfrak{T},5)\rvert}$	$(6-1)^{n+v}$	3^n+1	$\overline{\lvert\mathrm{Adm}(\mathfrak{T},6)\rvert}$	$(7-1)^{n+v}$	3^n+1	$\overline{\lvert\mathrm{Adm}(\mathfrak{T},7)\rvert}$
1	2	1	16	3	2.50	25	4	3.00	36	4	4.00
2	7	3	64	5	4.00	125	10	6.86	216	10	8.86
3	36	5	256	9	5.61	625	28	12.22	1,296	28	17.28
4	255	14	1,024	17	8.31	3,125	82	23.46	7,776	82	35.30
5	2305	30	4,096	33	12.02	15,625	244	43.00	46,656	244	70.44
6	24597	89	16,384	65	17.71	78,125	730	80.15	279,936	730	142.23

Table 3 Summary of the ability of $\mathrm{TV}_{r,1}$, $3 \leq r \leq 9$, to distinguish 3-manifolds with trivial (integral) homology up to complexity 11 from the 3-sphere. X/Y denotes the success rate, i.e., there are Y manifolds, X of which can be distinguished from the 3-sphere by the respective invariant.

n	$\mathrm{TV}_{3,1}$	$\mathrm{TV}_{4,1}$	$\mathrm{TV}_{5,1}$	$\mathrm{TV}_{6,1}$	$\mathrm{TV}_{7,1}$	$\mathrm{TV}_{8,1}$	$\mathrm{TV}_{9,1}$	$\mathrm{TV}_{5,1}$ and $\mathrm{TV}_{7,1}$
5	0/1	0/1	**1/1**	0/1	**1/1**	1/1	1/1	**1/1**
7	0/1	0/1	**1/1**	0/1	**1/1**	0/1	1/1	**1/1**
8	0/3	0/3	**1/3**	0/3	**3/3**	3/3	3/3	**3/3**
9	0/4	0/4	**3/4**	0/4	**3/4**	1/4	3/4	**4/4**
10	0/8	0/8	**5/8**	0/8	**7/8**	3/8	6/8	**8/8**
11	0/19	0/19	**11/19**	0/19	**16/19**	13/19	16/19	**18/19**

from the 3-sphere. Since homology can be computed in polynomial time, we only consider 3-manifolds M which cannot be distinguished from the 3-sphere using integral homology (i.e., $\beta_1(M, \mathbb{F}) = 0$, for any choice of field $\mathbb{F}$). There are 36 distinct such 3-manifolds of *complexity* at most 11, meaning, they can be triangulated with 11 tetrahedra or less. Due to Proposition 7(i) we already know that none of them can be distinguished from the 3-sphere by $\mathrm{TV}_{3,1}$. $\mathrm{TV}_{5,1}$, $\mathrm{TV}_{7,1}$, and $\mathrm{TV}_{9,1}$ distinguish 22, 31, and 30, a combination of $\mathrm{TV}_{5,1}$ and $\mathrm{TV}_{7,1}$ only fails once, and a combination of all three invariants never fails to distinguish them from the 3-sphere. See Table 3 for details.

Together with the favourable timings presented in Section 4, this indicates that our new approach to compute the Turaev-Viro invariants for odd values of r gives a practical fast way to distinguish manifolds with $\beta_1 = 0$ from the 3-sphere in many cases.

References

1 Benjamin A. Burton. Structures of small closed non-orientable 3-manifold triangulations. *J. Knot Theory Ramifications*, 16(5):545–574, 2007.

2 Benjamin A. Burton. Detecting genus in vertex links for the fast enumeration of 3-manifold triangulations. In *Proceedings of ISSAC*, pages 59–66. ACM, 2011.

3 Benjamin A. Burton. A new approach to crushing 3-manifold triangulations. *Discrete Comput. Geom.*, 52(1):116–139, 2014.

4 Benjamin A. Burton, Ryan Budney, Will Pettersson, et al. Regina: Software for 3-manifold topology and normal surface theory. `http://regina.sourceforge.net/`, 1999–2014.

5 Benjamin A. Burton, Clément Maria, and Jonathan Spreer. Algorithms and complexity for Turaev-Viro invariants. In *Proceedings of ICALP 2015*, pages 281–293. Springer, 2015.

6 Benjamin A. Burton and Melih Ozlen. A fast branching algorithm for unknot recognition with experimental polynomial-time behaviour. *arXiv:1211.1079*, 2012.

7 Allen Hatcher. *Algebraic Topology.* Cambridge University Press, Cambridge, 2002. `http://www.math.cornell.edu/~hatcher/AT/ATpage.html`.

8 Craig D. Hodgson and Jeffrey R. Weeks. Symmetries, isometries and length spectra of closed hyperbolic three-manifolds. *Experiment. Math.*, 3(4):261–274, 1994.

9 William Jaco and J. Hyam Rubinstein. 0-efficient triangulations of 3-manifolds. *J. Differential Geom.*, 65(1):61–168, 2003.

10 Robion Kirby and Paul Melvin. The 3-manifold invariants of Witten and Reshetikhin-Turaev for sl$(2, \mathbf{C})$. *Invent. Math.*, 105(3):473–545, 1991. `doi:10.1007/BF01232277`.

11 Robion Kirby and Paul Melvin. Local surgery formulas for quantum invariants and the Arf invariant. *Geom. Topol. Monogr.*, pages (7):213–233, 2004. `doi:10.2140/gtm.2004.7.213`.

12 Sergei Matveev. *Algorithmic Topology and Classification of 3-Manifolds.* Number 9 in Algorithms and Computation in Mathematics. Springer, Berlin, 2003.

13 Sergei Matveev et al. Manifold recognizer. `http://www.matlas.math.csu.ru/?page=recognizer`, accessed August 2012.

14 Justin Roberts. Skein theory and Turaev-Viro invariants. *Topology*, 34(4):771–787, 1995. `doi:10.1016/0040-9383(94)00053-0`.

15 Vladimir G. Turaev. *Quantum Invariants of Knots and 3-Manifolds*, volume 18 of *de Gruyter Studies in Mathematics*. Walter de Gruyter & Co., Berlin, revised edition, 2010. `doi:10.1515/9783110221848`.

16 Vladimir G. Turaev and Oleg Y. Viro. State sum invariants of 3-manifolds and quantum $6j$-symbols. *Topology*, 31(4):865–902, 1992.

The Computational Complexity of Genetic Diversity

Ruta Mehta[1], Ioannis Panageas[*2], Georgios Piliouras[†3], and Sadra Yazdanbod[4]

1 **University of Illinois at Urbana-Champaign, USA**
`mehta.ruta@gmail.com`

2 **Georgia Institute of Technology, Atlanta, USA**
`ioannis@gatech.edu`

3 **Singapore University of Technology and Design, Singapore**
`georgios.piliouras@gmail.com`

4 **Georgia Institute of Technology, Atlanta, USA**
`yazdanbod@gatech.edu`

Abstract

A key question in biological systems is whether genetic diversity persists in the long run under evolutionary competition, or whether a single dominant genotype emerges. Classic work by Kalmus in 1945 [14] has established that even in simple diploid species (species with chromosome pairs) diversity can be guaranteed as long as the heterozygous[1] individuals enjoy a selective advantage. Despite the classic nature of the problem, as we move towards increasingly polymorphic traits (e.g., human blood types) predicting diversity (and its implications) is still not fully understood. Our key contribution is to establish complexity theoretic hardness results implying that even in the textbook case of single locus (gene) diploid models, predicting whether diversity survives or not given its fitness landscape is algorithmically intractable.

Our hardness results are structurally robust along several dimensions, e.g., choice of parameter distribution, different definitions of stability/persistence, restriction to typical subclasses of fitness landscapes. Technically, our results exploit connections between game theory, nonlinear dynamical systems, and complexity theory and establish hardness results for predicting the evolution of a deterministic variant of the well known multiplicative weights update algorithm in symmetric coordination games; finding one Nash equilibrium is easy in these games. In the process we characterize stable fixed points of these dynamics using the notions of Nash equilibrium and negative semidefiniteness. This as well as hardness results for decision problems in coordination games may be of independent interest. Finally, we complement our results by establishing that under randomly chosen fitness landscapes diversity survives with significant probability. The full version of this paper is available at `http://arxiv.org/abs/1411.6322`.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases Dynamical Systems, Stability, Complexity, Optimization, Equilibrium

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.65

[*] Ioannis Panageas was supported by NSF EAGER award grants CCF-1415496 and CCF-1415498.
[†] Georgios Piliouras was supported by SUTD grant SRG ESD 2015 097.
[1] Having different alleles for a gene on two chromosomes.

1 Introduction

The beauty and complexity of natural ecosystems have always been a source of fascination and inspiration for the human mind. The exquisite biodiversity of Galapagos' ecosystem, in fact, inspired Darwin to propose his theory of natural selection as an explanatory mechanism for the origin and evolution of species. This revolutionary idea can be encapsulated in the catch phrase "survival of the fittest"[2]. Natural selection promotes the survival of those genetic traits that provide to their carriers an evolutionary advantage.

A marker of the true genius behind any paradigm is its longevity and in this respect the success of natural selection is without precedent. The formalized study of evolution, dating back to the work of Fisher, Haldane, and Wright in the beginning of the twentieth century, still and to a large extent focuses on simple, almost toy-like, alas concrete, models of this famous aphorism and the experimental and theoretical analysis of them. Arguably, the most influential result in the area of mathematical biology is Fisher's fundamental theorem of natural selection (1930) [9]. It states that the rate of increase in fitness of any organism at any time is equal to its genetic variance in fitness at that time. In the classical model of population genetics (Fisher-Wright-Haldane, discrete or continuous version) of single locus (one gene) multi-allele diploid models[3] it implies that the average fitness of the species populations is always strictly increasing unless we are at an equilibrium. In fact, convergence to equilibrium is point-wise even if there exist continuum of equilibria (See [21] and references therein). From a dynamical systems perspective, this is a rather strong characterization, since it establishes that the average fitness acts as a Lyapunov function for the system and that every trajectory converges to an equilibrium.

Besides the purely dynamical systems interpretation, an alternative, more palpable, game theoretic interpretation of these genetic systems is possible. Specifically, these systems can be interpreted as symmetric coordination/partnership two-agent games[4] where both agents, starting with the same mixed initial strategy, apply (discrete) replicator dynamics[5]. The analogies are as follows: The two players correspond each to a locus (or gene) on a pair of homologous chromosomes[6] and the alleles are their strategies. When both players choose a strategy, say i and j, an individual (i, j) is defined whose fitness, say A_{ij}, is the payoff to both players, hence we have a coordination game. Furthermore, allele pairs are unordered so we have $A_{ij} = A_{ji}$, *i.e.*, A is symmetric and so is the game. The frequencies of the alleles in the initial population, namely $\boldsymbol{x} := (x_1, ..., x_n) \in \Delta_n$[7] of n different alleles, corresponds to the initial common mixed strategy of both players. In each generation, every individual from the population mates with another individual picked at random from the population, and the updates of the mixed strategies/allele frequencies are captured by replicator/MWUA dynamics, *i.e.*,

$$x'_i = x_i \frac{\sum_j A_{ij} x_j}{\boldsymbol{x}^T A \boldsymbol{x}} \tag{1}$$

[2] The phrase "survival of the fittest" was coined by Herbert Spencer.
[3] We present information related to biology in Section A.1.
[4] A coordination/partnership game is a game where at each outcome all agents receive the same utility.
[5] Replicator dynamics (as well as their discrete variants) are close analogues to the well known multiplicative updates (MWUA) family of dynamics [19].
[6] Most multicellular organisms have two sets of chromosomes; *i.e.*, they are diploid. These chromosomes are referred to as homologous chromosomes. If both alleles at a locus (or gene) on the homologous chromosomes are the same, they and the organism are homozygous with respect to that gene. If the alleles are different, they and the organism are heterozygous with respect to that gene. See section A.
[7] Δ_n denotes the simplex of dimension n.

where x_i' is the proportion of allele i in the next generation (for details see Section 4). In game theoretic language, the fundamental theorem of natural selection implies that the social welfare $\boldsymbol{x}^T A \boldsymbol{x}$ (average fitness in biology terms) of the game acts as potential for the game dynamics. This implies convergence to fixed points. Fixed points are superset of Nash equilibria where each strategy played with positive probability has the same average payoff.

We say that population is genetically diverse if at least two alleles have non-zero proportion in the population, *i.e.,* allele frequencies form a *mixed (polymorphic) strategy*. The game theoretic results do not provide insight on the survival of genetic diversity. One way to formalize this question is whether there exists a mixed fixed point that the dynamics converges to with positive probability, given a uniformly random starting point in Δ_n. The answer to this question for the minimal case of $n = 2$ alleles (alleles b/B, individuals $bb/bB/BB$) is textbook knowledge and can be traced back to the classic work of Kalmus (1945) [14]. The intuitive answer here is that diversity can survive when the heterozygote individuals, bB, have a fitness advantage. Intuitively, this can be explained by the fact that even if evolution tries to dominate the genetic landscape by bB individuals, the random genetic mixing during reproduction will always produce some bb, BB individuals, so the equilibrium that this process is bound to reach will be mixed. On the other hand, it is trivial to create instances where homozygote individuals are the dominant species regardless of the initial condition.

As we increase the size/complexity of the fitness landscape, not only is not clear that a tight characterization of the diversity-inducing fitness landscape exists (a question about global stability of nonlinear dynamical systems), but also, it is even less clear whether one can decide efficiently whether such conditions are satisfied by a given fitness landscape (a computational complexity consideration). How can one address this challenge and moreover, how can one account for the apparent genetic diversity of the ecosystems around us?

In a nutshell, we establish that the decision version of the problem is computationally hard, by sandwiching limit points of the dynamics between various stability notions. This core result is shown to be robust across a number of directions. Deciding the existence of stable (mixed) polymorphic equilibria remains hard under a host of different definitions of stability examined in the dynamical systems literature. The hardness results persist even if we restrict the set of allowable landscape instances to reflect typical instance characteristics. Despite the hardness of the decision problems, randomly chosen fitness landscapes are shown to support polymorphism with significant probability (at least 1/3). The game theoretic interpretation of our results allow for proving hardness results for understanding standard game theoretic dynamics in symmetric coordination games. We believe that this is an important result of independent interest as it points out at a different source of complexity in understanding social dynamics.

2 Technical Overview

To study survival of diversity in diploidy, we need to characterize limiting population under evolutionary pressure. We focus on the simplest case of single locus (one gene) species. For this case, evolution under natural selection has been shown to follow replicator dynamics in symmetric two-player coordination games [21], where the genes on two chromosomes are players and alleles are their strategies as described in the introduction. Losert and Akin established point-wise convergence for this dynamics through a potential function argument [21]; here average fitness $\boldsymbol{x}^T A \boldsymbol{x}$ is the potential. The limiting population corresponds to fixed points (FP), and so to make predictions about diversity (if the limiting population has support size at least 2) we need to characterize and compute these limiting FPs.

Let $\mathcal{L}$ denote the set of fixed points (FP) with region of attraction[8] of positive (Lebesgue) measure. Hence, given a random starting point replicator dynamics converges to such a FP with positive probability, *i.e.,* the set of FP with region of attraction of positive measure. It seems that an exact characterization of $\mathcal{L}$ is unlikely. Establishing necessary and sufficient conditions so that a FP has a region of attraction of positive measure in classes of nonlinear systems is a rather formidable task. Instead we try to capture this property as closely as possible through different stability notions. First we consider two standard notions defined by Lyapunov, the *stable* and *asymptotically stable* FP. Informally, if we start close to a *stable* FP then we stay close to it forever, while in case of *asymptotically stable* FP furthermore the dynamics converges to it (see Section 4.3). Thus *asymptotically stable* $\subseteq \mathcal{L}$ follows, *i.e.,* an asymptotically stable FP has region of attraction of positive measure (*e.g.*, a small ball around the FP). Note that $\mathcal{L}$ may have points that are not (asymptotically) stable.

There exist some well known connections between stability notions and properties of the (absolute) eigenvalues (EVal) of the Jacobian of the update rule (function) of the dynamics are well known: if the Jacobian at a FP has an Eval > 1 then the FP is *un-stable* (not *stable*), and if all Eval < 1 then it is asymptotically stable. The case when all Eval ≤ 1 with equality holding for some, is the ambiguous one. In that case we can say nothing about the stability because the Jacobian does not suffice. We will call these FPs *linearly-stable.* At a FP, say $\boldsymbol{x}$, if some EVal > 1 then the direction of corresponding eigenvector is repelling, and therefore any starting vector with a component of this vector can never converge to $\boldsymbol{x}$. Thus points converging to $\boldsymbol{x}$ can not have positive measure. Using this as an intuition we show that in our dynamical systems $\mathcal{L} \subseteq$ *linearly-stable* FPs. In other words the set of initial points so that the dynamics converges to linearly-*un*-stable FPs has zero measure (Theorem 13). This theorem is heavily utilized to understand the (non-)existence of diversity.

Efficient computation requires efficient verification. However, note that whether a given FP is (asymptotically) stable or not does not seem easy to verify. To achieve this, one of the contributions of this paper is the definition of two more notions: *Nash stable* and *strict Nash stable.*[9] It is straightforward to check that Nash equilibria (NE) of the corresponding coordination game described in introduction are FPs of the replicator dynamics (Equations 1,2) but not vice-versa. Keeping this in mind we define *Nash stable* FP, which is a NE with the additional property that the sub-matrix corresponding to its support satisfies certain negative semi-definiteness. The latter condition is derived from the fact that *stability* is related to local optima of $\boldsymbol{x}^T A \boldsymbol{x}$ and also from Sylvester's law of inertia [38] (see Section 5 and proofs). For *strict Nash stable* both conditions are strict, namely strict NE and negative definiteness. Combining all of these notions we show the following:

▶ **Theorem 1.**

$$\begin{array}{ccccccccc} \textit{Strict Nash stable} & \subseteq & \textit{Asymptotically stable} & \subseteq & \mathcal{L} & \subseteq & \textit{linearly-stable} & = & \textit{Nash stable} \\ & & \cap\!\mid & & & & & & \\ & & \textit{stable} \subseteq & \textit{linearly-stable} & = & \textit{Nash stable} & & & \end{array}$$

We note that the sets of asymptotically stable, stable, $\mathcal{L}$ and linearly stable FPs of Theorem 1 do not coincide in general.[10] For example, let x_{t+1} be the next step for the

[8] Region of attraction of an FP is the set of all initial points so that dynamics converges to it.

[9] These two notions are not the same as evolutionary stable strategies/states.

[10] We also note that generically these sets coincide. It can be shown [22] that given a fitness matrix, its entries can be perturbed to ensure no eigenvalue on the unit circle for the Jacobian at any fixed point. Such fixed points are called *hyperbolic.* Formally, if we consider the dynamics as an operator (called *Fisher* operator) then the set of hyperbolic operators is dense in the space of Fisher operators.

following update rules:

$$f(x_t) = \frac{1}{2}x_t, g(x_t) = x_t - \frac{1}{2}x_t^2, h(x_t) = x_t + \frac{x_t^3}{2}, d(x_t) = x_t.$$

Then for dynamics governed by f, 0 is asymptotically stable, stable and linearly stable, and hence is also in $\mathcal{L}$. While for g it is linearly stable and is in $\mathcal{L}$, but is not stable or asymptotically stable. For d it is linearly stable and stable, but not asymptotically stable and is not in $\mathcal{L}$. Finally, for h it is only linearly-stable, and does not belong to any other class. Our primary goal was to see if diversity is going to survive. We formalize this by checking whether set $\mathcal{L}$ contains a mixed point, *i.e.,* where more than one alleles have non-zero proportion, implying that *diversity survives with some positive probability*, where the randomness is w.r.t the random initial $\boldsymbol{x} \in \Delta_m$. In Section 7 we show that for all five notions of stability, checking existence of mixed stable FP is NP-hard.

Our reductions are from k**-clique** - given an undirected graph check if it has a clique of size k; a well known NP-hard problem. Given an instance G of k*-clique*, we will construct a symmetric matrix A as shown in Figure 1, and consider coordination game (A, A). We show game (A, A) has the following two properties.

(i) If G has a clique of size k then (A, A) has a mixed *strict* Nash stable equilibrium. Therefore, it has a mixed stable FP for any notions of the (strict) Nash stability, (asymptotic) stability, linearly-stability, and $\mathcal{L}$ because of Theorem 1.

(ii) If (A, A) has a mixed Nash stable equilibrium then G has a clique of size k. Therefore, if (A, A) has a mixed stable FP for all notions of the (strict) Nash stability, (asymptotic) stability, linearly-stability, and $\mathcal{L}$ then G has a clique of size k because of Theorem 1.

Note that the above two properties imply NP-hardness for checking existence of mixed stable FP for all five notions of stability as well as set $\mathcal{L}$. The latter implies NP-hardness for checking survival of diversity in diploid species even with single gene.

▶ **Theorem 2** (Informal). *Given a symmetric matrix A, it is NP-hard to check if replicator dynamics with payoff A has mixed (asymptotically) stable, linear-stable, or (strict) Nash stable fixed points. A common reduction for all together with Theorem 1 will imply that it is NP-hard to check whether diversity survives for a given fitness matrix.*

The main idea in the construction of matrix A (Figure 1) is to use a modified version of the adjacency matrix E of the graph as one of the blocks in the payoff matrix such that the existence of a clique of size k or more implies a stable Nash equilibrium in that block, and all stable mixed equilibria are only in that block. Here E' is the modification of E where off-diagonal zeros are replaced with $-h$ where h is a large (polynomial-size number). Note that A is $2n \times 2n$, where the first n strategies correspond to the n nodes of the graph. To argue (i), given a k-clique we construct a maximal clique containing it, say of size m, and then show that the distribution that assigns probability $\frac{1}{m}$ to the strategies corresponding to nodes in the clique, and zero otherwise, is a (strict Nash) stable FP. To show (ii), we use the negative semi-definiteness property of Nash stable (which contains all other notions), which implies $A_{ii} \leq 2A_{ij}$ for all i, j in the support a Nash stable FP. Therefore if $h > 2(k-1)$, then none of $\{n+1, \ldots, 2n\}$ can be in the support. Next we concentrate only on strategies that have big enough probability, call it set S. The Nash equilibrium property of Nash stable states can be shown to force each non-zero probability to be $\leq \frac{1}{k}$. Using these facts we show that $|S| \geq k$ and the corresponding vertices in G form a clique. Thus, we prove the existence of a k-clique.

The fitness matrix constructed for the hardness results is rather specific. Do these hardness results carry over to "typical" (at least not completely worst case) instances of fitness landscapes? What is the complexity of checking if a given allele survives? We show that our computational intractability results still apply to these settings.

There has been a lot of work on NP-hardness for decision versions of Nash equilibrium in general games [11, 5, 34, 10], where finding one equilibrium is also PPAD-hard [3]. Whereas to the best of our knowledge these are the first NP-hardness results for coordination games, where finding one Nash equilibrium is easy, and therefore may be of independent interest. Finally, in Section 6 we show that even though checking the survival of diversity is hard, on average diversity persists.

▶ **Theorem 3** (Informal). *If the entries of a fitness matrix are i.i.d. on a continuous distribution then with significantly high probability, at least $\frac{1}{3}$, diversity will surely survive.*

Survival is ensured if every fixed point in $\mathcal{L}$ is mixed. This itself is guaranteed as long as every diagonal entry (i, i) of the fitness matrix is dominated by some entry in its row or column. We can lower bound the probability of the latter *by a constant* for a random symmetric matrix (from continuous distribution) of *any size*. The tricky part is to avoid correlation arising due to symmetry and we achieve this using inclusion-exclusion arguments.

3 Related Work

In the last few years we have witnessed a rapid cascade of theoretical results on the intersection of computer science and evolution. Livnat *et al.* [20] introduced the notion of mixability, the ability of an allele to combine itself successfully with other alleles within a specific population. In [2, 1] connections were established between haploid evolution and game theoretic dynamics in coordination games. Even more recently Meir and Parkes [24] provided a more detailed examination of these connections. These dynamics are close variants of the standard (discrete) replicator dynamics [12]. Replicator dynamics is closely connected to the multiplicative weights update algorithm [19]. Analogous game theoretic interpretations are known for diploids [21].

Analyzing limit sets of dynamical systems is a critical step towards understanding the behavior of processes that are inherently dynamic, like evolution. There has been an upsurge in studying the complexity of computing these sets. Quite few works study such questions for dynamical systems governed by arbitrary continuous functions or ODEs [15, 16, 35]. Limit cycles are inherently connected to dynamical systems and recent works by Papadimitriou and Vishnoi [29] showed that computing a point on an approximate limit cycle is PSPACE-complete. Cyclic limit sets also arise in game theoretic dynamics [18, 32, 31, 28]. On the positive side, in [27], it was shown that a class of evolutionary Markov chains mix rapidly, where techniques from dynamical systems were used.

The complexity of checking if a game has an evolutionary stable strategy (ESS) has been studied first by Nissan and then by Etessami and Lochbihler [26, 8] and has been nailed down to be Σ_2^P-complete by Conitzer [4]. Unlike our setting here the game is between different species to survive in a common environment. These problems are orthogonal to understanding issues of genetic diversity, and thus not directly comparable to our work.

Other connections between computational complexity and ecology/evolution examine the complexity of finding local/global minima of structured fitness landscapes [37, 17, 39], as well as, complexity questions in regards to the probability that a new invader (or a new mutant) will take over a resident population [33]. The sexual dynamics and the questions about diversity considered here are not captured in any of the settings above.

In [23] Mehta, Panageas and Piliouras examine the question of diversity for haploid species. Despite the systems' superficial similarities the two analyses come to starkly different conclusions. In haploids systems all mixed (polymorphic) equilibria are unstable and evolution converges to monomorphic states. In the case of diploid systems the answer to whether diversity survives or not depends crucially on the geometry of the fitness landscape.

4 Preliminaries

Notations: We use boldface letters, like $\boldsymbol{x}$, to denote column vectors, and x_i is its i^{th} coordinate. Vectors $\mathbf{0}_n$ and $\mathbf{1}_n$ are n-D vectors with all zeros and ones. For matrix A, A_{ij} is the entry in i^{th} row and j-th column and we denote $(A\boldsymbol{x})_i = \sum_j A_{ij}x_j$. By norm $||.||$ we mean $||.||_\infty$. $[n]$ denotes set $\{1,\dots,n\}$, and Δ_n denotes n-D simplex, and let $SP(\boldsymbol{x})$ be $\{i \mid x_i > 0\}$.

4.1 Evolutionary dynamics

Consider a diploid single locus species, *i.e.,* a species with a chromosome pair and single gene. Every gene has a set of representative alleles, e.g., gene for eye color has different alleles for brown, black and blue eyes. Let n be the number of alleles for the single gene of our species, and let these be numbered $1,\dots,n$. An individual is represented by an unordered pair of alleles (i,j) one in each chromosome, and we denote its fitness by A_{ij}; clearly A is symmetric. Here fitness represents its ability to reproduce during a mating. In every generation two individuals are picked uniformly at random from the population, say (i,j) and (i',j'), and they mate. The allele pair of the offspring can be any of the four possible combinations, namely $(i,i'),(i,j'),(i',j),(j,j')$, with equal probability. Let x_i be a random variable that denotes the proportion of the population with allele i. After one generation, the expected number of offsprings with allele i is proportional to $x_i \cdot x_i \cdot (A\boldsymbol{x})_i + 2\frac{1}{2}(1-x_i)x_i \cdot (A\boldsymbol{x})_i = x_i(A\boldsymbol{x})_i$ (x_i^2 stands for the probability that first individual has both his alleles i, *i.e.,* is represented by (i,i) - and thus the offspring will inherit allele i - and $2\frac{1}{2}(1-x_i)x_i$ stands for the probability that the first individual has allele i exactly once in his representation and the offspring will inherit). Hence, if $\boldsymbol{x}$ denote the frequencies of the alleles in the population in the next generation (random variables)

$$E[x_i'|\boldsymbol{x}] = \frac{x_i(A\boldsymbol{x})_i}{\boldsymbol{x}^T A\boldsymbol{x}}.$$

We focus on the *deterministic* version of the equations above, which captures the infinite population model. Thus if $\boldsymbol{x} \in \Delta_n$ represents the proportions of alleles in the current population. Under the evolutionary process of natural-selection (the reproduction happens as described) this proportion changes as per the following multi-variate function $f : \Delta_n \to \Delta_n$ under the infinite population model [21]; in the literature this is often called *Discrete Replicator Dynamics.*

$$\boldsymbol{x}' = f(\boldsymbol{x}) \quad \text{where} \quad x_i' = f_i(\boldsymbol{x}) = x_i\frac{(A\boldsymbol{x})_i}{\boldsymbol{x}^T A\boldsymbol{x}}, \quad \forall i \in [n] \tag{2}$$

where $\boldsymbol{x}'$ are the proportions of the next generation. f is a continuous function with convex, compact domain (= range), and therefore always has a fixed point [13]. Furthermore, limit points of f have to be fixed points, *i.e.,* $\boldsymbol{x}$ such that $f(\boldsymbol{x}) = \boldsymbol{x}$.

▶ **Fact 4.** *Profile $\boldsymbol{x}$ is a fixed point of f iff $\forall i \in [n],\ x_i > 0 \Rightarrow (A\boldsymbol{x})_i = \boldsymbol{x}^T A\boldsymbol{x}$.*

One of the fundamental questions is: starting from arbitrary population of alleles how does the population look like in the limit under evolutionary pressures? Does the system converge? If so, then what are the properties of these limit points? Fisher's Fundamental Theorem of Natural Selection [21, 22] says that mean fitness function $\pi(\boldsymbol{x}) = \boldsymbol{x}^T A \boldsymbol{x}$ (potential function in game theory terms) satisfies the inequality $\pi(f(\boldsymbol{x})) \geq \pi(\boldsymbol{x})$ and the equality holds iff $\boldsymbol{x}$ is a fixed point. Losert and Akin [21, 22] showed that the dynamics above converge point-wise to fixed points and that f is a diffeomorphism in Δ_n.

4.2 Games, Nash equilibria and symmetries

In this paper we consider two-player (locus) games, where each player has finitely many pure strategies (alleles). Let $S_i,\ i = 1, 2$ be the set of strategies for player i, and let $m \overset{\text{def}}{=} |S_1|$ and $n \overset{\text{def}}{=} |S_2|$, then such a game can be represented by two payoff matrices A and B of dimension $m \times n$. Players may randomize amongst their strategies. The set of mixed strategies are Δ_m and Δ_n respectively.

▶ **Definition 5.** Nash Equilibrium [36] A strategy profile is said to be a Nash equilibrium (NE) if no player can achieve a better payoff by a unilateral deviation [25]. Formally, $(\boldsymbol{x}, \boldsymbol{y}) \in \Delta_m \times \Delta_n$ is a NE iff $\forall \boldsymbol{x}' \in \Delta_m,\ \boldsymbol{x}^T A \boldsymbol{y} \geq \boldsymbol{x}'^T A \boldsymbol{y}$ and $\forall \boldsymbol{y}' \in \Delta_n,\ \boldsymbol{x}^T B \boldsymbol{y} \geq \boldsymbol{x}^T B \boldsymbol{y}'$.

Game (A, B) is said to be symmetric if $B = A^T$. In a symmetric game the strategy sets of both players are identical, *i.e.,* $m = n$, and $S_1 = S_2$. We will use n, S and Δ_n to denote the strategy related quantities. Strategy $\boldsymbol{x} \in \Delta_n$ is a symmetric NE, both play $\boldsymbol{x}$, iff

$$\forall i \in S, x_i > 0 \Rightarrow (A\boldsymbol{x})_i = \max_k (A\boldsymbol{x})_k \tag{3}$$

▶ **Definition 6.** NE $\boldsymbol{x}$ is **strict** if $\forall k \notin SP(\boldsymbol{x}),\ (A\boldsymbol{x})_k < (A\boldsymbol{x})_i$, where $i \in SP(\boldsymbol{x})$.

Symmetric Coordination Game. In a coordination/partnership game $B = A$, *i.e.,* both the players get the same payoff always. Thus if A is symmetric then (A, A) is a symmetric coordination game. The next lemma follows using (3) and Fact 4

▶ **Lemma 7.** *If $\boldsymbol{x}$ is a symmetric NE of game (A, A), it is a fixed point of dynamics (2).*

From now on, by *mixed* (fixed point) strategy we mean strictly mixed (fixed point) strategy, *i.e.,* $\boldsymbol{x}$ such that $|SP(\boldsymbol{x})| > 1$, and non-mixed are called *pure.*

4.3 Basics in Dynamical Systems

Next we describe well-known stability notions in dynamical systems.

▶ **Definition 8.** A fixed point $\mathbf{r}$ of $f : \mathbb{R}^n \to \mathbb{R}^n$ is called **stable** if, for every $\epsilon > 0$, there exists a $\delta = \delta(\epsilon) > 0$ such that, for all $\boldsymbol{p} \in \mathbb{R}^n$ with $||\boldsymbol{p} - \mathbf{r}||| < \delta$ we have that $||f^n(\boldsymbol{p}) - \mathbf{r}|| < \epsilon$ for every $n \geq 0$, otherwise it is called **unstable**.

▶ **Definition 9.** A fixed point $\mathbf{r}$ of $f : \mathbb{R}^n \to \mathbb{R}^n$ is called **asymptotically stable** if it is stable and there exists a (neighborhood) $\delta > 0$ such that, for all $\boldsymbol{p} \in \mathbb{R}^n$ with $||\boldsymbol{p} - \mathbf{r}|| < \delta$ we have that $||f^n(\boldsymbol{p}) - \mathbf{r}|| \to 0$ as $n \to \infty$.

By definition it follows that if $\boldsymbol{x}$ is asymptotically stable w.r.t. dynamics f (2), then the set of initial conditions in Δ so that the dynamics converge to $\boldsymbol{x}$ has positive measure. Using that under f the potential function $\pi(\boldsymbol{x}) = \boldsymbol{x}^T A \boldsymbol{x}$ strictly decreases unless $\boldsymbol{x}$ is a fixed point, the next theorem was derived in [22].

▶ **Theorem 10** ([22], § 9.4.7)**.** *A fixed point $\boldsymbol{r}$ of dynamics (2) is stable if and only if it is a local maximum of π, and is asymptotically stable if and only if it is a strict local maximum.*

As the domain of π is closed and bounded, there exists a global maximum of π in Δ_n, which by Theorem 10 is a stable fixed point, and therefore its existence follows. However, existence of asymptotically stable fixed point is not guaranteed, for example if $A = [1]_{m\times n}$ then no $\boldsymbol{x} \in \Delta_n$ is attracting under f.

4.3.1 Stability and Eigenvalues

To analyze limiting points of f with respect to the notion of stability in terms of perturbation resistent, we need to use the eigenvalues of the Jacobian of f at fixed points. Let $J^{\mathbf{r}}$ denote the Jacobian at $\mathbf{r} \in \Delta_n$. The following theorem in dynamics/control theory relates (asymptotically) stable fixed points with the eigenvalue of its Jacobian.

▶ **Theorem 11** ([30])**.** *At fixed point $\boldsymbol{x}$ if $J^{\boldsymbol{x}}$ has at least one eigenvalue with absolute value > 1, then $\boldsymbol{x}$ is unstable. If all the eigenvalues have absolute value < 1 then it is asymptotically stable.*

▶ **Definition 12.** A fixed point $\mathbf{r}$ is called linearly stable, if the eigenvalues $J^{\mathbf{r}}$ are at most 1 in absolute value. Otherwise, it is called linearly unstable.

Theorem 11 implies that eigenvalues of the Jacobian at a stable fixed point have absolute value at most 1, however the converse may not hold. Using properties of $J^{\boldsymbol{x}}$ and [21], we prove next: (see the full version for Jacobian equations).

▶ **Theorem 13.** *The set of initial conditions in Δ_n so that the dynamics (2) converge to linearly unstable fixed points has measure zero.*

In Theorem 13 we manage to discard only those fixed points whose Jacobian has eigenvalue with absolute value > 1, while characterizing limiting points of f; the latter is finally used to argue about the survival of diversity.

5 Convergence, Stability, and Characterization

As established in Section 4.1, evolution in single locus diploid species is governed by dynamics f of (2). Understanding survival of diversity requires to analyze the following set,

$$\mathcal{L} = \{\boldsymbol{x} \in \Delta_n \mid \text{positive measure of starting points converge to } \boldsymbol{x} \text{ under } f\} \tag{4}$$

The next lemma follows from Definition 9 and Theorem 13.

▶ **Lemma 14.** *asymptotically stable $\subseteq \mathcal{L} \subseteq$ linearly stable.*

In this section we try to characterize $\mathcal{L}$ using various notions of stability, which have game theoretic and combinatorial interpretation. These notions sandwich set-wise the classical notions of stability given in Section 4.3, and thereby give us a partial characterization of $\mathcal{L}$. This characterization turns out to be crucial for our hardness results as well as results on survival in random instances.

Given a symmetric matrix A, a two-player game (A, A) forms a symmetric coordination game. In this section we identify special symmetric NE of this game to characterize stable fixed points of f. Given a profile $\boldsymbol{x} \in \Delta_n$, define a transformed matrix $T(A, \boldsymbol{x})$ of dimension $(k-1) \times (k-1)$, where $k = |SP(\boldsymbol{x})|$, as follows.

$$\text{Let } SP(\boldsymbol{x}) = \{i_1, \dots, i_k\},\ B = T(A, \boldsymbol{x}).\ \forall a, b < k,\ B_{ab} = A_{i_a i_b} + A_{i_k i_k} - A_{i_a i_k} - A_{i_k i_b} \quad (5)$$

Since A is symmetric it is easy to check that B is also symmetric, and therefore has real eigenvalues. Recall the Definition 6 of strict symmetric NE.

▶ **Definition 15.** A strategy $\boldsymbol{x}$ is called (strict) Nash stable if it is a (strict) symmetric NE of the game (A, A), and $T(A, \boldsymbol{x})$ is negative (definite) semi-definite.

▶ **Lemma 16.** *For any given $\boldsymbol{x} \in \Delta_n$, $T(A, \boldsymbol{x})$ is negative (definite) semi-definite iff ($\boldsymbol{y}^T A \boldsymbol{y} < 0$) $\boldsymbol{y}^T A \boldsymbol{y} \leq 0$, $\forall y \in \mathbb{R}^n$ such that $\sum_i y_i = 0$ and $x_i = 0 \Rightarrow y_i = 0$.*

Using that stable fixed point are local optima, we map them to Nash stable strategies.

▶ **Lemma 17.** *Every stable fixed point $\boldsymbol{r}$ of f is a Nash stable of game (A, A).*

Since stable fixed points always exist, so do Nash stable strategies (Lemma 17). Next we map strict Nash stable strategies to asymptotically stable fixed points, as the negative definiteness and strict symmetric Nash of the former implies strict local optima, and the next lemma follows.

▶ **Lemma 18** ([22] § 9.2.5)**.** *Every strict Nash stable is asymptotically stable.*

The above two lemmas show that strict Nash stable $\subseteq$ asymptotically stable (by definition) $\subseteq$ stable (by definition) $\subseteq$ Nash stable. Further, by Theorem 11 and the definition of *linearly stable* fixed points we know that stable $\subseteq$ linearly-stable. What remains is the relation between Nash stable and linearly stable. The next lemma answers this.

▶ **Lemma 19.** *Strategy $\boldsymbol{r}$ is Nash stable iff it is a linearly stable fixed point.*

Using Theorems 10 and 13, and Lemmas 14, 17, 18 and 19 we get the following characterization among all the notions of stability that we have discussed so far.

▶ **Theorem 20.** *Given a symmetric matrix A, we have*

$$\begin{array}{ccccccccc} \textit{Strict Nash stable} & \subseteq & \textit{Asymptotically stable} & \subseteq & \mathcal{L} & \subseteq & \textit{linearly-stable} & = & \textit{Nash stable} \\ & & \cap\!\shortmid & & & & & & \\ & & \textit{stable} \subseteq & \textit{linearly-stable} & = & \textit{Nash stable} & & & \end{array}$$

As stated before, generically (random fitness matrix) we have hyperbolic fixed points and all the previous notions coincide.

6 Survival of Diversity

In this section we characterize two extreme cases of fitness matrix for the survival of diversity, namely where diversity always survives and where diversity disappears regardless of the starting population. Using this characterization we analyze the chances of survival of diversity when fitness matrix and starting populations are picked uniformly at random from continuous distributions.

Given a fitness (positive) matrix A, let $\boldsymbol{x}$ be a limit point of dynamics f governed by (2) If it is not pure, *i.e.,* $|SP(\boldsymbol{x})| > 1$ then at least two alleles survive among the population, and we say the population is diverse in the limit.

▶ **Definition 21.** We say that *diversity survives* in the limit if there exists $\boldsymbol{x} \in \mathcal{L}$ such that $\boldsymbol{x}$ is not pure. And *diversity survives surely* if no $\boldsymbol{x} \in \mathcal{L}$ is pure.

Since $\mathcal{L} \subseteq$ Nash stable = linearly stable (Theorem 20), there has to be at least one mixed Nash (or linearly) stable strategy for diversity to survive.

Next we give a definition that captures the homozygote/heterozygote advantage and a lemma which uses it to identify instances that lack mixed Nash stable strategies.

▶ **Definition 22.** Diagonal entry A_{ii} is called *dominated* if and only if $\exists j$, such that $A_{ij} > A_{ii}$. And it is called *dominating* if and only if $A_{ii} > A_{ij}$ for all $j \neq i$.

In full version we show that diversity dies for matrices with all *dominating* diagonals. Next we show sure survival of diversity when diagonals are *dominated*.

▶ **Lemma 23.** *Let $\boldsymbol{r}$ be a fixed point of f with $r_t = 1$. If A_{tt} is dominated, then $\boldsymbol{r}$ is linearly unstable.*

If all pure fixed points are linearly unstable, then all linearly stable fixed points are mixed, and thus the next theorem follows using Theorem 20 and Lemma 23.

▶ **Theorem 24.** *If every diagonal of A is dominated then no $\boldsymbol{x} \in \mathcal{L}$ is pure, i.e., diversity survives almost surely.*

The following lemma shows that when the entries of a fitness matrix are picked uniformly independently from a continuous distribution, there is a positive probability (bounded away from zero for all n) so that every diagonal in A is dominated. This essentially means that generically, diversity survives with positive probability, bounded away from zero, where the randomness is taken with respect to both the payoff matrix and initial conditions.

▶ **Lemma 25.** *Let entries of A be chosen i.i.d from a continuous distribution. The probability that all diagonals of A are dominated is at least $\frac{1}{3} - o(1)$.*

The next theorem follows using Theorem 24 and Lemma 25.

▶ **Theorem 26.** *Assume that the fitness matrix has entries picked independently from a continuous distribution then with probability, at least $\frac{1}{3}$, diversity will survive almost surely.*

▶ Remark 27 (Typical instance). Observe that letting X_i be the indicator random variable that A_{ii} is dominating and $X = \sum_i X_i$ we get that $E[X] = \sum_i E[X_i] = \sum_i \Pr[E_i] = n \times \frac{1}{n} = 1$ so in expectation we will have one dominating element. Also from the above proof of Lemma 25 we get that $E[X^2] = \sum_i E[X_i] + 2\sum_{i<j} E[X_iX_j] = 1 + n(n-1)\Pr[E_i \cap E_j] \approx 2 - o(1)$ (namely $Var[X] \approx 1 - o(1)$) so by Chebyshev's inequality $\Pr[|X-1| > k]$ is $O(\frac{1}{k^2})$.

7 NP-Hardness Results

Positive chance of survival of phenotypic (allele) diversity in the limit under the evolutionary pressure of selection (dynamics (2)), implies existence of a mixed linearly stable fixed point (Theorem 13). This notion encompasses all the other notions of stability (Theorem 20), and may contain points that are not attracting. Whereas, strict Nash stable and asymptotically stable are attracting.

In this section we show that checking if there exists a mixed stable profile, for any of the five notions of stability (Definitions 8, 9, 12 and 15), may not be easy. In particular, we show that the problem of checking if there exists a mixed profile that satisfies any of the stability

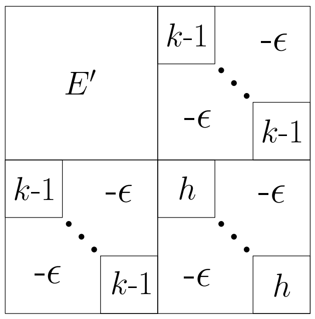


Figure 1 Matrix A as defined in (6), where E' is modification of E where off-diagonal zeros are replaced with $-h$ where h is a large (polynomial-size number).

conditions is NP-hard. In order to obtain hardness for checking survival of diversity, in other words checking if set $\mathcal{L}$ has a mixed strategy, we need to obtain a unifying reduction.

Our reduction also gives NP-hardness for checking if a given pure strategy is played with non-zero probability (subset) at these. In other words, it is NP-hard to check if a particular allele is going to survive in the limit under the evolution. Finally we extend all the results to the typical class of matrices, where exactly one diagonal entry is dominating (see Remark 27). All the reductions are from **k-Clique**, a well known NP-complete problem [6].

▶ **Definition 28. (k-Clique)** Given an undirected graph $G = (V, E)$, with V vertices and E edges, and an integer $0 < k < |V| - 1 = n - 1$, decide if G has a clique of size k.

Properties of G. Given a simple graph $G = (V, E)$ if we create a new graph $\bar{G}$ by adding a vertex u and connecting it to all the vertices $v \in V$, then it is easy to see that graph G has a clique of size k if and only if $\bar{G}$ has a clique of size $k + 1$. Therefore, w.l.o.g we can assume that there exists a vertex in G which is connected to all the other vertices. Further, if $n = |V|$, then for us such a vertex is the n^{th} vertex. By abuse of notation we will use E an adjacency matrix of $\bar{G}$ too, $E_{ij} = 1$ if edge (i, j) present in $\bar{G}$ else it is zero.

7.1 Hardness for checking stability

In this section we show NP-hardness (completeness for some) results for decision versions on (strict) Nash stable strategies and (asymptotically) stable fixed points. Given graph $G = (V, E)$ and integer $k < n$, we construct the following symmetric $2n \times 2n$ matrix A (see Figure 1), where E' is modification of E where off-diagonal zeros are replaced with $-h$ where $h > 2n^2 + 5$.

$$\forall i \leq j,\ A_{ij} = A_{ji} = \begin{cases} E'_{ij} & \text{if } i, j \leq n \\ k-1 & \text{if } |i-j| = n \\ h & \text{if } i, j > n \text{ and } i = j, \text{ where } h > 2n^2 + 5 \\ -\epsilon & \text{otherwise, where } 0 < \epsilon \leq \frac{1}{10n^3} \end{cases} \tag{6}$$

A is a symmetric but is not non-negative. The next lemma maps a k-clique to a mixed-strategy that is also strict Nash stable fixed point (FP). Note that such a FP satisfies all other stability notions as well, and hence implies existence of mixed limit point in $\mathcal{L}$.

▶ **Lemma 29.** *If there exists a clique of size at least k in graph G, then the game (A, A) has a mixed strategy $\mathbf{p}$ that is strict Nash stable.*

Since strict Nash stable is contained in all other sets, the above lemma implies existence of mixed-strategy for all of them if there is a clique in G. Next we want to show the converse for all notions of stability. That is if mixed-strategy exists for any notion of the five notions of stability then there is a clique of size at least k in the graph G. Since each of the five stability implies Nash stability, it suffices to map mixed Nash stable strategy to clique of size k. For this, and reductions that follow, we will use the following property due to negative semi-definiteness of Nash stability extensively.

▶ **Lemma 30.** *Given a fixed point $\boldsymbol{x}$, if $T(A,\boldsymbol{x})$ is negative semi-definite, then $\forall i \in SP(\boldsymbol{x}), A_{ii} \leq 2A_{ij},\ \forall j \neq i \in SP(\boldsymbol{x})$. Moreover if $\boldsymbol{x}$ is a mixed Nash stable then it has in its support at most one strategy t with A_{tt} is dominating.*

Nash stable also implies symmetric Nash equilibrium. The next lemma maps (special) symmetric NE to k-clique.

▶ **Lemma 31.** *Let $\boldsymbol{p}$ be a symmetric NE of game (A,A). If $SP(\boldsymbol{p}) \subset [n]$ and $|SP(\boldsymbol{p})| > 1$, then there exists a clique of size k in graph G.*

We obtain the following lemma essentially using Lemmas 30 and 31.

▶ **Lemma 32.** *If game (A,A) has a mixed Nash stable strategy, then graph G has a clique of size k.*

Note that adding a constant to A does not change its set of strict Nash stable and Nash stable.

▶ **Lemma 33.** *Let A be a symmetric matrix, and $B = A + c$ for a $c \in \mathbb{R}$, then the set of (strict) Nash stable strategies of B are identical to that of A.*

The next theorem follows using Theorem 20, Lemmas 29 and 32, and the property observed in Lemma 33. Since there is no polynomial-time checkable condition for (asymptotically) stable fixed points[11] its containment in NP is not clear, while for (strict) Nash stable strategies containment in NP follows from the Definition 15.

▶ **Theorem 34.** *Given a symmetric matrix A,*

- *it is NP-complete to check if game (A,A) has a mixed Nash stable (or linearly stable) strategy.*
- *it is NP-complete to check if game (A,A) has a mixed strict Nash stable strategy.*
- *it is NP-hard to check if dynamics (2) applied on A has a mixed stable fixed-point.*
- *it is NP-hard to check if dynamics (2) applied on A has a mixed asymptotically stable fixed-point.*

even if A is a assumed to be positive.

As we note in Remark 27, matrix with i.i.d entries from any continuous distribution has in expectation exactly one row with dominating diagonal. One could ask does the problem become easier for this typical case. We answer negatively by extending all the NP-hardness results of Theorem 34 to this case as well. Consider the following modification of matrix A

[11] These are same as (strict) local optima of function $\pi(\boldsymbol{x}) = \boldsymbol{x}^T A\boldsymbol{x}$, and checking if a given $\boldsymbol{p}$ is a local optima can be inconclusive if hessian at $\boldsymbol{p}$ is (negative) semi-definite.

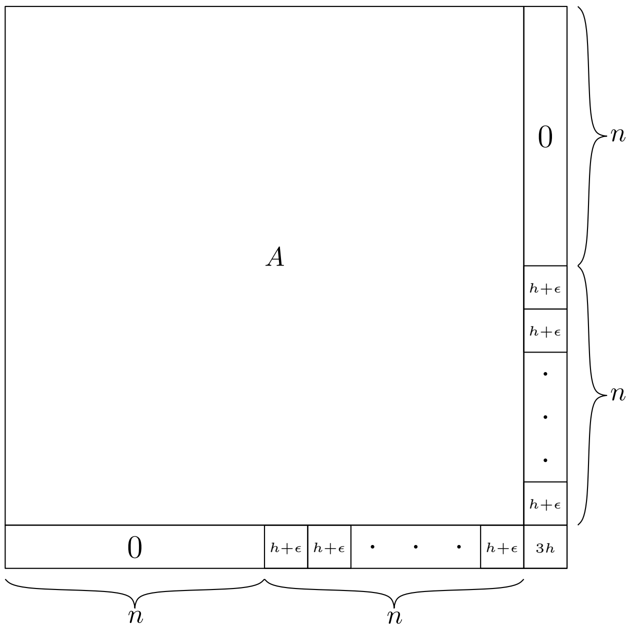


Figure 2 Matrix M as defined in (7).

from (6), where we add an extra row and column. Matrix M is of dimension $(2n+1)\times(2n+1)$ (See Figure (2)). Recall that $h > 2n^2 + 5$ and k is the given integer.

$$\begin{array}{ll} M_{ij} = A_{ij} & \text{if } i,j \le 2n \\ M_{(2n+1)i} = M_{i(2n+1)} = 0 & \text{if } i \le n \\ M_{(2n+1)i} = M_{i(2n+1)} = h+\epsilon & \text{if } n < i \le 2n, \text{ where } 0 < \epsilon < 1 \\ M_{(2n+1)(2n+1)} = 3h & \end{array} \tag{7}$$

Note that M has exactly one row whose diagonal entry dominates all other entries of the row, *i.e.*, $\exists i: \ M_{ii} > M_{ij}, \ \forall j \neq i$. See the full version of paper for details, and thus the next theorem holds.

▶ **Theorem 35.** *Given a symmetric matrix M such that exactly one row/column in M has a dominating diagonal,*

- *it is NP-complete to check if game (M,M) has a mixed Nash stable (or linearly stable) strategy.*
- *it is NP-complete to check if game (M,M) has a mixed strict Nash stable strategy.*
- *it is NP-hard to check if dynamics (2) applied on M has a mixed stable fixed-point.*
- *it is NP-hard to check if dynamics (2) applied on M has a mixed asymptotically stable fixed-point.*

even if M is a assumed to be positive.

Hardness for Subset. Another natural question is whether a particular allele is going to survive with positive probability in the limit for a given fitness matrix. In full version we show that this may not be easy either, by proving hardness for checking if there exists a stable strategy $\boldsymbol{p}$ such that $i \in SP(\boldsymbol{p})$ for a given i. In general, given a subset S of pure strategies it is hard to check if $\exists$ a stable profile $\boldsymbol{p}$ with $S \subseteq SP(\boldsymbol{p})$.

Survival of Diversity and Hardness. As discussed in Section 5 checking if diversity survives in the limiting population of single locus diploid organism reduces to checking "if f converges to a *mixed* fixed point with positive probability". In absence of clear characterization of the *mixed* limit points of f in terms of any of the stability definition, the hardness does not follow directly from the above result. In full version we explain how above results can be combined to obtain the following theorem. Also see the following remark on complexity of decision problem for general Nash equilibrium in coordination games.

▶ **Theorem 36.** *Given a fitness matrix A for a diploid organism with single locus, it is NP-hard to decide if, under evolution, diversity will survive (by converging to a specific mixed equilibrium with positive probability) when starting allele frequencies are picked i.i.d from from a continuous distribution. Also, deciding if a given allele will survive is NP-hard.*

▶ Remark 37. As noted in Section 4.2, coordination games are very special and they always have a pure Nash equilibrium which is easy to find; NE computation in general game is PPAD-complete [7]. Thus, it is natural to wonder if decision versions on coordination games are also easy to answer.

In the process of obtaining the above hardness results, we stumbled upon NP-hardness for checking if a symmetric coordination game has a NE (not necessarily symmetric) where each player randomizes among at least k strategies. Again the reduction is from k-clique. Thus, it seems highly probable that other decision version on (symmetric) coordination games are also NP-complete.

Acknowledgements. Part of this work was completed while Ruta Mehta, Ioannis Panageas and Georgios Piliouras were hosted by the Simons Institute for the Theory of Computing.

References

1 E. Chastain, A. Livnat, C. H. Papadimitriou, and U. Vazirani. Algorithms, games, and evolution. *PNAS*, 2014. `doi:10.1073/pnas.1406556111`.

2 E. Chastain, A. Livnat, C. H. Papadimitriou, and U. V. Vazirani. Multiplicative updates in coordination games and the theory of evolution. In *ITCS*, pages 57–58, 2013. `doi:10.1145/2422436.2422444`.

3 X. Chen, X. Deng, and S.-H. Teng. Settling the complexity of computing two-player Nash equilibria. *Journal of the ACM*, 56(3), 2009.

4 V. Conitzer. The exact computational complexity of evolutionarily stable strategies. In *The 9th Conference on Web and Internet Economics (WINE)*, 2013.

5 V. Conitzer and T. Sandholm. New complexity results about Nash equilibria. *Games and Economic Behavior*, 63(2):621–641, 2008.

6 T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

7 C. Daskalakis, P. W. Goldberg, and C. H. Papadimitriou. The complexity of computing a Nash equilibrium. *SIAM Journal on Computing*, 39(1):195–259, 2009.

8 K. Etessami and A. Lochbihler. The computational complexity of evolutionarily stable strategies. *International Journal of Game Theory*, 37(1):93–113, 2008.

9 R. A. Fisher. *The Genetical Theory of Natural Selection*. Clarendon Press, Oxford, 1930.

10 J. Garg, R. Mehta, V. V. Vazirani, and S. Yazdanbod. Etr-completeness for decision versions of multi-player (symmetric) nash equilibria. In *ICALP*, 2015.

11 I. Gilboa and E. Zemel. Nash and correlated equilibria: Some complexity considerations. *Games Econ. Behav.*, 1:80–93, 1989.

12 J. Hofbauer and K. Sigmund. *Evolutionary Games and Population Dynamics*. Cambridge University Press, Cambridge, 1998.

13 V. I. Istratescu. *Fixed Point Theory: An Introduction*. Mathematics and Its Applications. Springer Netherlands, 2001. URL: `http://books.google.com/books?id=2bsDnwEACAAJ`.

14 H. Kalmus. Adaptive and selective responses of a population of drosophila melanogaster containing e and e+ to differences in temperature, humidity, and to selection for development speed. *Journal of Genetics*, 47:58–63, 1945.

15 A. Kawamura. Lipschitz continuous ordinary differential equations are polynomial-space complete. *Computational Complexity*, 19(2):305–332, 2010.

16 A. Kawamura, H. Ota, C. R osnick, and M. Ziegler. Computational complexity of smooth differential equations. In *In Mathematical Foundations of Computer Science*, pages 578–589, 2012.

17 A. Kaznatcheev. Complexity of evolutionary equilibria in static fitness landscapes. *arXiv preprint arXiv:1308.5094*, 2013.

18 R. Kleinberg, K. Ligett, G. Piliouras, and É. Tardos. Beyond the Nash equilibrium barrier. In *Symposium on Innovations in Computer Science (ICS)*, 2011.

19 R. Kleinberg, G. Piliouras, and É. Tardos. Multiplicative updates outperform generic no-regret learning in congestion games. In *STOC*, 2009.

20 A. Livnat, C. H. Papadimitriou, J. Dushoff, and M. W. Feldman. A mixability theory for the role of sex in evolution. *PNAS*, 2008. `doi:10.1073/pnas.0803596105`.

21 V. Losert and E. Akin. Dynamics of games and genes: Discrete versus continuous time. *Journal of Mathematical Biology*, 1983.

22 Y. Lyubich. *Mathematical Structures in Population Genetics*. Springer-Verlag, 1992.

23 R. Mehta, I. Panageas, and G. Piliouras. Natural selection as an inhibitor of genetic diversity: Multiplicative weights updates algorithm and a conjecture of haploid genetics. In *ITCS*, 2015.

24 R. Meir and D. Parkes. A note on sex, evolution, and the multiplicative updates algorithm. In *Proceedings of the 12th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 15)*, 2015.

25 J. Nash. Equilibrium points in n-person games. *PNAS*, pages 48–49, 1950.

26 N. Nisan. A note on the computational hardness of evolutionary stable strategies. *Electronic Colloquium on Computational Complexity (ECCC)*, 13(076), 2006.

27 I. Panageas, P. Srivastava, and N. K. Vishnoi. Evolutionary dynamics in finite populations mix rapidly. In *Proc. of the 27th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA'16)*, pages 480–497, 2016. `doi:10.1137/1.9781611974331.ch36`.

28 C. Papadimitriou and G. Piliouras. From nash equilibria to chain recurrent sets: Solution concepts and topology. In *Proc. of the 2016 ACM Conf. on Innovations in Theoretical Computer Science*, ITCS'16, pages 227–235, New York, NY, USA, 2016. ACM. `doi:10.1145/2840728.2840757`.

29 C. H. Papadimitriou and N. K. Vishnoi. On the computational complexity of limit cycles in dynamical systems. In *ITCS*, 2016.

30 L. Perko. *Differential Equations and Dynamical Systems*. Springer, 1991.

31 G. Piliouras, C. Nieto-Granda, H. I. Christensen, and J. S. Shamma. Persistent patterns: Multi-agent learning beyond equilibrium and utility. In *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS'14, pages 181–188, Richland, SC, 2014. International Foundation for Autonomous Agents and Multiagent Systems.

32 G. Piliouras and J. S. Shamma. Optimization despite chaos: Convex relaxations to complex limit sets via Poincaré recurrence. In *SODA*, 2014.

33 K. C. Rasmus Ibsen-Jensena and M. A. Nowak. Computational complexity of ecological and evolutionary spatial dynamics. In *PNAS*, 2015.

34 M. Schaefer and D. Štefankovič. Fixed points, Nash equilibria, and the existential theory of the reals. Manuscript, 2011.
35 S.-M. Sun and N. Zhong. Computability aspects for 1st-order partial differential equations via characteristics. *Theoretical Computer Science*, 583:27–39, 2015.
36 B. von Stengel. Equilibrium computation for two-player games in strategic and extensive form. *Algorithmic Game Theory, eds. Nisan, Roughgarden, Tardos, and Vazirani*, 2007.
37 E. D. Weinberger. Np completeness of kauffman's nk model, a tuneable rugged fitness landscape. Technical report, Santa Fe Institute, 1996.
38 Wikipedia. Sylvester's law of inertia. URL: `https://en.wikipedia.org/wiki/Sylvester's_law_of_inertia`.
39 A. H. Wright, R. K. Thompson, and J. Zhang. The computational complexity of nk fitness functions. *IEEE Transactions on Evolutionary Computation*, 4(4):373–379, 2000.

A Terms Used in Biology

A.1 Terms in Biology

We provide brief non-technical definitions of a few biological terms that we use in this paper.

Gene. A unit that determines some characteristic of the organism, and passes traits to offsprings. All organisms have genes corresponding to various biological traits, some of which are instantly visible, such as eye color or number of limbs, and some of which are not, such as blood type.

Allele. Allele is one of a number of alternative forms of the same gene, found at the same place on a chromosome. Different alleles can result in different observable traits, such as different pigmentation.

Genotype. The genetic constitution of an individual organism.

Phenotype. The set of observable characteristics of an individual resulting from the interaction of its genotype with the environment.

Diploid. Diploid means having two copies of each chromosome. Almost all of the cells in the human body are diploid.

Haploid. A cell or nucleus having a single set of unpaired chromosomes. Our sex cells (sperm and eggs) are haploid cells that are produced by meiosis. When sex cells unite during fertilization, the haploid cells become a diploid cell.

A.2 Heterozygote Advantage (Overdominance)

Cases of heterozygote advantage have been demonstrated in several organisms. The first confirmation of heterozygote advantage was with a fruit fly, Drosophila melanogaster. Kalmus demonstrated in a classic paper [14] how polymorphism can persist in a population through heterozygote advantage. In humans, sickle-cell anemia is a genetic disorder caused by the presence of two recessive alleles. Where malaria is common, carrying a single sickle-cell allele (trait) confers a selective advantage, *i.e.,* being a heterozygote is advantageous. Specifically, humans with one of the two alleles of sickle-cell disease exhibit less severe symptoms when infected with malaria.

Approximation and Hardness of Token Swapping

Tillmann Miltzow*[1], Lothar Narins[2], Yoshio Okamoto†[3], Günter Rote[4], Antonis Thomas[5], and Takeaki Uno[6]

1 **Freie Universität Berlin, Berlin, Germany**
2 **Freie Universität Berlin, Berlin, Germany**
3 **University of Electro-Communications, Tokyo, Japan**
4 **Freie Universität Berlin, Berlin, Germany**
5 **Department of Computer Science, ETH Zürich, Zürich, Switzerland**
6 **National Institute of Informatics, Tokyo, Japan**

Abstract

Given a graph $G = (V, E)$ with $V = \{1, \ldots, n\}$, we place on every vertex a token $T_1, \ldots, T_n$. A swap is an exchange of tokens on adjacent vertices. We consider the algorithmic question of finding a shortest sequence of swaps such that token T_i is on vertex i. We are able to achieve essentially matching upper and lower bounds, for exact algorithms and approximation algorithms. For exact algorithms, we rule out any $2^{o(n)}$ algorithm under the ETH. This is matched with a simple $2^{O(n \log n)}$ algorithm based on a breadth-first search in an auxiliary graph. We show one general 4-approximation and show APX-hardness. Thus, there is a small constant $\delta > 1$ such that every polynomial time approximation algorithm has approximation factor at least δ.

Our results also hold for a generalized version, where tokens and vertices are colored. In this generalized version each token must go to a vertex with the same color.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases token swapping, minimum generator sequence, graph theory, NP-hardness, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.66

1 Introduction

In the theory of computation, we regularly encounter the following type of problem: Given two configurations, we wish to transform one to the other. In these problems we also need to fix a family of operations that we are allowed to perform. Then, we need to solve two problems: (1) Determine if one can be transformed to the other; (2) If so, find a shortest sequence of such operations. Motivations come from the better understanding of solution spaces, which is beneficial for design of local-search algorithms, enumeration, and probabilistic analysis. See [9] for example. The study of *combinatorial reconfigurations* is a young growing field [2].

Among problem variants in combinatorial reconfiguration, we study the *token swapping problem* on a graph. The problem is defined as follows. We are given an undirected connected graph with n vertices $v_1, \ldots, v_n$. Each vertex v_i holds exactly one token $T_{\pi(i)}$, where π is a permutation of $\{1, \ldots, n\}$. In one step, we are allowed to swap tokens on a pair of adjacent

* Partially supported by the ERC grant PARAMTIGHT: "Parameterized complexity and the search for tight complexity results", no. 280152.
† Partially supported by MEXT/JSPS KAKENHI Grant Numbers 24106005, 24700008, 24220003 and 15K00009, and JST, CREST, Foundation of Innovative Algorithms for Big Data.

vertices, that is, if v_i and v_j are adjacent, v_i holds the token T_k, and v_j holds the token T_ℓ, then the swap between v_i and v_j results in the configuration where v_i holds T_ℓ, v_j holds T_k, and all the other tokens stay in place. Our objective is to determine the minimum number of swaps so that every vertex v_i holds the token T_i. It is known (and not difficult to observe) that such a sequence of swaps always exists [22].

The problem was introduced by Yamanaka et al. [22] in its full generality, but special cases had been studied before. When the graph is a path, the problem is equivalent to counting the number of adjacent swaps in bubble sort, and it is folklore that this is exactly the number of inversions of π (see Knuth [16, Section 5.2.2]). When the graph is complete, it was already known by Cayley [4] that the minimum number is equal to n minus the number of cycles in π (see also [14]). Note that the number of inversions and the number of cycles can be computed in $O(n \log n)$ time. Thus, the minimum number of swaps can be computed in $O(n \log n)$ time for paths and complete graphs. Jerrum [14] gave an $O(n^2)$-time algorithm to solve the problem for cycles. When the graph is a star, a result by Pak [19] implies an $O(n \log n)$-time algorithm. Exact polynomial-time algorithms are also known for complete bipartite graphs [22] and complete split graphs [24]. Polynomial-time approximation algorithms are known for trees with factor two [22] and for squares of paths with factor two [12]. Since Yamanaka et al. [22], it has remained open whether the problem is polynomial-time solvable or NP-complete, even for general graphs, and whether there exists a constant-factor polynomial-time approximation algorithm for general graphs.

Our results. In this paper, we resolve the open problems above from Yamanaka et al. [22]. Namely, we prove that the token swapping problem is NP-complete, and give a polynomial-time approximation algorithm with factor 4 for general graphs and factor 2 for trees. For the NP-completeness, we consider a more general problem, called the *colored token swapping problem*. In the colored token swapping problem, each token has a color, each vertex of the graph has a color, and we are asked to move the tokens to the vertices of the same color. This can also be seen as a graph-theoretic generalization of bubble sort on a multiset. Yamanaka et al. [23] proved that the colored token swapping problem is NP-complete. We strengthen their result in the sense that our hardness results hold for instances with special structure. Furthermore we design a gadget called an *even permutation* network (not to be confused with a sorting network). This can be regarded as a special class of instances of the token swapping problem. It allows us to achieve *any* even permutation of the tokens between dedicated input and output vertices. Using even permutation networks allows us to reduce further from the colored token swapping problem to the more specific uncolored version. We believe that this gadget is of general interest and we hope that similar gadgets will prove useful in other situations too. A close look at the hardness proof gives APX-hardness. Therefore we have that the constant approximation, we provide, is essentially tight (up to the constant). In addition, the proof rules out any $2^{o(n)}$-time algorithm under the Exponential Time Hypothesis, where n is the number of vertices of the input graph. To complement this, we provide a simple $2^{O(n \log n)}$-time exact algorithm. Therefore, our algorithmic results are almost *tight*. Even though our exact algorithm is completely straightforward, our reductions show that we cannot hope for much better results. The approximation algorithm we suggest makes deep use of the structure of the problem and is nice to present. Our reductions are quite technical and, so, we try to modularize them as much as possible. In the process, we show hardness for a number of intermediate problems. Due to space limitations, some of the proofs are missing. A self-contained full version of this paper can be found online [17].

Organization. In Section 2, we provide a simple exact algorithm. Section 3 describes a 4-approximation algorithm of the token swapping problem. In Section 4, we show a general technique to attain approximation algorithms for the colored token swapping problem from the uncolored version. In this way, we attain a 4-approximation for the colored token swapping problem. In Section 5, we define and construct even permutation networks, which are interesting in their own right. The hardness results are modularized into four sections. In Section 6, we show a reduction from 3SAT to a problem of finding disjoint paths in a structured graph. (A precise definition can be found in the section.) Section 7 shows how to reduce further to the colored token swapping problem. Section 8 is committed to the reduction from the colored to the ordinary token swapping problem. For this purpose, we attach an even permutation network to each color class. Section 9 finally puts the three previous reductions together in order to attain the main results.

Related concepts. The famous 15-puzzle is similar to the token swapping problem, and indeed a graph-theoretic generalization of the 15-puzzle was studied by Wilson [21]. The 15-puzzle can be modeled as a token-swapping problem by regarding the empty square of the 4×4 grid as a distinguished token, but there remains an important difference from our problem: in the 15-puzzle, two adjacent tokens can be swapped only if one of them is the distinguished token. For the 15-puzzle and its generalization, the reachability question is not trivial, but by the result of Wilson [21] we can determine if the tokens can be moved to the right vertices in polynomial time. However, it is NP-hard to find the minimum number of swaps even for grids [20].

The token swapping problem can be seen as an instantiation of the minimum generator sequence problem of permutation groups. There, a permutation group is given by a set of generators $\pi_1, \dots, \pi_k$, and we want to find a shortest sequence of generators whose composition is equal to a given target permutation τ. Indeed, this is the problem that Jerrum [14] studied, where he gave an $O(n^2)$-time algorithm for the token swapping problem on cycles. He proved that the minimum generator sequence problem is PSPACE-complete [14]. To formulate the token swapping problem as the minimum generator sequence problem of permutation groups, for a given connected graph G, we consider the symmetric group on $\{1, \dots, n\}$ (the set of all permutations on $\{1, \dots, n\}$) that is given by the set of transpositions determined by the edges of G.

In the literature, we also find the problem of *token sliding*, but this is different from the token swapping problem. In the token sliding problem on a graph G, we are given two independent sets I_1 and I_2 of G of the same size. We place one token on each vertex of I_1, and we perfom a sequence of the following sliding operations: We may move a token on a vertex v to another vertex u if v and u are adjacent, u has no token, and after the movement the set of vertices with tokens forms an independent set of G. The goal is to determine if a sequence of sliding operations can move the tokens on I_1 to I_2. The problem was introduced by Hearn and Demaine [11], and they proved that the problem is PSPACE-complete even for planar graphs of maximum degree three. Subsequent research showed that the token sliding problem is PSPACE-complete for perfect graphs [15] and graphs of bounded treewidth [18]. Polynomial-time algorithms are known for cographs [15], claw-free graphs [3], trees [6] and bipartite permutation graphs [8].

Several other models of swapping have been studied in the literature, e.g. [10, 7, 5].

2 Simple Exact Algorithms

We start presenting our results with the simplest one. There is an exact, exponential time algorithm which, after the hardness results that we obtain in Section 9, will prove to be almost tight to the lower bounds.

▶ **Theorem 1.** *Let I be an instance of the token swapping problem on a graph $G = (V, E)$ with n vertices and m edges. There exists a simple algorithm for finding an optimal sequence in time $O(n!\ m) = 2^{O(n \log n)}$.*

Proof. The algorithm is breadth first search in the configuration graph. The vertices $\mathcal{V}$ of the configuration graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consist of all $n! = 2^{O(n \log n)}$ possible configurations of tokens on vertices V of G. Two configurations A and B are adjacent if there is a single swap that transforms A to B. Each configuration has $m \leq \binom{n}{2} = O(n^2)$ adjacent configurations. Thus the total number of edges is $n!\ m/2 \leq O(n^2) \cdot 2^{O(n \log n)} = 2^{O(n \log n)}$. It is easy to see that any shortest path in $\mathcal{G}$ from the start to the target configuration gives a shortest sequence of swaps. Breadth first search finds this shortest path. The running time of breadth first search is $O(|\mathcal{V}| + |\mathcal{E}|) = 2^{O(n \log n)}$. ◀

3 A 4-Approximation Algorithm

In this section, we describe an approximation algorithm for the token swapping problem, which has approximation factor 4 on general graphs and 2 on trees. Firstly, we state the following simple lemma.

▶ **Lemma 2.** *Let $d(T_i)$ be the distance of token T_i to the target vertex i. Let L be the sum of distances of all tokens to their target vertices:*

$$L := \sum_{i=1}^{n} d(T_i).$$

Then any solution needs at least $L/2$ swaps.

Proof. Every swap reduces L by at most 2. ◀

We are now ready to describe our algorithm. It has two atomic operations. The first one is called an *unhappy swap*: This is an edge swap where one of the tokens swapped is already on its target and the other token reduces its distance to its target vertex (by one).

The second operation is called a *happy swap chain.* Consider a path of $\ell + 1$ distinct vertices $v_1, \ldots, v_{\ell+1}$. We swap the tokens over edge (v_1, v_2), then (v_2, v_3), etc., performing ℓ swaps in total. The result is that the token that was on vertex v_1 is now on vertex $v_{\ell+1}$ and all other tokens have moved from v_j to v_{j-1}. If every swapped token reduces its distance by at least 1, we call this a happy swap chain of length ℓ. A single swap that is part of a happy swap chain is called a *happy swap.* When our algorithm applies a happy swap chain, there will be an edge between $v_{\ell+1}$ and v_1, closing a cycle. In this case, the happy swap chain performs a cyclic shift. Figure 1 illustrates this definition with an example. Note that a happy swap chain might consist of a single swap. We leave it to the reader to find such an example.

▶ **Lemma 3.** *Let G an undirected graph $G = (V, E)$ with a token placement where not every token is at its target vertex. Then, there is a happy swap chain or an unhappy swap.*

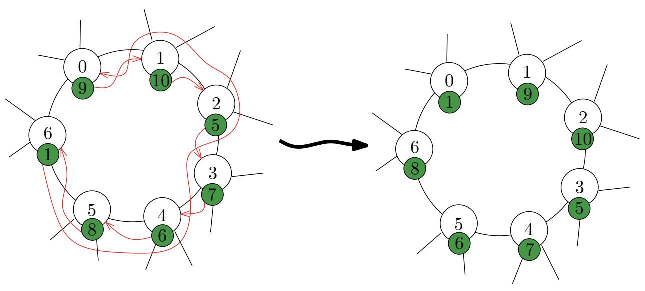


Figure 1 Before and after a happy swap chain. The swap sequence is, in this order, 6 – 5, 5 – 4, 4 – 3, 3 – 2, 2 – 1, 1 – 0. Token 1 swaps with every other token, moving counter-clockwise; every other token moves one step clockwise.

Proof. Given a token placement on a graph $G = (V, E)$ we define the auxiliary directed graph F on V as follows. For each undirected edge $e = \{v, w\}$ of G we define a directed edge from v to w if the token on v reduces its distance to its target vertex by swapping along e. Note that for a pair of vertices both directed edges might be part of the graph F. We can perform a happy swap chain whenever we find a directed cycle in F. The outdegree of a vertex v in F is 0 if and only if the token on v has target vertex v. Assume that not every token is in its target position. Choose any vertex v that does not hold the right token and perform a depth first search from v in F. This search ends with either revisiting a vertex and we get a directed cycle or we encounter a vertex with outdegree 0 and we get an unhappy swap. ◀

The above lemma gives rise to our algorithm: Search for a happy swap chain or unhappy swap; when one is found it is performed, until none remains. If there is no such swap, the final placement of every token is reached. This algorithm is polynomial time (follows from the proof of Lemma 3). Moreover, it correctly swaps the tokens to their target position with at most $2L$ swaps.

▶ **Lemma 4.** *Let T_i be a token on vertex i. If T_i participates in an unhappy swap, then the next swap involving T_i will be a happy swap.*

Proof. Refer to Figure 2. Let the vertices i, j be the ones participating in the unhappy swap and let e be the edge that connects them. On vertex j is token T_i that got unhappily removed from its target vertex and on vertex j is token S' whose target is neither i nor j. Based on Lemma 3, our algorithm performs either unhappy swaps or happy swap chains. Note that, currently, edge e cannot participate in an unhappy swap. This is because none of its endpoints holds the right token. Moreover, token T_i cannot participate in an unhappy swap that does not involve edge e, as that would not decrease its distance. Therefore, there has to be a happy swap chain that involves token T_i. ◀

▶ **Theorem 5.** *Any sequence of happy swap chains and unhappy swaps is at most 4 times as long as an optimal sequence of swaps on general graphs and 2 times as long on trees.*

Proof. Let L be the sum of all distances of tokens to its target vertex. We know that the optimal solution needs at least $L/2$ swaps as every swap reduces L by at most 2 (Lemma 2). We will show that our algorithm needs at most $2L$ swaps, and this implies the claim.

A happy swap chain of length ℓ reduces the sum of total distances by $\ell + 1$ and involves ℓ happy swaps. Thus, #(happy swaps) $< L$. By Lemma 4, #(unhappy swaps) $\leq$

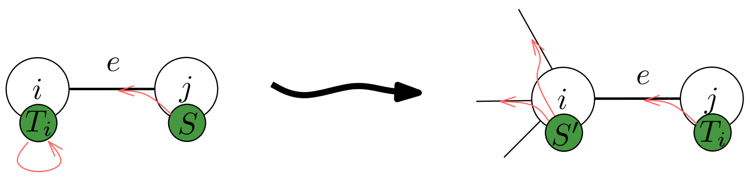


Figure 2 After a Token T_i makes an unhappy swap along edge e, T_i wants to go back to vertex i and is not willing to go to any other vertex. Also whatever token S' will be on vertex i, it has no desire to stay there. This implies that the next swap involving T_i will be part of a happy swap chain.

#(happy swaps), this implies: #(swaps) = #(unhappy swaps) + #(happy swaps) $\leq$ 2 #(happy swaps) $< 2L$. This algorithm is a 2-approximation algorithm on trees as the longest possible cycle in F (as in Lemma 3) has length 2 and thus every happy swap chain consists of only one happy swap that reduces L by *two*. This implies #(happy swaps) $= L/2$. ◀

4 The Colored Version of the Problem

In this section, we consider the *colored* token swapping problem as defined in the introduction. We will see that all approximation bounds from the previous sections carry over to this problem. Our approach to the problem is easy:

1. We first decide which token goes to which target vertex.
2. We then apply one of the algorithms from the previous sections for n distinct tokens.

The details can be found in the full version.

▶ **Theorem 6.** *There is a* 4*-approximation of the colored token swapping problem on general graphs and a* 2*-approximation on trees.*

5 Even Permutation Networks

Before we dive in the details of the reductions for token swapping (Sections 6–8), we introduce our even permutation network PN, the gadget that we advertised in the introduction. We consider it stand-alone and we present it independently of the other parts of the reduction.

The vertices of PN are partitioned into the sets V_{in}, V_{out}, and V_{PN}, where $|V_{\text{in}}| = |V_{\text{out}}|$, together with a bijection $\varphi\colon V_{\text{in}} \to V_{\text{out}}$ and a bijection $\psi\colon V_{\text{out}} \cup V_{\text{PN}} \to V_{\text{in}} \cup V_{\text{PN}}$, defining for each token on $V_{\text{PN}} \cup V_{\text{out}}$ a fixed target vertex in $V_{\text{in}} \cup V_{\text{PN}}$. We place on each $v \in V_{\text{out}} \cup V_{\text{PN}}$ the token $T_{\psi(v)}$. (The goal of token T_v is to be on vertex v.)

The target vertices of V_{in} lie in V_{out}, and they are specified by an additional variable permutation π of V_{in}. The instance $I(\pi)$ corresponding to π is obtained by placing on each $v \in V_{\text{in}}$ the token $T_{\varphi(\pi(v))}$.

This *even* permutation network has the property that the optimal number of swaps to solve $I(\pi)$ is the same for every even permutation π, see Figure 3. We will refer to the composition $\varphi \circ \pi$ as an *assignment*. Recall that a permutation is even if the number of inversions of the permutation is even. Realizing *all* permutations at the same cost is impossible, since every swap changes the parity, and hence all permutations reachable by a given number of swaps have the same parity.

We will later use even permutation networks to reduce from the *colored* token swapping problem on a network H to the token swapping problem, see Lemma 10. We will attach an even permutation network PN to each layer (color class) of H by identifying that layer with the input vertices V_{in} of PN. This permutation network will bring the colored tokens of H,

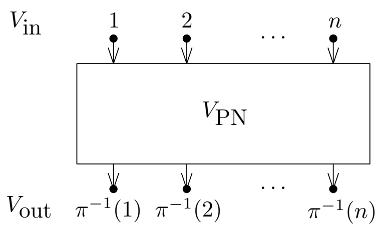


Figure 3 The interface of a permutation network PN. The permutation φ is not indicated; it maps each vertex of V_{in} to the vertex of V_{out} of that is vertically below.

which have arrived in V_{in} in a first phase, to their individual final destinations in V_{out}. To ensure that even permutations suffice, we use the simple trick of doubling the whole input graph before attaching the permutation networks: the product of two permutations of the same parity is always even.

▶ **Lemma 7.** *For every n, there is a permutation network* PN *with n input vertices* V_{in}*, n output vertices* V_{out}*, and $O(n^3)$ additional vertices* V_{PN}*, which has the following properties, for some value T:*

- *For every target assignment π between the inputs* V_{in} *and the outputs* V_{out} *that is an even permutation, the shortest swapping sequence has length T.*
- *For any other target assignment, the shortest realizing sequence has length at least $T+1$.*
- *The same statement holds for any extension of the network PN, which is the union of PN with another graph H that shares only the vertices* V_{in} *with PN, and in which the starting positions of the tokens assigned to* V_{out} *may be anywhere in H.*

The last clause concerns not only all assignments between V_{in} and V_{out} that are odd permutations, but also all other conceivable situations where the tokens destined for V_{out} do not end up in V_{in}, but somewhere else in the graph H. The lemma confirms that such non-optimal solutions for H cannot be combined with solutions for PN to yield better swapping sequences than the ones for which the network was designed.

Proof. (Sketch. For the complete proof with diagrams of the gadgets, see the full version.)

The even permutation network will be built up hierarchically from small gadgets. Each gadget is built in a layered manner, subject to the following rules.

1. There is a strict layer structure: The vertices are partitioned into layers $V_1, \dots, V_t$ of the same size.
2. Each gadget has its own input layer V_{in} at the top and its output layer V_{out} at the bottom, just as the overall network PN.
3. Edges may run between two vertices of the same layer (*horizontal edges*), or between adjacent layers (*downward edges*).
4. Every vertex has at most one neighbor in the successor layer and at most one neighbor in the predecessor layer.

The goal is to bring the input tokens from the input layer V_{in} to the output layer V_{out}. By Rule 3, the cheapest conceivable way to achieve this is by using only the downward edges, and then every such edge is used precisely once. Our first gadget is the so-called *swapping gadget*, which can either realize the identity permutation, or swap a specified input vertex with one of two others at a cost of one additional swap. Our second gadget, the *shift gadget*, consists of two swapping gadgets chained together in such a way that both the identity permutation and a cyclic shift of three input vertices can be realized in the same minimal number of swaps, but every other permutation takes more swaps. Both of these constructions involve some auxiliary input tokens that have set destinations within the gadget. We then chain these shifting gadgets together in a cascading fashion, so that any even permutation of the input vertices can be realized as a composition of the cyclic shifts of three vertices. In this way, all the even permutations can be achieved with the same amount of swaps, and all other permutations are more costly. ◀

6 Reduction to a Disjoint Paths Problem

For the lower bounds of the Token Swapping problem, we study some auxiliary problems. The first problem is a special multi-commodity flow problem.

Disjoint Paths on a Directed Acyclic Graph (DP)
Input: A directed acyclic graph $G = (V, E)$ and a bijection $\varphi\colon V^- \to V^+$ between the sources V^- (vertices without incoming arcs) and the sinks V^+ (vertices without outgoing arcs), with the following properties:

1. The vertices can be partitioned into layers $V_1, V_2, \ldots, V_t$ such that, for every vertex in some layer V_j, all incoming arcs (if any) come from the same layer V_i, with $i < j$. Note that i need not be the same for every vertex in V_j.
2. Every layer contains at most 10 vertices.
3. For every $v \in V^-$, there is a path from v to $\varphi(v)$ in G. Let $n(v)$ denote the number of vertices on the shortest path from v to $\varphi(v)$.
4. The total number of vertices is $|V| = \sum_{v \in V^-} n(v)$.

Question: Is there a set of vertex-disjoint directed paths $P_1, \ldots, P_k$ with $k = |V^-| = |V^+|$, such that P_i starts at some vertex $v \in V^-$ and ends at $\varphi(v)$?

By Property 4, the k paths must completely cover the vertices of the graph. The graphs that we will construct in our reduction have in fact the stronger property that *any* directed path from $v \in V^-$ to $\varphi(v)$ contains the same number $n(v)$ of vertices.

In our construction, we will label the source and the sink that should be connected by a path by the same symbol X, and "the path X" refers to this path. In the drawings, the arcs will be directed from top to bottom.

▶ **Lemma 8.** *There is a linear-size reduction from 3SAT to the Disjoint Paths Problem on a Directed Acyclic Graph (DP).*

Proof. Let $x_1, \ldots, x_n$ be the variables and $C_1, \ldots, C_m$ the clauses of the 3SAT formula. Each variable x_i is modeled by a variable path, which has a choice between two tracks. The track is determined by the choice of the first vertex on the path after x_i: either x_i^T or x_i^F . This choice models the truth assignment. There is also a path for each clause. In addition, there will be supplementary paths that fill the unused variable tracks. Figure 4 shows an example of a variable x_i that appears in three clauses C_u, C_v, and C_w. Consequently, the two tracks x_i^T and x_i^F, which run in parallel, pass through three *clause* gadgets, which are shown schematically as gray boxes in Figure 4 and which are drawn in greater detail in Figure 5. The bold path in Figure 4 corresponds to assigning the value *false* to x_i: the path follows the track x_i^F. For a variable that appears in ℓ clauses, there are $\ell + 1$ supplementary paths. In Figure 4, they are labeled $s_1, \ldots, s_4$. The path s_j covers the unused track (x_i^T in the example) between the $(j-1)$-st and the j-th clause in which the variable x_i is involved.

Initially, the path x_i can choose between the tracks x_i^T and x_i^F; the other track will be covered by a path starting at s_1. This choice is made possible by a *crossing* gadget. Each variable has two crossing gadgets attached, one at the beginning of the variable path and

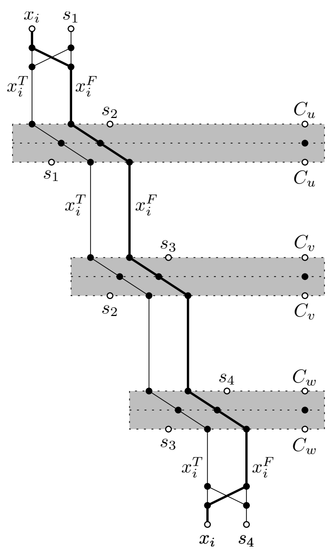


Figure 4 Schematic representation of a variable x_i. Sources and sinks are marked by white vertices, and their labels indicated the one-to-one correspondence φ between sources and sinks.

one at the end. Those gadgets consist of 6 vertices: x_i, s_1, x_i^T, x_i^F and two auxiliary vertices that allow the variable to change tracks. In Figure 4, the crossing gadgets appear at the top and the bottom. Note, that a variable can only change tracks in the crossing gadgets; the last supplementary path $s_{\ell+1}$ allows the path x_i to reach its target sink.

Figure 5 shows a clause gadget C_z in greater detail. It consists of three successive layers and connects the three variables that occur in the clause. The clause itself is represented by a clause path that spans only these three layers. A supplementary path starts at the first layer and one ends at the third layer of each clause gadget. Each layer of the clause gadget has at most 10 vertices: three for each variable, two that are on the tracks x_i^T, x_i^F, one for the supplementary path of the variable. There is at most three variables per clause. Finally, there is a vertex that corresponds to the clause itself.

Let C_z be the current clause which is the jth clause in which x_i appears, and it does so as a positive literal (as in Figure 5a). Each track, say x_i^T, connects to the corresponding vertex in the middle and bottom layer of the clause gadget and to s_j (the end of the supplementary path). The track of the literal that does not appear in this clause (x_i^F in this case) is also connected to the vertex of the clause on the middle layer (z in Figure 5). Moreover, the middle layer vertex of this track is connected to the top and bottom layer vertices that correspond to the clause. The supplementary path s_{j+1} starts in this clause gadget and goes through the middle layer and then connects to the vertices of both tracks on the bottom layer. Figure 5 depicts all these connections.

We can make the following observations about the interaction between the variable x_i and the clause C_z. We, further, illustrate those in Figure 5.

1. A variable path (shown in black) that enters on the track x_i^T or x_i^F must leave the gadget on the same track. Equivalently, a path cannot change track except in the crossing gadgets.
2. The clause path (in red) can make a detour through vertex v (w.r.t. Figure 5) only if the variable x_i makes the clause true, according to the track chosen by the variable path. In this case, the supplementary path s_j covers the intermediate vertex z of the clause.
3. If variable x_i makes the clause true, the clause path may also choose a different detour, in case more variables make the clause true.

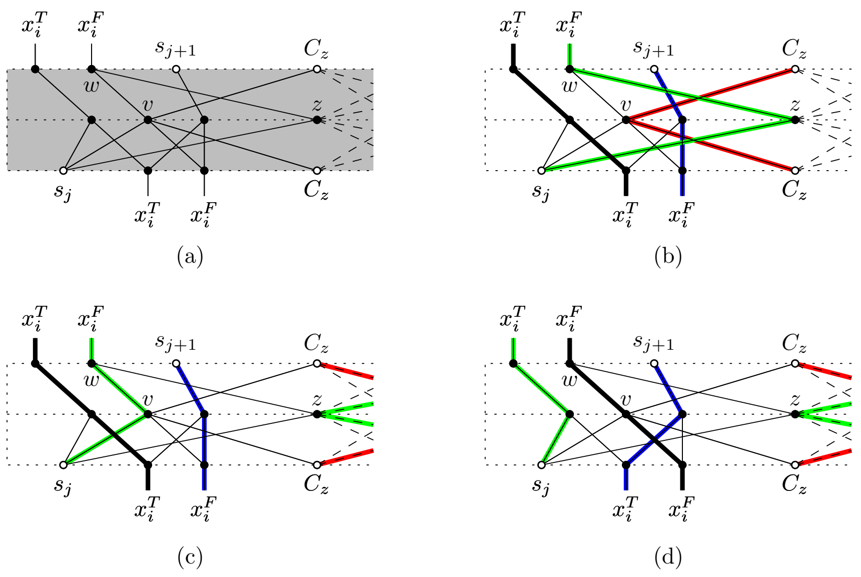


Figure 5 (a) Gadget for a clause C_z containing a variable x_i as a positive literal. The clause involves two other variables, whose connections are indicated by dashed lines. (b–d) The possibilities of paths passing through the gadget: (b) $x_i = \mathit{true}$, the clause is fulfilled, and the clause path makes its detour via the vertex v. (c) $x_i = \mathit{true}$, the clause is fulfilled, and the clause path makes its detour via another vertex. (d) $x_i = \mathit{false}$, this variable does not contribute to fulfilling the clause, and the clause path *has to* make its detour via another vertex. For a negative literal, the detour vertex v and the upper neighbor w of z would be placed on the other track, x_i^T.

4. The supplementary path s_j (shown in green) can reach its sink vertex.
5. The supplementary path s_{j+1} (in blue) can reach the track the track x_i^T or x_i^F which is not used by the variable path.

Since each clause path *must* make a detour into one of the variables, it follows from Property 2 that a set of disjoint source-sink paths exists if and only if all clauses are satisfiable.

The special properties of the graph that are required for the Disjoint Paths Problem on a Directed Acyclic Graph (DP) can be checked easily. Whenever a vertex has one or more incoming arcs, they come from the previous layer of the same clause gadget. We can assign three distinct layers to each clause gadget and to each crossing gadget. Thereby, we ensure that every layer contains at most 10 vertices. It follows from the construction that the graph has just enough vertices that the shortest source-sink paths can be disjointly packed, but we can also check this explicitly. (See the full version for an explicit calculation.) ◀

7 Reduction to Colored Token Swapping

We have shown in Lemma 8 how to reduce 3SAT to the Disjoint Paths Problem on a Directed Acyclic Graph (DP) in linear time. Now, we show how to reduce from this problem to the colored token swapping problem.

▶ **Lemma 9.** *There exists a linear reduction from DP to the colored token swapping problem.*

Proof. Let G be a directed graph and φ be a bijection, as in the definition of DP. We place k tokens $t_1, \ldots, t_k$ of distinct colors on the vertices in V^-, see Figure 6. Their target positions are in V^+ as determined by the assignment φ. We define a color for each layer $V_1, \ldots, V_{t-1}$. Each vertex in layer V_i which is not a sink is colored by the corresponding color.

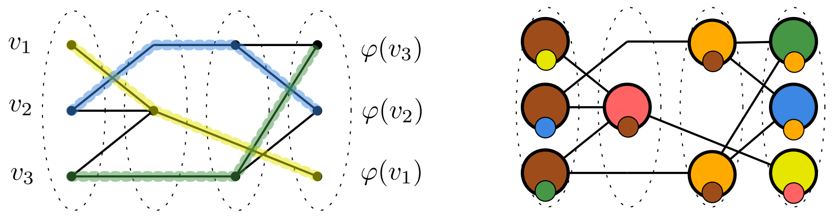


Figure 6 left: an instance of the disjoint paths problem with $t = 4$ and $k = 3$, together with a solution; right: an equivalent instance of the colored token swapping problem.

Recall that for each vertex v all ingoing edges come from the same layer, which we denote by L_v. On each vertex v which is not a source, we place a token with the color of layer L_v. We call these tokens *filler tokens*.

We set the threshold T to $|V(G)| - k$. This equals the number of filler tokens.

We have to show that there are k paths with the properties above if and only if there is a sequence of at most T swaps that brings every token to its target position (see the full version). ◀

We conclude that the Colored Token Swapping Problem is NP-hard. This has already been proved by Yamanaka et al. [23], even when there are only three colors.

The reduction in Lemma 9 produces instances of the colored token swapping problem with additional properties, which are directly derived from the properties of DP. A precise definition of the *structured token swapping problem* can be found in the full version.

8 Reduction to the Token Swapping Problem

In this section we describe the final reduction which results in an instance of the token swapping problem. To achieve this we make use of the even permutation network gadget from Section 5. For adhering to space constraints we omit from here an illustration of the reduction, the the full version for details.

▶ **Lemma 10.** *There exists a linear reduction from the structured colored token swapping problem to the token swapping problem.*

Proof. Let I be an instance of the structured colored token swapping instance. We denote the graph by G, the layers by $V_1, \ldots, V_t$, the sources by V^-, the sinks by V^+ and the threshold for the number of swaps by k.

We construct an instance J of the token swapping problem. The graph $\overline{G}$ consists of two copies of G. For each set $V_j \setminus V^+$, we add *one* even permutation network to the union of both copies. In other words, the two copies of $V_j \setminus V^+$ serve as inputs of the permutation network. We denote the output vertices of the permutation network attached to the copies of $V_j \setminus V^+$ by V_j'. The filler tokens that were are destined for $V_j \setminus V^+$ in I have V_j' as their new final destination in J, see the full version for illustrations and details. It is not important how we assign each token to a target vertex, as long as this assignment is consistent between the two copies of G (that is, for the bijection φ_j between the input and output vertices of the permutation network, we have that the tokens in the first copy are sent to the image under φ_j of the vertices in the first copy, while the tokens in the second are sent to the corresponding vertices in the image of the second copy).

Further each token gets a unique label. The threshold $\overline{k}$ for the number of swaps is defined by $2k$ plus the number of swaps needed for the permutation networks.

We show at first that this reduction is linear. For this it is sufficient to observe that the size of each layer is constant. And thus also the permutation network attached to these layers have constant size each.

Now we show correctness. Let I be an instance of the structured colored token swapping problem and J the constructed instance as described above.

[$\Rightarrow$] Let S be a valid sequence of swaps that brings every token on G to a correct target position within k swaps. We need to show that there is a sequence of $\overline{k}$ swaps that brings each token in $\overline{G}$ to its unique target position. We perform S on each copy of G. Thereafter every token is swapped through the permutation network to its target position. Note that the permutation between the input and output is an *even* permutation. This is because we doubled the graph G. Therefore, the number of swaps in the permutation network is constant regardless of the permutation of the filler tokens in layer V_i, by Lemma 7. This also implies that the total number of swaps is $\overline{k}$.

[$\Leftarrow$] Assume that there is a sequence S' of $\overline{k}$ swaps that brings each token of J to its target position. This implies that each filler token went through the permutation network to its correct target vertex. In order to do that each filler token must have gone to some input vertex of its corresponding permutation network. As the number of swaps inside each permutation network is independent of the permutation of the tokens on the input vertices, there remain exactly $2k$ swaps to put all tokens in each copy of G at its right place. This implies that each token in G can be swapped to its correct position in I in k swaps as claimed. ◀

9 Hardness of the Token Swapping Problem

In this section we put together all the reductions. They imply the following theorem.

▶ **Theorem 11.** *The token swapping problem has the following properties:*

1. *It is NP-complete.*
2. *It cannot be solved in time $2^{o(n)}$ unless ETH fails, where n is the number of vertices.*
3. *It is APX-hard.*

These properties also hold, when we restrict ourselves to instances of bounded degree.

Proof. For the NP-completeness, we reduce from 3SAT. For the lower bound under ETH, we need to use the Sparsification Lemma, see [13], and reduce from 3SAT instances where the number of clauses is linear in the number of variables. This prevents a potential quadratic blow up of the construction. For the inapproximability result, we reduce from 5-OCCURRENCE-MAX-3SAT. (In this variant of 3SAT each variable is allowed to have at most 5 *occurrences*.) This gives us some additional structure that we use for the argument later on. In all three cases the reduction is exactly the same.

Let f be a 3SAT instance. We denote by $K(f)$ the instance of the token swapping problem after applying the reduction of Lemma 8, Lemma 9 and Lemma 10 in this order. (It is easy to see that the graph of $K(f)$ has bounded degree.)

As all three reductions are correct, we can immediately conclude that the problem is NP-hard. NP-membership follows easily from the fact that a valid sequence of swaps is at most quadratic in the size of the input and can be checked in polynomial time.

The Exponential Time Hypothesis (ETH) asserts that 3SAT cannot be solved in $2^{o(n')}$, where n' is the number of variables. The Sparsification Lemma implies that 3SAT cannot be solved in $2^{o(m')}$, where m' is the number of clauses. As the reductions are linear the number of vertices in $K(f)$ is linear in the number of clauses of f. Thus a subexponential-time algorithm for the token swapping problem implies a subexponential-time algorithm for 3SAT and contradicts ETH.

To show APX-hardness, we do the same reductions as before, but we reduce from 5-OCCURRENCE-MAX-3SAT. Thus we can assume that each variable in f appears in at most 5 clauses. This variant of 3SAT is also APX-hard, see [1]. Assume a constant fraction of the clauses of f are not satisfiable. We have to show that we need an additional constant fraction on the total number of swaps. For this, we assume that the reader is familiar with the proofs of Lemma 8, 9 and 10. It follows from these proofs that there is a constant sized gadget in $K(f)$ for each clause of f. Also there are certain tokens that represent variables and the paths they take correspond to a variable assignment. We denote with x, y, z the variables of some clause C, and we denote with T_x, T_y, T_z the tokens corresponding to these variables and G_C the gadget corresponding to C. We need two crucial observations. In the case where the paths taken by the tokens T_x, T_y and T_z do not correspond to an assignment that makes C true, at least one more swap is needed. This swap can be attributed to the clause gadget G_C. In case that a token T_x changes its track, which corresponds to another assignment of the variable, then at least one more swap needs to be performed that can be attributed to all its clauses with value 1/5. These two observations follow from the proofs of the above lemmas, as otherwise it would be possible to "cheat" at each clause gadget and the above lemmas would be incorrect. The observations also imply the claim. Let f be a 3SAT formula with a constant fraction of the clauses not satisfiable. Assume at first that the swaps are "honest" in the sense that the variable tokens T_x does not change its track and corresponds consistently with the same assignment. In this case, by the first observation, we need at least one extra swap per clause. And thus a constant fraction of extra swaps, compared to the total number of swaps. In a dishonest sequence of swaps, changing the track of some variable token T_x fixes at most five clauses. This implies at least one extra swap for every five unsatisfied clauses, which is a constant fraction of all the swaps as the total number of swaps is linear in the number of clauses of f. This finishes the APX-hardness proof. ◀

Acknowledgment. This project was initiated on the GWOP 2014 workshop in Val Sinestra, Switzerland. We want to thank the organizers (GREMO) for inviting us to the very enjoyable workshop.

References

1 Sanjeev Arora and Carsten Lund. Hardness of approximations. In Dorit Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, pages 399–446. PWS Publishing, 1996.

2 Paul Bonsma, Takehiro Ito, Marcin Kamiński, and Naomi Nishimura. Invitation to combinatorial reconfiguration. Presentation at the First International Workshop on Combinatorial Reconfiguration (CoRe 2015), `http://www.ecei.tohoku.ac.jp/alg/core2015/page/template.pdf`, February 2015.

3 Paul Bonsma, Marcin Kamiński, and Marcin Wrochna. Reconfiguring independent sets in claw-free graphs. In R. Ravi and Inge Li Gørtz, editors, *Algorithm Theory – SWAT 2014*, volume 8503 of *Lecture Notes in Computer Science*, pages 86–97. Springer International Publishing, 2014. `doi:10.1007/978-3-319-08404-6_8`.

4 Arthur Cayley. Note on the theory of permutations. *Philosophical Magazine Series 3*, 34:527–529, 1849. `doi:10.1080/14786444908646287`.

5 Gruia Călinescu, Adrian Dumitrescu, and János Pach. Reconfigurations in graphs and grids. *SIAM Journal on Discrete Mathematics*, 22(1):124–138, 2008. `doi:10.1137/060652063`.

6 Erik D. Demaine, Martin L. Demaine, Eli Fox-Epstein, Duc A. Hoang, Takehiro Ito, Hirotaka Ono, Yota Otachi, Ryuhei Uehara, and Takeshi Yamada. Linear-time algorithm for sliding tokens on trees. *Theoretical Computer Science*, 600:132–142, 2015. doi:10.1016/j.tcs.2015.07.037.

7 Ruy Fabila-Monroy, David Flores-Peñaloza, Clemens Huemer, Ferran Hurtado, Jorge Urrutia, and David R. Wood. Token graphs. *Graphs and Combinatorics*, 28:365–380, 2012. doi:10.1007/s00373-011-1055-9.

8 Eli Fox-Epstein, Duc A. Hoang, Yota Otachi, and Ryuhei Uehara. Sliding token on bipartite permutation graphs. In Khaled Elbassioni and Kazuhisa Makino, editors, *Algorithms and Computation*, volume 9472 of *Lecture Notes in Computer Science*, pages 237–247. Springer Berlin Heidelberg, 2015. doi:10.1007/978-3-662-48971-0_21.

9 Parikshit Gopalan, Phokion G. Kolaitis, Elitza N. Maneva, and Christos H. Papadimitriou. The connectivity of Boolean satisfiability: Computational and structural dichotomies. *SIAM J. Comput.*, 38(6):2330–2355, 2009. doi:10.1137/07070440X.

10 Daniel Graf. How to sort by walking on a tree. In Nikhil Bansal and Irene Finocchi, editors, *Algorithms – ESA 2015*, volume 9294 of *Lecture Notes in Computer Science*, pages 643–655. Springer Berlin Heidelberg, 2015. doi:10.1007/978-3-662-48350-3_54.

11 Robert A. Hearn and Erik D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoretical Computer Science*, 343(1-2):72–96, 2005. doi:10.1016/j.tcs.2005.05.008.

12 Lenwood S. Heath and John Paul C. Vergara. Sorting by short swaps. *Journal of Computational Biology*, 10(5):775–789, 2003. doi:10.1089/106652703322539097.

13 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

14 Mark R. Jerrum. The complexity of finding minimum-length generator sequences. *Theoretical Computer Science*, 36:265–289, 1985. doi:10.1016/0304-3975(85)90047-7.

15 Marcin Kamiński, Paul Medvedev, and Martin Milanič. Complexity of independent set reconfigurability problems. *Theoretical Computer Science*, 439:9–15, 2012. doi:10.1016/j.tcs.2012.03.004.

16 Donald E. Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching.* Addison-Wesley, 1973.

17 Tillmann Miltzow, Lothar Narins, Yoshio Okamoto, Günter Rote, Antonis Thomas, and Takeaki Uno. Approximation and Hardness of Token Swapping. Preprint, February 2016. arXiv:1602.05150.

18 Amer E. Mouawad, Naomi Nishimura, Venkatesh Raman, and Marcin Wrochna. Reconfiguration over tree decompositions. In Marek Cygan and Pinar Heggernes, editors, *Parameterized and Exact Computation*, volume 8894 of *Lecture Notes in Computer Science*, pages 246–257. Springer International Publishing, 2014. doi:10.1007/978-3-319-13524-3_21.

19 Igor Pak. Reduced decompositions of permutations in terms of star transpositions, generalized Catalan numbers and k-ARY trees. *Discrete Mathematics*, 204(1-3):329–335, 1999. Selected papers in honor of Henry W. Gould. doi:10.1016/S0012-365X(98)00377-X.

20 Daniel Ratner and Manfred Warmuth. The (n^2-1)-puzzle and related relocation problems. *Journal of Symbolic Computation*, 10(2):111–137, 1990. doi:10.1016/S0747-7171(08)80001-6.

21 Richard M. Wilson. Graph puzzles, homotopy, and the alternating group. *Journal of Combinatorial Theory, Series B*, 16(1):86–96, 1974. doi:10.1016/0095-8956(74)90098-7.

22 Katsuhisa Yamanaka, Erik D. Demaine, Takehiro Ito, Jun Kawahara, Masashi Kiyomi, Yoshio Okamoto, Toshiki Saitoh, Akira Suzuki, Kei Uchizawa, and Takeaki Uno. Swapping labeled tokens on graphs. *Theoretical Computer Science*, 586:81–94, 2015. Special issue for the conference *Fun with Algorithms* 2014. doi:10.1016/j.tcs.2015.01.052.

23 Katsuhisa Yamanaka, Takashi Horiyama, David Kirkpatrick, Yota Otachi, Toshiki Saitoh, Ryuhei Uehara, and Yushi Uno. Swapping colored tokens on graphs. In Frank Dehne, Jörg-Rüdiger Sack, and Ulrike Stege, editors, *Algorithms and Data Structures*, volume 9214 of *Lecture Notes in Computer Science*, pages 619–628. Springer International Publishing, 2015. `doi:10.1007/978-3-319-21840-3_51`.

24 Gaku Yasui, Kouta Abe, Katsuhisa Yamanaka, and Takashi Hirayama. Swapping labeled tokens on complete split graphs. *SIG Technical Reports*, 2015-AL-153(14):1–4, 2015.

A 7/3-Approximation for Feedback Vertex Sets in Tournaments[*]

Matthias Mnich[†1], Virginia Vassilevska Williams[‡2], and László A. Végh[§3]

1 **Universität Bonn, Bonn, Germany**
mmnich@uni-bonn.de

2 **Computer Science Department, Stanford University, Stanford, USA**
virgi@cs.stanford.edu

3 **London School of Economics, London, UK**
l.vegh@lse.ac.uk

Abstract

We consider the minimum-weight feedback vertex set problem in tournaments: given a tournament with non-negative vertex weights, remove a minimum-weight set of vertices that intersects all cycles. This problem is NP-hard to solve exactly, and Unique Games-hard to approximate by a factor better than 2. We present the first 7/3 approximation algorithm for this problem, improving on the previously best known ratio 5/2 given by Cai et al. [FOCS 1998, SICOMP 2001].

1998 ACM Subject Classification F2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Approximation algorithms, feedback vertex sets, tournaments

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.67

1 Introduction

Among the most basic concepts in graph theory is the notion of a *feedback vertex set (FVS)* of a digraph: a subset of the vertices S such that removing S makes the digraph acyclic. The computational problem of finding a FVS of minimum size is known as the Feedback Vertex Set problem. A fundamental problem with numerous applications (e.g., in deadlock recovery in operating systems), the Feedback Vertex Set problem is among Karp's 21 original NP-complete problems [13]. Karp's proof of NP-hardness also implies that the problem is APX-hard. Obtaining a constant factor polynomial-time approximation algorithm for the Feedback Vertex Set problem seems elusive and is a major open problem. The best known approximation factor achievable in polynomial time is $O(\log n \log\log n)$ [8, 21].

The Feedback Vertex Set problem is particularly interesting for the special case when the input graph is a *tournament*, i.e., an orientation of the complete graph. The problem restricted to tournaments has many interesting applications, most notably in social choice theory where it is essential to the definition of a certain type of election winners called the Banks set [1].

[*] Part of this work was done during the trimester program on Combinatorial Optimization at the Hausdorff Institute for Mathematics in Bonn; we would like to thank HIM for their hospitality during the program.
[†] Supported by ERC Starting Grant 306465 (BeyondWorstCase).
[‡] Supported by NSF Grants CCF-1417238, CCF-1528078 and CCF-1514339, and BSF Grant BSF:2012338.
[§] Supported by EPSRC First Grant EP/M02797X/1.

The Feedback Vertex Set problem remains NP-complete and APX-hard in tournaments. Moreover, Speckenmeyer [22] gave an approximation-ratio preserving polynomial time reduction from the Vertex Cover problem in general undirected graphs to the Feedback Vertex Set problem in tournaments. Consequently, the FVS problem in tournaments cannot be approximated in polynomial time within a factor better than 1.3606, unless $\mathsf{P} = \mathsf{NP}$ [6], and not within a factor better than 2 assuming the Unique Games Conjecture (UGC) [14].

On the upper bound side, the Feedback Vertex Set problem in tournaments admits an easy 3-approximation algorithm: while the tournament contains a directed triangle, place all the triangle vertices in the FVS and remove them from the tournament (see also Bar-Yehuda and Rawitz [2] for another simple 3-approximation algorithm). Cai, Deng and Zang [4] improved the simple algorithm and gave a polynomial time algorithm with approximation guarantee 5/2, even in the case when vertices have non-negative weights and one seeks a solution of approximate minimum weight.

In this paper we develop a 7/3-approximation algorithm for the minimum weight Feedback Vertex Set problem in tournaments, obtaining the first improvement over the eighteen year old result of Cai et al. [4]. Our result shows that the 2.5-approximation ratio is not best possible, and gives hope that a 2-approximation algorithm, that would be optimal under the UGC, might be achievable.

▶ **Theorem 1.** *There exists a polynomial-time* 7/3*-approximation algorithm for finding a minimum-weight feedback vertex set in a tournament.*

In the process of proving the above theorem, we uncover a structural theorem about tournament graphs that has interesting connections to the tournament colouring problem investigated by Berger et al. [3]. We explain these connections in Sect. 5.

1.1 Overview

Let us first give an overview of Cai et al.'s result [4]. Let $\mathcal{T}_5$ denote the set of tournaments on 5 vertices where the minimum FVS has size 2 (note that every tournament on 5 vertices has a FVS of size at most 2). Cai et al. showed that for any tournament free of subtournaments from $\mathcal{T}_5$, the minimum-weight FVS problem becomes polynomial-time solvable. They in fact show that the natural LP relaxation of the problem is integral in $\mathcal{T}_5$-free tournaments: the minimum weight of a FVS equals the maximum value of a fractional directed triangle packing.

For the special case of unit weights only, their 5/2-approximation algorithm starts by greedily choosing subtournaments in $\mathcal{T}_5$, and including all 5 vertices in the FVS. Once the remaining tournament admits no more subtournaments in $\mathcal{T}_5$, the optimal covering algorithm is used. The algorithm returns a 5/2-approximate solution, since every step of removing a subtournament decreases the optimum value by at least 2, and includes 5 vertices in the FVS. The algorithm extends to non-negative weights using the local ratio technique.

We now give an overview of our approach. We define the set $\mathcal{T}_7$ as the set of 7-vertex tournaments where the minimum size of a FVS is 3. The algorithm comprises two stages. The first stage uses the iterative rounding technique, and removes all subtournaments in $\mathcal{T}_7$; the weight of the vertices included at this stage will be at most 7/3-times the decrease in the optimum weight. In the second stage, we give a 7/3-approximate combinatorial algorithm for the remaining $\mathcal{T}_7$-free tournament.

The analogous first stage of Cai et al. obtains a worse factor 5/2. In the second stage, their algorithm delivers an optimal solution. In contrast, we only give an approximation algorithm in the second stage, but that is sufficient for the overall approximation guarantee.

We now provide some more detail of the two stages. In the first stage we use the iterative rounding technique. We formulate the natural LP relaxation of the minimum-weight FVS problem in the given tournament T, including a covering constraint for every directed triangle of T, and further we include that every subtournament of T belonging to $\mathcal{T}_7$ must be covered by at least three vertices. We consider an optimal solution of the LP relaxation. If there is a vertex of T with fractional value at least $3/7$, we include it in our FVS and remove it from T. We then resolve the LP on the remaining tournament, and again include a vertex with fractional value at least $3/7$, if there exists one. We iterate until there are no more such vertices. At this point, the tournament will be $\mathcal{T}_7$-free, and the fractional optimum value equals exactly one third of the total weight of the vertices (see Lemma 6).

In the second stage, we develop a polynomial time combinatorial algorithm that delivers a FVS of weight at most $7/9$ times the *total weight* of the vertices in a $\mathcal{T}_7$-free tournament (Theorem 4). This algorithm implies our main theorem, since an optimal FVS in the remaining $\mathcal{T}_7$-free tournament is of size at least the optimum fractional value, which by the previous paragraph is exactly a third of the total weight of the nodes, which itself is at least $1/3 \cdot 9/7 = 3/7$ of the size of the FVS returned.

To prove Theorem 4, we decompose the vertex set into "layers". Our algorithm divides the vertices into $\mathcal{T}_5$-free layers, while also identifying a certain vertex set S to be included in the FVS right away. Our final FVS will be composed of the initially selected S, every second layer, and the optimal FVS's inside the remaining layers. To obtain these, we use Cai et al.'s algorithm as a subroutine to find an optimal solution on a $\mathcal{T}_5$-free layer. The layering idea is inspired by Cai et al.'s structural analysis of $\mathcal{T}_5$-free tournaments; nevertheless, we use it quite differently.

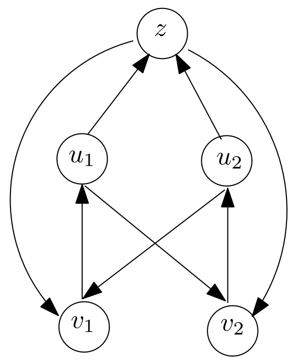
It is natural to conjecture that our approach can be extended to lead to a $(2 + \varepsilon)$-approximation for the FVS problem in tournaments, for all $\varepsilon > 0$. At this point it is unclear how to improve the approximation ratio in the above second stage. Nevertheless, our paper provides the next substantial step towards reaching the UGC-based lower bound.

1.2 Related work

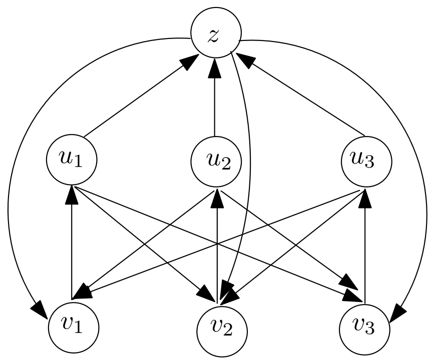
Feedback vertex sets in tournaments are a well-studied subject. Dom et al. [7] showed how to decide existence of an FVS of size at most k in time $2^k \cdot n^{O(1)}$, and gave a kernel with $O(k^2)$ vertices. An exponential-time algorithm by Fomin et al. [10] finds an FVS of minimum size in time $O(1.674^n)$, improving on earlier algorithms [18, 7, 11, 16]. Gaspers and Mnich [11] gave a polynomial-space algorithm to enumerate all minimal FVS of a given tournament with polynomial delay; the currently best upper bound on their number is $O(1.6667^n)$ [10].

The related question of FVS in *bipartite* tournaments has also been studied, i.e., orientations of the complete bipartite graph. First, Cai et al. [5] using a similar framework to their 5/2-approximation algorithm [4], developed a 7/2-approximation algorithm for FVS in bipartite tournaments. This was improved by Sasatte [20] giving a 3-approximation, and finally, by van Zuylen [23] who developed a polynomial time 2-approximation algorithm.

Iterative rounding is a standard and powerful method in approximation algorithms; we refer the reader to the book by Lau et al. [17]. The approach was made popular by Jain's groundbreaking 2-approximation for survivable network design [12], and the main application area is network design. However, the same principle was already used earlier for various



(a) S_5



(b) S_7

Figure 1 Examples of tournaments from $\mathcal{T}_5$ and $\mathcal{T}_7$.

problems. In particular, Krivelevich used implicitly iterative rounding for the undirected triangle cover problem [15]; our application is similar to his argument. Van Zuylen [23] used iterative rounding for FVS in bipartite tournaments.

The Cluster Vertex Deletion problem is another restrictions of the vertex cover problem in 3-uniform hypergraphs. Here the goal is to cover all induced paths of length 2 in an undirected graph. A very recent paper by Fiorini et al. [9] provides a 7/3-approximation algorithm for Cluster Vertex Deletion, improving on the previous best ratio 2.5. An approximation-preserving reduction from the Vertex Cover problem shows that the best possible factor is 2 under the Unique Games Conjecture. Despite these similarities, no approximation-preserving reduction is known between Cluster Vertex Deletion and FVS in Tournaments. The techniques used are also quite different.

2 Description of the Algorithm

Let $T = (V, A)$ be a tournament, equipped with a weight function $w : V \to \mathbb{R}_{\geq 0}$. An arc between $u, v \in V$ will be denoted by $(u, v) \in A$ or $u \to v$. The tournament T is *transitive* if it does not contain any directed cycles, or equivalently, its vertices admit a topological order. A vertex set $S \subseteq V$ is a *feedback vertex set* if $T[V \setminus S]$ is transitive. For a vertex set $S \subseteq V$, let $T - S$ denote the tournament resulting from the removal of the vertex set S from T. If $S = \{v\}$ has a single element, we also use the notation $T - v$.

The following straightforward characterization of FVS's in tournaments is well-known.

▶ **Proposition 2.** *For any tournament T, a set S is a feedback vertex set for T if and only if S intersects every directed triangle of T.*

Let $\mathcal{T}_5$ denote the family of tournaments T' on 5 vertices that do not contain a transitive subtournament on 4 vertices; equivalently, every FVS of T' has size at least 2. The set $\mathcal{T}_5$ contains 3 tournaments, the same ones used by Cai et al. [4]. Characterizations of many related classes of tournaments were given by Sanchez-Flores [19].

Our main focus will be the set $\mathcal{T}_7$ defined as follows. Let $\mathcal{T}_7$ denote the family of tournaments on 7 vertices that do not contain a transitive subtournament on 5 vertices. This is equivalent to the property that every FVS is of size at least 3. We remark that $\mathcal{T}_7$ consists of 121 tournaments.

Fig. 1 gives important examples of tournaments $S_5 \in \mathcal{T}_5$ and $S_7 \in \mathcal{T}_7$. The arcs not included in the figures can be oriented arbitrarily; hence both figures represent multiple

Algorithm 1 TOURNAMENT FVS

```
Input: A tournament $T = (V, A)$ with weight function $w : V \to \mathbb{Q}_{\geq 0}$.
Output: A feedback vertex set of $T$ of weight at most $\frac{7}{3}OPT(T)$.
1: Initialize $F = \emptyset$, $T' = T$.
2: Find an optimal solution $x^*$ to (LP).
3: while $T' \neq \emptyset$ and there exists a vertex $v \in V(T')$ with $x^*_v \geq \frac{3}{7}$ do
4:     Set $F := F \cup \{v : x^*_v \geq \frac{3}{7}\}$ and $T' := T' \setminus \{v : x^*_v \geq \frac{3}{7}\}$.
5:     Remove every vertex from $T'$ not contained in any directed triangle; denote this
       resulting tournament also by $T'$.
6:     Solve (LP) for $T'$ to obtain an optimal solution $x^*$.
7: If $T' \neq \emptyset$ then run Algorithm LAYERS (Algorithm 2) for $T'$, returning a FVS $F'$ of $T'$.
8: return $F \cup F'$.
```

tournaments. Tournament S_5 is identical to F_1 of Cai et al. [4]. We leave the proof of the following simple claim to the reader.

▶ **Proposition 3.** *For the tournaments in Fig. 1, $S_5 \in \mathcal{T}_5$ and $S_7 \in \mathcal{T}_7$.*

For a tournament T, let $\Delta(T)$ denote the family of vertex sets of directed triangles in T. According to Proposition 2, T is transitive if and only if $\Delta(T) = \emptyset$. Similarly, $\mathcal{T}_5(T)$ and $\mathcal{T}_7(T)$ denote the family of vertex sets of the subtournaments of T isomorphic to a tournament in $\mathcal{T}_5$ and $\mathcal{T}_7$, respectively. We say that T is $\mathcal{T}_5$*-free* if $\mathcal{T}_5(T) = \emptyset$ and $\mathcal{T}_7$*-free* if $\mathcal{T}_7(T) = \emptyset$.

We use iterative rounding for the following LP relaxation of the FVS problem in a tournament $T = (V, A)$ with weight function $w : V \to \mathbb{Q}_{\geq 0}$. For a vector $x : V \to \mathbb{R}$ and a set $S \subseteq V$, let $x(S) = \sum_{v \in S} x_v$.

$$\begin{aligned} &\min\ w^T x \\ &x(R) \geq 1 \quad \forall R \in \Delta(T) \\ &x(Q) \geq 3 \quad \forall Q \in \mathcal{T}_7(T) \\ &\quad x \geq 0 \end{aligned} \tag{LP}$$

Notice that (LP) does not impose any constraints for subtournaments in $\mathcal{T}_5(T)$. This is an LP of polynomial size. Let $OPT(T)$ denote the optimum value of (LP).

Our algorithm (Algorithm 1), iteratively builds a FVS F of T, initialized empty. We find an optimal solution x^* to (LP), and as long as there exist vertices v such that $x^*_v \geq \frac{3}{7}$, we include all of them in F and remove them from T. We iterate this process, by resolving the LP for the smaller tournament T'. By the first stage of the algorithm we mean the sequence of these iterative rounding steps, which terminate once T' becomes empty (in which case we are done), or every fractional value x^*_v satisfies $x^*_v < \frac{3}{7}$.

In this case, the current tournament T' must be $\mathcal{T}_7$-free. Indeed, the constraint on the elements of $\mathcal{T}_7(T')$ guarantees that in every $\mathcal{T}_7$ subtournament at least one element must have fractional value at least 3/7. Note that this is true already after the very first iteration. The analogous task of removing all subtournaments from $\mathcal{T}_5(T')$ is done by Cai et al. [4] using the local ratio technique. As shown by Bar-Yehuda and Rawitz [2], this could also be done via a primal-dual algorithm. The local ratio and primal-dual techniques easily give a 3-approximation for the formulation with triangles only (given as (P) in the next section). However, these do not seem to easily extend for our second goal with the iterative rounding, when we only have triangle constraints left, and we proceed as long as there is a vertex of fractional value at least 3/7.

In the second stage we apply Algorithm Layers (Algorithm 2). That is the algorithm described in the following theorem.

▶ **Theorem 4.** *There is an algorithm that, given any $\mathcal{T}_7$-free tournament $T' = (V', A')$ with weight function $w : V' \to \mathbb{Q}_{\geq 0}$, in polynomial time finds a FVS F' of T' of weight at most $\frac{7}{9}w(V')$.*

We defer the description of Algorithm Layers as well as the proof of Theorem 4 to Sect. 4. We now prove the validity of Algorithm 1, provided this result.

3 Proof of Theorem 1

It is straightforward to see that the set $F \cup F'$ returned by the algorithm is a FVS of T. The next simple lemma shows that in every iterative rounding step, the weight of the elements added to F can be bounded by the decrease of $OPT(T)$.

▶ **Lemma 5.** *In every iteration during the first stage of the algorithm with current tournament T' and set F, we have*

$$w(F) \leq \frac{7}{3}(OPT(T) - OPT(T')) \ .$$

Proof. We prove the claim by induction. It is clearly true at the beginning when $T' = T$. Whenever we remove a vertex not contained in any triangle, the left-hand side remains unchanged and the right-hand side may only increase. It is sufficient to prove that if x^* is an optimal solution to (LP) for T' and $S = \{v : x^*_v \geq \frac{3}{7}\} \neq \emptyset$, then $OPT(T' \setminus S) + \frac{3}{7}w(S) \leq OPT(T')$.

Note that x^* restricted to $T' \setminus S$ is feasible to (LP) for $T' \setminus S$, and thus $OPT(T' \setminus S) \leq OPT(T') - \sum_{v \in S} w(v)x^*_v \leq OPT(T') - \frac{3}{7}w(S)$, as required. ◀

As observed above, the tournament T' at the end of the first stage is $\mathcal{T}_7$-free. Theorem 4 guarantees that the FVS F' of T' returned by Algorithm Layers has weight $w(F') \leq \frac{7}{9}w(T')$.

▶ **Lemma 6.** *At the end of the first stage, $OPT(T') = \frac{1}{3}w(T')$.*

Before proving this lemma, let us see how it concludes the proof of Theorem 1. According to Theorem 4 and Lemma 6, $w(F') \leq \frac{7}{9}w(T') \leq \frac{7}{3}OPT(T')$. Using Lemma 5, we see that the weight of the constructed FVS $F \cup F'$ is $w(F \cup F') \leq \frac{7}{3}OPT(T)$.

The proof of Lemma 6 analyzes the LP relaxation with triangle constraints only. At the end of the first stage, T' is $\mathcal{T}_7$-free. Hence, the second set of constraints in (LP) for T' is empty. Let us omit these constraints and write (LP) together with its dual:

$$\begin{aligned} \min \quad & w^T x \\ & x(R) \geq 1 \quad \forall R \in \Delta(T') \\ & x : V \to \mathbb{R}_+ \end{aligned} \qquad \text{(P)} \qquad\qquad \begin{aligned} & \max \mathbf{1}^T y \\ & \sum_{R: v \in R} y_R \leq w_v \quad \forall v \in V' \\ & y : \Delta(T') \to \mathbb{R}_+ \end{aligned} \qquad \text{(D)}$$

Proof of Lemma 6. If $\Delta(T') \neq \emptyset$, then T' is empty and the statement of the lemma holds. Therefore, we can assume that $\Delta(T') \neq \emptyset$, and that $x^*_v \leq \frac{3}{7}$ for every $v \in V'$, where $V' = V(T')$ is the vertex set of T'.

▶ **Claim 7.** *$x^*_v > 0$ for every $v \in V'$.*

Proof. For sake of contradiction, suppose that $x_v^* = 0$ for some $v \in V'$. Every vertex in T' is contained in a directed triangle; say $\{v, u, z\} \in \Delta(T')$. The relaxation (LP) includes a constraint $x_v^* + x_u^* + x_z^* \geq 1$, and therefore $x_u^* \geq \frac{1}{2}$ or $x_z^* \geq \frac{1}{2}$, a contradiction to $x_v^* \leq \frac{3}{7}$ for all $v \in V'$. ◀

By primal-dual slackness, we must have $\sum_{u \in R} y_R = w(u)$ for all $u \in V'$. Then

$$w(V') = \sum_{u \in V'} \sum_{R: u \in R} y_R = \sum_{R \in \Delta(T')} y_R \sum_{u \in R} 1 = 3 \sum_{R \in \Delta(T')} y_R = 3 \cdot OPT(T'),$$

completing the proof. In the third equation, we used that every triangle contains exactly three vertices. ◀

4 The Algorithm Layers

In this section, we present Algorithm Layers and prove Theorem 4. First, we need the following result by Cai et al. [4, Sect. 4].

▶ **Theorem 8** ([4]). *There exists an algorithm that, given any $\mathcal{T}_5$-free tournament $\hat{T}$ with non-negative vertex weights, finds in polynomial time a minimum weight FVS in $\hat{T}$.*

We shall refer to the algorithm as the Cai-Deng-Zang algorithm. We also need a property of $\mathcal{T}_5$-free tournaments established by Cai et al. [4, Thm. 3.2].

▶ **Proposition 9** ([4]). *For any $\mathcal{T}_5$-free tournament $\hat{T}$ with non-negative vertex weights, the minimum weight of a FVS equals the maximum value of a fractional triangle packing.*

Observe that computing the maximum value of a fractional triangle packing amounts to solving (D) to optimality.

The next simple lemma bounds the cost of the FVS found by the Cai-Deng-Zang algorithm in terms of the total weight of the vertices $w(\hat{V})$.

▶ **Lemma 10.** *Let $\hat{T} = (\hat{V}, \hat{A})$ be a $\mathcal{T}_5$-free tournament with weight function $w : \hat{V} \to \mathbb{Q}_{\geq 0}$, and let $\hat{F}$ be an FVS of $\hat{T}$ returned by the Cai-Deng-Zang algorithm applied to $(\hat{T}, w)$. Then $w(\hat{F}) \leq w(\hat{V})/3$.*

Proof. By Proposition 9, the polyhedron (P) applied to $T' = \hat{T}$, and w is integral. Setting $x_v = \frac{1}{3}$ for every $v \in \hat{V}$ is a feasible solution, and hence $w(\hat{F}) \leq w(\hat{V})/3$. ◀

4.1 Layers from a vertex

Recall that Theorem 4 takes as input a $\mathcal{T}_7$-free tournament $T' = (V', A')$ with weight function $w : V' \to \mathbb{Q}_{\geq 0}$. For a set $S \subseteq V'$, let $N(S) = \{v \notin S \mid \exists u \in S, v \to u\}$ denote the set of its in-neighbours; let $N(u) := N(\{u\}) = \{v \mid v \to u\}$.

For any vertex $z \in V'$ and $\ell \in \{1, \ldots, n\}$, let us define $V_\ell(z)$ as the set of vertices v such that the shortest directed path from v to z has length exactly $\ell - 1$. Equivalently, let $V_1(z) = \{z\}$, $V_2(z) = N(z)$, and for $\ell \geq 2$ let

$$V_{\ell+1}(z) := \{v \in V' \setminus (V_1(z) \cup \ldots \cup V_\ell(z)) \mid \exists u \in V_\ell(z), v \to u\} \ .$$

These correspond to the layers of the BFS algorithm starting from z. We will prove the following structural result. For two disjoint sets $S, Z \subseteq V'$, let us say that Z *in-dominates* S if for every $s \in S$ there exists a $z \in Z$ with $s \to z$. We say that Z *2-in-dominates* S if Z has a subset $Z' \subseteq Z$ with $|Z'| \leq 2$ such that Z' in-dominates S.

▶ **Theorem 11.** *For every vertex z of positive in-degree, the following hold:*

(a) *The set $V_3(z)$ is $\mathcal{T}_5$-free, and is 2-in-dominated by $V_2(z)$.*

(b) *The set $V_4(z)$ is $\mathcal{T}_5$-free, and is 2-in-dominated by $V_3(z)$.*

(c) *If z is a minimum in-degree vertex in the tournament, then $V_3(z) \neq \emptyset$, and $V_2(z)$ is also $\mathcal{T}_5$-free.*

The proof of Theorem 11 is given in Sect. 4.4. Let us now provide some context and motivation. Cai et al. [4] showed that for any $\mathcal{T}_5$-free tournament, if we select a minimum in-degree vertex z, then every layer $V_i(z)$ induces a transitive tournament and is 1-in-dominated by $V_{i-1}(z)$. This is an important step in their algorithm for finding the exact optimal solution in $\mathcal{T}_5$-free tournaments.

Assume that the analogous property held for $\mathcal{T}_7$-free tournaments T': starting from a minimum in-degree vertex z, every layer $V_{i-1}(z)$ is $\mathcal{T}_5$-free. Then one could get a FVS of T' with weight at most $\frac{2}{3}w(V')$ as follows. Compare the total weight of the even and odd layers, and include in the FVS whichever of the two is smaller. Let us assume the total weight of the odd layers is smaller; the argument is same for the other case. For every remaining even layer $V_i(z)$, run the Cai-Deng-Zang algorithm to obtain a FVS F_i of $V_i(z)$. Form the final FVS F' of T' as the union of all odd layers and the union of the F_i's for the even layers. Using Proposition 10, it is easy to verify that $w(F') \leq \frac{2}{3}w(V')$. Further, F' will be a FVS of T', since by the construction of the $V_i(z)$'s, every triangle must fall on consecutive layers. That is, it is, if a triangle T intersects layers $V_i(z)$ and $V_j(z)$ with $i < j$, then $j \leq i + 2$, and if $j = i + 2$ then T must also intersect layer $V_{i+1}(z)$.

However, Theorem 11 only claims $\mathcal{T}_5$-freeness of layers $V_i(z)$ for $i \leq 4$. This property might not hold for higher values of i. To overcome this difficulty, we modify the layering procedure. While the layers are constructed, we already include certain vertices in the final FVS. This is to make sure that for every layer U_i, it holds that $U_i = V_j(z')$ in some subtournament of T, for a certain vertex z' in a previous layer and $j = 3$ or $j = 4$. Hence Theorem 11 guarantees that all constructed layers are $\mathcal{T}_5$-free. The construction of the final FVS will be a modification of the simple argument above.

4.2 Description of the layering algorithm

The algorithm (Algorithm 2) first partitions the vertex set V' into $S \cup \bigcup_{j=1}^{2k} U_j$ for some $2k \leq n$. We now describe how the layers are constructed in Steps 1-11. We start by setting $U_1 = \{z_1\}$ for a vertex z_1 of minimum in-degree. We let $U_2 = N(z_1)$ be the set of in-neighbours of z_1. The set W will denote the set of vertices not yet included in some U_k or in S; at this point, $W = V' \setminus (U_1 \cup U_2)$.

While W is not empty, we construct an odd layer U_{2k+1}, an even layer U_{2k+2}, and S_{2k+1} as follows. First consider the case when U_{2k} has at least one in-neighbour in W. We set $U_{2k+1} = N(U_{2k}) \cap W$, and remove U_{2k+1} from W. Let U' be the set of in-neighbours of U_{2k+1} in W. We note that $U' = \emptyset$ is possible. We partition U' into U_{2k+2} and S_{2k+2}, and remove U' from W. To obtain this partitioning, we pick a vertex $z_{2k+1} \in U_{2k+1}$ such that $w(N(z_{2k+1}) \cap U') \geq w(U')/2$. The existence of such a vertex z_{2k+1} is non-trivial, and will be proved in Lemma 12(c). We set $U_{2k+2} = N(z_{2k+1}) \cap U'$, and $S_{2k+2} = U' \setminus U_{2k+2}$; the set S_{2k+2} will be part of S.

Let us now address the case when U_{2k} does not have any in-neighbours in W. In this case, we select $U_{2k+1} = \{z\}$ for a vertex $z \in W$ that has minimum in-degree inside W. We refer to the latter scenario as a *fresh start.* We set U_{2k+2} as the set of in-neighbours of z in W, and remove these vertices from W; here $U_{2k+2} = \emptyset$ is possible.

Algorithm 2 LAYERS

```
Input: A $\mathcal{T}_7$-free tournament $T' = (V', A')$ with weight function $w : V' \to \mathbb{Q}_{\geq 0}$.
Output: A feedback vertex set $F'$ of $T'$ of weight at most $\frac{7}{9}w(V')$.
1: Choose $z_1$ as a vertex of minimum in-degree.
2: Set $U_1 := \{z_1\}$,
3: Set $U_2 := N(z_1)$, $W := V' \setminus (U_1 \cup U_2)$, $k := 1$.
4: while $W \neq \emptyset$ do
5:     if $N(U_{2k}) \cap W \neq \emptyset$ then
6:         Set $U_{2k+1} := N(U_{2k}) \cap W$,
7:         $W := W \setminus U_{2k}$.
8:         Set $U' := N(U_{2k+1}) \cap W$, $W := W \setminus U'$.
9:         Choose $z_{2k+1} \in U_{2k+1}$ such that $w(U' \cap N(z_{2k+1})) \geq w(U')/2$.
10:        Set $U_{2k+2} := U' \cap N(z_{2k+1})$; $S_{2k+2} := U' \setminus N(z_{2k+1})$.
11:    else        //fresh start
12:        Choose $z \in W$ with $|N(z) \cap W|$ minimal.
13:        Set $U_{2k+1} := \{z\}$, $U_{2k+2} := N(z) \cap W$, and $S_{2k+2} := \emptyset$.
14:        Set $W := W \setminus (U_{2k+1} \cup U_{2k+2})$.
15:    Set $k := k + 1$.
16: Set $L_0 := \cup_{j=1}^{k} U_{2j}$, $L_1 := \cup_{j=0}^{k-1} U_{2j+1}$, and $S := \cup_{j=1}^{k} S_{2j}$.
17: if $w(L_0) \geq w(L_1)$ then
18:    Run the CAI-DENG-ZANG algorithm for every $U_{2j}$ to obtain a FVS $F_{2j}$ of $U_{2j}$.
19:    Set $F' := (\cup_{j=1}^{k} F_{2j}) \cup S \cup L_1$.
20: else
21:    Run the CAI-DENG-ZANG algorithm for every $U_{2j+1}$ to obtain a FVS $F_{2j+1}$ of $U_{2j+1}$.
22:    Set $F' := (\cup_{j=0}^{k-1} F_{2j+1}) \cup S \cup L_0$.
23: return $F'$.
```

The layering procedure finishes once $W = \emptyset$. At this point, we denote by $L_0 = \bigcup_{j=1}^{k} U_{2j}$ the set of all even and by $L_1 = \bigcup_{j=0}^{k-1} U_{2j+1}$ the set of all odd layers, and by $S = \bigcup_{j=1}^{k} S_{2j}$ the set of vertices removed during the procedure. Thus, $V' = S \cup L_0 \cup L_1$. Given the layering, the algorithm constructs a FVS in Steps 12-18 as follows. If $w(L_0) \geq w(L_1)$, then we use the CAI-DENG-ZANG algorithm to find an optimal FVS F_{2j} in all even layers U_{2j}. We set the entire FVS as $F' := (\cup_{j=1}^{k} F_{2j}) \cup S \cup L_1$. Otherwise, we use the CAI-DENG-ZANG algorithm in all odd layers to find optimal FVS's F_{2j+1}, and set $F' := (\cup_{j=0}^{k-1} F_{2j+1}) \cup S \cup L_0$.

The algorithm clearly runs in polynomial time: every while cycle decreases the size of W by at least one, and every step amounts to examining in-neighbourhoods of vertices and comparing weights of sets.

4.3 Proof of correctness

The following lemma summarizes the essential properties of the layering obtained.

▶ **Lemma 12.** *The sets S and U_i returned by Algorithm* LAYERS *satisfy the following properties.*

(a) *If $i > j + 1$, then $u \to v$ for every $u \in U_j$ and $v \in U_i$.*
(b) *Every subtournament $T'[U_i]$ is $\mathcal{T}_5$-free.*
(c) *There always exists a vertex $z_{2i+1} \in U_{2i+1}$ as required in line 9 of the algorithm.*
(d) $w(S) \leq w(L_0)$.

Proof. Part (a) is immediate, since if $u \in U_j$, then $N(u) \subseteq \cup_{\ell=0}^{j+1}(U_\ell \cup S_\ell)$ (let us use the convention $S_\ell = \emptyset$ for all odd values of ℓ).

We prove parts (b) and (c) simultaneously. We prove it for all layers before the first fresh start happens. If $N(z_1) = \emptyset$, then $U_2 = \emptyset$, hence U_1 and U_2 are trivially $\mathcal{T}_5$-free, and U_3 will be obtained by a fresh start. Otherwise, part (b) is a direct consequence of Theorem 11 for layers $1 \le i \le 4$, as z_1 was chosen as a minimum in-degree vertex; note that $V_3(z_1) \ne \emptyset$ and hence $U_3 = V_3(z_1)$ was not obtained by a fresh start. In this case, the existence of vertex $z_3 \in U_3$ follows by Theorem 11(c): $U' = V_4(z_1)$, and thus U' is 2-in-dominated by U_3. This means that there exist $z, z' \in U_3$ such that $N(z) \cup N(z') \supseteq U'$ (we allow $z = z'$). Without loss of generality, we may assume $w(U' \cap N(z)) \ge w(U' \cap N(z'))$. Then $z_3 = z$ gives an appropriate choice.

Assuming that U_5 is not obtained by a fresh start, let us apply Theorem 11 in the tournament T'' that is the restriction of T' to the ground set $\{z_3\} \cup U_4 \cup U_5 \cup (U_6 \cup S_6)$. In T'' we have $V_3(z_3) = U_5$ and $V_4(z_3) = U_6 \cup S_6$, and therefore U_5 and $U_6 \cup S_6$ are both $\mathcal{T}_5$-free. Further, $U_6 \cup S_6$ is 2-in-dominated by U_5 and therefore we can choose an appropriate $z_5 \in U_5$ as above. The same argument works for all values of $i \ge 3$: consider the restriction of T' to $\{z_{2i-1}\} \cup U_{2i} \cup U_{2i+1} \cup (U_{2i+2} \cup S_{2i+2}$, and apply Theorem 11. We obtain that U_{2i+1} and $U_{2i+2} \cup S_{2i+2}$ are $\mathcal{T}_5$-free as well as the choice of $z_{2i+1} \in U_{2i+1}$.

Assume now that a certain layer U_{2i+1} is obtained by a fresh start. Then we can apply the same argument as above to show parts (b) and (c) for all subsequent layers until the next fresh start: we restrict the tournament from V to the ground set W at the beginning of the iteration when U_{2i+1} is constructed.

Finally, part (d) is straightforward, since $w(U_{2i+2}) \ge w(S_{2i+2})$ by the choice of z_{2i+2}. ◀

We are ready to prove the correctness and approximation ratio of the algorithm.

Proof of Theorem 4. By Lemma 12(b), the CAI-DENG-ZANG algorithm can be applied in all layers U_i and finds an optimal FVS F_i in polynomial time.

First, let us show that the set F' returned by Algorithm LAYERS is indeed a FVS of T'. For a contradiction, assume $V' \setminus F'$ contains a directed triangle uvs.

Let us assume $w(L_0) \ge w(L_1)$; the other case follows similarly. In this case, $V' \setminus F' \subseteq L_0$. The three vertices u, v and s cannot fall into the same layer U_{2i}, as in every such layer we removed a FVS F_{2i}. Hence they must fall into at least two different U_{2i}'s. By Lemma 12(a), if vertices fall into different even layers, then all arcs from the lower layers point towards the higher layers, excluding the possibility of such a triangle.

The proof is complete by showing $w(F') \le \frac{7}{9}w(V')$, or equivalently, $w(V' \setminus F') \ge \frac{2}{9}w(V')$.

Case I: $w(L_0) \ge w(L_1)$. In this case, $w(V' \setminus F') = \cup_{j=1}^{k}(U_{2j} \setminus F_{2j})$. By Proposition 10, $w(F_{2j}) \le w(U_{2j})/3$ for all layers, and thus $w(V' \setminus F') \ge \frac{2}{3}w(L_0)$. Using Lemma 12(d), $w(L_0) \ge \max\{w(L_1), w(S)\}$, and thus $w(L_0) \ge w(V')/3$. Thus $w(V' \setminus F') \ge \frac{2}{9}w(V')$ follows.

Case II: $w(L_0) < w(L_1)$. Using the same argument as in the previous case, we obtain $w(V' \setminus F') \ge \frac{2}{3}w(L_1)$. Again using Lemma 12(d), $w(L_1) > w(L_0) \ge w(S)$, and therefore $w(L_1) \ge w(V')/3$, implying $w(V' \setminus F') \ge \frac{2}{9}w(V')$. ◀

4.4 Proof of Theorem 11

Let us first verify part (c):

▶ **Lemma 13.** *Let z be a minimum in-degree vertex in a $\mathcal{T}_7$-free tournament. Then $V_2(z)$ is $\mathcal{T}_5$-free. If $V_2(z) \neq \emptyset$, then $V_3(z) \neq \emptyset$.*

Proof. The claim is trivial if $V_2(z) = \emptyset$. Hence we assume $V_2(z) \neq \emptyset$ in the sequel. We first claim that for every $u \in V_2(z)$ there must exist a $v \in V_3(z)$ with $v \to u$. Indeed, assume that for some u there exists no such v. Then $N(u) \subsetneq V_2(z) = N(z)$ must hold. This is a contradiction to the choice of z with $|N(z)|$ minimum. This already shows that $V_3(z) \neq \emptyset$.

Consider a subset $H \subseteq V_3(z)$ containing at least one vertex v with $v \to u$ for every $u \in V_2$; choose H minimal for containment. If $|H| \geq 3$, then there must be three vertices $v_1, v_2, v_3 \in H$, and three vertices $u_1, u_2, u_3 \in V_2(z)$ such that $v_i \to u_i$ for $i = 1, 2, 3$, while $u_i \to v_j$ if $i \neq j$. Then z and these vertices together form an $S_7 \in \mathcal{T}_7$ subtournament as in Fig. 1(b), a contradiction.

Hence $|H| \leq 2$. For a contradiction, assume $X \subseteq V_2(z)$ forms a $\mathcal{T}_5$-graph ($|X| = 5$). There exists a $v \in H$ with $|\{s \in X : v \to s\}| \geq 3$. We claim that $X \cup \{v, z\} \in \mathcal{T}_7$. Indeed, assume it contains a transitive tournament Y on 5 vertices. Since $X \in \mathcal{T}_5$, $|X \cap Y| \leq 3$; hence $v, z \in Y$ and $|X \cap Y| = 3$. There must be a vertex $t \in X \cap Y$ with $v \to t$, and thus vtz is a directed triangle, a contradiction. ◀

For (a) and (b) of Theorem 11, we show that the 2-in-domination claim implies $\mathcal{T}_5$-freeness:

▶ **Lemma 14.** *Let z be an arbitrary vertex in a $\mathcal{T}_7$-free tournament. For $i \geq 3$, if $V_i(z)$ is 2-in-dominated by $V_{i-1}(z)$, then $V_i(z)$ is $\mathcal{T}_5$-free.*

Proof. Consider a $\mathcal{T}_5$-subtournament X in $V_i(z)$. By 2-in-domination, there must be a $v \in V_{i-1}(z)$ such that $|N(v) \cap X| \geq 3$. Let $s \in V_{i-2}(z)$ be such that $v \to s$. We obtain a contradiction as in the previous proof, showing that $X \cup \{v, s\} \in \mathcal{T}_7$. Indeed, assume that $X \cup \{v, s\}$ has a transitive subtournament Y of size 5. We have $|X \cap Y| \leq 3$ since X is $\mathcal{T}_5$; thus $|X \cap Y| = 3$, and $v, s \in Y$. But then there exists a vertex $t \in X \cap Y \cap N(v)$. We have $s \to t$ because $N(s) \cap V_{i-1}(z) = \emptyset$. Thus stv is a directed triangle. ◀

The proof of Theorem 11 is complete by the following two lemmata, that show that both $V_3(z)$ and $V_4(z)$ are 2-in-dominated by the previous layer.

▶ **Lemma 15.** *For an arbitrary vertex z in a $\mathcal{T}_7$-free tournament T', the set $V_3(z)$ is 2-in-dominated by $V_2(z)$.*

Proof. Let $H \subseteq V_2(z)$ be a minimal set for containment that in-dominates $V_3(z)$. We show that $|H| \leq 2$. Indeed, if $|H| \geq 3$, then again there must be a tournament $S_7 \in \mathcal{T}_7$ as in Fig. 1(b), formed by z, three vertices in $V_2(z)$ and three in $V_3(z)$. ◀

In the sequel, let $\{a, b\} \subseteq V_2(z)$ be a set that 2-in-dominates $V_3(z)$.

▶ **Lemma 16.** *For an arbitrary vertex z in a $\mathcal{T}_7$-free tournament T', the set $V_4(z)$ is 2-in-dominated by $V_3(z)$.*

Proof. For sake of contradiction, assume that any minimal set in $V_3(z)$ that 2-in-dominates $V_4(z)$ has size at least 3. Then there must exists vertices $u_1, u_2, u_3 \in V_3(z)$ and $v_1, v_2, v_3 \in V_4(z)$ such that $v_i \to u_i$ for $i = 1, 2, 3$, while $u_i \to v_j$ if $i \neq j$.

If all $u_1, u_2, u_3 \in N(a)$, then $\{a, u_1, u_2, u_3, v_1, v_2, v_3\}$ forms a tournament in $\mathcal{T}_7$, a contradiction. A similar argument applies for b. We may therefore assume (by possibly renaming the indices) that $u_1 \to a, u_2 \to a, u_3 \to b, a \to u_3$, and $b \to u_2$. See Fig. 2.

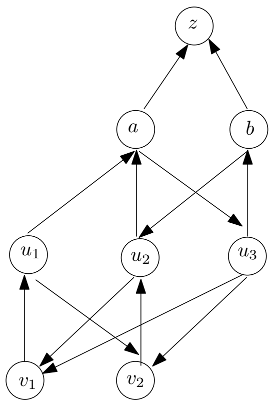


Figure 2 Illustration of the proof of Lemma 16. A few directed edges that are not portrayed are: from z to each one of $\{u_1, u_2, u_3, v_1, v_2\}$ and from each of $\{a, b\}$ to each of $\{v_1, v_2\}$.

Since T' is $\mathcal{T}_7$-free, then every 7-vertex subgraph of $\{z, a, b, u_1, u_2, u_3, v_1, v_2, v_3\}$ must contain a transitive tournament on 5 vertices. Let $Q = \{z, a, b, u_2, u_3\}$, and for $i = 1, 2$, let $Q_i = Q \cup \{u_1, v_i\}$. Let T_i be a transitive tournament on 5 nodes in Q_i.

Notice that Q forms a $\mathcal{T}_5$. Because of this, for $i = 1, 2$ it holds $\{u_1, v_i\} \subseteq T_i$. Furthermore, b, u_3, z cannot all be in T_i since they form a directed triangle; so $\{a, u_2\} \cap T_i \neq \emptyset$. A symmetric argument shows that $\{b, u_3\} \cap T_i \neq \emptyset$ as well.

Now, since either $u_2 \to u_1$ or $u_1 \to u_2$, either $u_2 u_1 v_2$ or $u_1 u_2 v_1$ forms a directed triangle. Thus, $u_2 \notin T_i$ for either $i = 1$ or $i = 2$. For the same i, $a \in T_i$ because of $\{a, u_2\} \cap T_i \neq \emptyset$. Then z cannot be in T_i because $u_1 a z$ forms a directed triangle. Hence $T_i = \{a, b, u_1, u_3, v_i\}$, and this implies that (i) $a \to b$ since $a \to u_3 \to b$, (ii) $u_1 \to u_3$ since $u_1 \to a \to u_3$, and (iii) $u_1 \to b$ since $u_1 \to a \to b$, using (i).

As noted above, $\{u_1, v_1\} \subseteq T_1$. By (ii), $v_1 u_1 u_3$ forms a directed triangle, and by (iii), $v_1 u_1 b$ forms a triangle. Hence, neither u_3 nor b can be contained in T_1, contradicting that $\{b, u_3\} \cap T_1 \neq \emptyset$. This completes the proof of Lemma 16. ◀

5 Connections to Tournament Colouring

We explore a connection to the notion of heroes and celebrities in tournaments studied by Berger, Choromanski, Chudnovsky, Fox, Loebl, Scott, Seymour and Thomassé [3]. Colouring a tournament means partitioning its vertex set into transitive subtournaments; the chromatic number of a tournament is the minimum number of colours needed. A tournament H is called a *hero*, if there exists a constant c_H such that every H-free tournament has chromatic number at most c_H. Further, H is called a *celebrity*, if for some constant $c'_H > 0$, every H-free tournament T has a transitive subtournament of size at least $c'_H |V(T)|$. Clearly, every hero is a celebrity; Berger et al. show that the converse also holds: every celebrity is a hero. Their work gives a characterization of all tournaments that are heros (or equivalently, celebrities).

In this context, our Theorem 4 shows that $\mathcal{T}_7$ collectively form a celebrity set. Further, our constant $c' = 2/9$ seems much better than the constants that could be derived using the techniques of Berger et al. [3]. The set $\mathcal{T}_7$ includes some heros as well as some non-hero tournaments. In contrast, the set $\mathcal{T}_5$ is precisely the set of heros on 5 vertices.

Berger et al.'s [3] characterization rules out the following possible modification of our algorithm to obtain a 2-approximation for the FEEDBACK VERTEX SET in tournaments problem. Instead of $\mathcal{T}_7$, one could use the single tournament ST_6, the unique 6-vertex tournament not containing a transitive subtournament of order 4 [19]. All copies ST_6 can be removed from the input tournament by losing a factor 2 in the approximation ratio only (instead of losing 7/3 by removing copies of subtournaments from $\mathcal{T}_7$). However, according to Berger et al. [3, Thm. 1.2], ST_6 is *not* a hero, and hence there is no hope to prove a version of Theorem 4 for this setting.

References

1 Jeffrey S. Banks. Sophisticated voting outcomes and agenda control. *Soc. Choice Welf.*, 1(4):295–306, 1985.

2 Reuven Bar-Yehuda and Dror Rawitz. On the equivalence between the primal-dual schema and the local ratio technique. *SIAM J. Discrete Math.*, 19(3):762–797, 2005.

3 Eli Berger, Krzysztof Choromanski, Maria Chudnovsky, Jacob Fox, Martin Loebl, Alex Scott, Paul Seymour, and Stéphan Thomassé. Tournaments and colouring. *J. Combin. Theory Ser. B*, 103(1):1–20, 2013.

4 Mao-Cheng Cai, Xiaotie Deng, and Wenan Zang. An approximation algorithm for feedback vertex sets in tournaments. *SIAM J. Comput.*, 30(6):1993–2007 (electronic), 2001.

5 Mao-cheng Cai, Xiaotie Deng, and Wenan Zang. A min-max theorem on feedback vertex sets. *Math. Oper. Res.*, 27(2):361–371, 2002.

6 Irit Dinur and Samuel Safra. On the hardness of approximating minimum vertex cover. *Ann. of Math. (2)*, 162(1):439–485, 2005.

7 Michael Dom, Jiong Guo, Falk Hüffner, Rolf Niedermeier, and Anke Truss. Fixed-parameter tractability results for feedback set problems in tournaments. *J. Discrete Algorithms*, 8(1):76–86, 2010.

8 Guy Even, Joseph (Seffi) Naor, Baruch Schieber, and Madhu Sudan. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20(2):151–174, 1998.

9 Samuel Fiorini, Gwenaël Joret, and Oliver Schaudt. Improved approximation algorithms for hitting 3-vertex paths. In *Proc. IPCO 2016*, volume 9682 of *Lecture Notes Comput. Sci.*, pages 238–249, 2016.

10 Fedor V. Fomin, Serge Gaspers, Daniel Lokshtanov, and Saket Saurabh. Exact algorithms via monotone local search. In *Proc. STOC 2016*, pages 764–775, 2016.

11 Serge Gaspers and Matthias Mnich. Feedback vertex sets in tournaments. *J. Graph Theory*, 72(1):72–89, 2013.

12 Kamal Jain. A factor 2 approximation algorithm for the generalized Steiner network problem. *Combinatorica*, 21(1):39–60, 2001.

13 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations (Proc. Sympos., IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y., 1972)*, pages 85–103. Plenum, New York, 1972.

14 Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within $2-\epsilon$. *J. Comput. System Sci.*, 74(3):335–349, 2008.

15 Michael Krivelevich. On a conjecture of Tuza about packing and covering of triangles. *Discrete Math.*, 142(1):281–286, 1995.

16 Mithilesh Kumar and Daniel Lokshtanov. Faster exact and parameterized algorithm for feedback vertex set in tournaments. In *Proc. STACS 2016*, volume 47 of *Leibniz Int. Proc. Informatics*, pages 49:1–49:13, 2016.

17 Lap Chi Lau, R. Ravi, and Mohit Singh. *Iterative methods in combinatorial optimization.* Cambridge Texts in Applied Mathematics. Cambridge University Press, New York, 2011.

18 John W. Moon. On maximal transitive subtournaments. *Proc. Edinburgh Math. Soc. (2)*, 17:345–349, 1970/71.

19 Adolfo Sanchez-Flores. On tournaments free of large transitive subtournaments. *Graphs Comb.*, 14(2):181–200, 1998.

20 Prashant Sasatte. Improved approximation algorithm for the feedback set problem in a bipartite tournament. *Oper. Res. Lett.*, 36(5):602–604, 2008.

21 Paul D. Seymour. Packing directed circuits fractionally. *Combinatorica*, 15(2):281–288, 1995.

22 Ewald Speckenmeyer. On feedback problems in digraphs. In *Proc. WG 1989*, volume 411 of *Lecture Notes Comput. Sci.*, pages 218–231. Springer, 1990.

23 Anke van Zuylen. Linear programming based approximation algorithms for feedback set problems in bipartite tournaments. *Theor. Comput. Sci.*, 412(23):2556–2561, 2011.

Scheduling Distributed Clusters of Parallel Machines: Primal-Dual and LP-based Approximation Algorithms*

Riley Murray[1], Megan Chao[2], and Samir Khuller[3]

1 Department of Industrial Engineering & Operations Research, University of California, Berkeley, USA
rjmurray@berkeley.edu

2 Department of Electrical Engineering & Computer Science, Massachusetts Institute of Technology, Cambridge, USA
megchao@mit.edu

3 Department of Computer Science, University of Maryland, College Park, USA
samir@cs.umd.edu

Abstract

The Map-Reduce computing framework rose to prominence with datasets of such size that dozens of machines on a single cluster were needed for individual jobs. As datasets approach the exabyte scale, a single job may need distributed processing not only on multiple machines, but on multiple *clusters*. We consider a scheduling problem to minimize weighted average completion time of n jobs on m distributed clusters of parallel machines. In keeping with the scale of the problems motivating this work, we assume that (1) each job is divided into m "subjobs" and (2) distinct subjobs of a given job may be processed concurrently.

When each cluster is a single machine, this is the NP-Hard *concurrent open shop* problem. A clear limitation of such a model is that a serial processing assumption sidesteps the issue of how different tasks of a given subjob might be processed in parallel. Our algorithms explicitly model clusters as pools of resources and effectively overcome this issue.

Under a variety of parameter settings, we develop two constant factor approximation algorithms for this problem. The first algorithm uses an LP relaxation tailored to this problem from prior work. This LP-based algorithm provides strong performance guarantees. Our second algorithm exploits a surprisingly simple mapping to the special case of one machine per cluster. This mapping-based algorithm is combinatorial and extremely fast. These are the first constant factor approximations for this problem.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases approximation algorithms, distributed computing, machine scheduling, LP relaxations, primal-dual algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.68

1 Introduction

It is becoming increasingly impractical to store full copies of large datasets on more than one data center [7]. As a result, the data for a single job may be located not on multiple

* All authors conducted this work at the University of Maryland, College Park. This work was made possible by the National Science Foundation, REU Grant CNS 1262805, and the Winkler Foundation. This work was also partially supported by NSF Grant CCF 1217890.

machines, but on multiple *clusters* of machines. To maintain fast response-times and avoid excessive network traffic, it is advantageous to perform computation for such jobs in a completely distributed fashion [8]. In addition, commercial platforms such as AWS Lambda and Microsoft's Azure Service Fabric are demonstrating a trend of centralized cloud computing frameworks in which the user manages neither data flow nor server allocation [1, 11]. In view of these converging issues, the following scheduling problem arises:

If computation is done locally to avoid excessive network traffic, how can individual clusters on the broader grid coordinate schedules for maximum throughput?

This was precisely the motivation for Hung, Golubchik, and Yu in their 2015 ACM Symposium on Cloud Computing paper [8]. Hung et al. modeled each cluster as having an arbitrary number of identical parallel machines, and choose an objective of average job completion time. As such a problem generalizes the NP-Hard concurrent open shop problem, they proposed a heuristic approach. Their heuristic (called "SWAG") runs in $O(n^2m)$ time and performed well on a variety of data sets. Unfortunately, SWAG offers poor worst-case performance, as we show in Section 5.

Our contributions to this problem are to extend the model considered by Hung et al. and to introduce the first constant-factor approximation algorithms for this general problem. Our extensions of Hung et al.'s model are (1) to allow different machines within the same cluster to operate at different speeds, (2) to incorporate pre-specified "release times" (times before which a subjob cannot be processed), and (3) to support *weighted* average job completion time. We present two algorithms for the resulting problem. Our combinatorial algorithm exploits a surprisingly simple mapping to the special case of one machine per cluster, where the problem can be approximated in $O(n^2 + nm)$ time. We also present an LP-rounding approach with strong performance guarantees. E.g., a 2-approximation when machines are of unit speed and subjobs are divided into equally sized (but not necessary *unit*) tasks.

1.1 Formal Problem Statement

▶ **Definition 1** (Concurrent Cluster Scheduling).

- There is a set M of m clusters, and a set N of n jobs. For each job $j \in N$, there is a set of m "subjobs" (one for each cluster).
- Cluster $i \in M$ has m_i parallel machines, and machine ℓ in cluster i has speed $v_{\ell i}$. Without loss of generality, assume $v_{\ell i}$ is decreasing in ℓ.[1]
- The i^{th} subjob for job j is specified by a set of tasks to be performed by machines in cluster i, denote this set of tasks T_{ji}. For each task $t \in T_{ji}$, we have an associated processing time p_{jit} (again w.l.o.g., assume p_{jit} is decreasing in t). We will frequently refer to "the subjob of job j at cluster i" as "subjob (j, i)."
- Different subjobs of the same job may be processed concurrently on different clusters.
- Different tasks of the same subjob may be processed concurrently on different machines within the same cluster.
- A subjob is complete when all of its tasks are complete, and a job is complete when all of its subjobs are complete. We denote a job's completion time by "C_j".
- The objective is to minimize weighted average job completion time (job j has weight w_j).
- For the purposes of computing approximation ratios, it is equivalent to minimize $\sum w_j C_j$. We work with this equivalent objective throughout this paper.

[1] Where we write "decreasing", we mean "non-increasing." Where we write "increasing", we mean "non-decreasing".

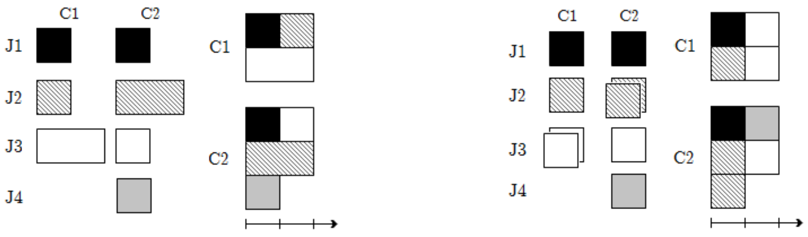


Figure 1 Two examples of our scheduling model. **Left**: Our baseline example. There are 4 jobs and 2 clusters. Cluster 1 has 2 identical machines, and cluster 2 has 3 identical machines. Note that job 4 has no subjob for cluster 1 (this is permitted within our framework). In this case every subjob has at most one task. **Right**: Our baseline example with a more general subjob framework : subjob (2,2) and subjob (3,1) both have two tasks. The tasks shown are unit length, but our framework *does not* require that subjobs be divided into equally sized tasks.

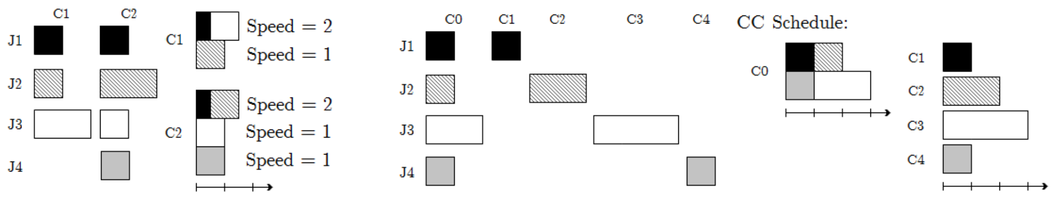


Figure 2 Two additional examples of our model. **Left**: Our baseline example, with variable machine speeds. Note that the benefit of high machine speeds is only realized for tasks assigned to those machines in the final schedule. **Right**: A problem with the peculiar structure that (1) all clusters but one have a single machine, and (2) most clusters have non-zero processing requirements for only a single job. We will use such a device for the total weighted lateness reduction in Section 6.

A machine is said to operate at *unit speed* it if can complete a task with processing requirement "p" in p units of time. More generally, a machine with speed "v" ($v \geq 1$) processes the same task in p/v units of time. Machines are said to be *identical* if they are all of unit speed, and *uniform* if they differ only in speed.

In accordance with Graham et al.'s $\alpha|\beta|\gamma$ taxonomy for scheduling problems [6] we take $\alpha = CC$ to refer to the concurrent cluster environment, and denote our problem by $CC||\sum w_j C_j$.[2] Optionally, we may associate a release time r_{ji} to every subjob. If any subjobs are released after time zero, we write $CC|r|\sum w_j C_j$.

1.1.1 Example Problem Instances

We now illustrate our model with several examples (see Figures 1 and 2). The tables at left have rows labeled to identify jobs, and columns labeled to identify clusters; each entry in these tables specifies the processing requirements for the corresponding subjob. The diagrams to the right of these tables show how the given jobs might be scheduled on clusters with the indicated number of machines.

[2] A problem $\alpha|\beta|\gamma$ implies a particular environment α, objective function γ, and optional constraints β.

1.2 Related Work

Concurrent cluster scheduling subsumes many fundamental machine scheduling problems. For example, if we restrict ourselves to a single cluster (i.e. $m = 1$) we can schedule a set of jobs on a bank of identical parallel machines to minimize makespan ($C_{\max}$) or total weighted completion time ($\sum w_j C_j$). With a more clever reduction, we can even minimize *total weighted lateness* ($\sum w_j L_j$) on a bank of identical parallel machines (see Section 6). Alternatively, with $m > 1$ but $\forall i \in M, m_i = 1$, our problem reduces to the well-studied "concurrent open shop" problem.

Using Graham et al.'s taxonomy, the concurrent open shop problem is written as $PD||\sum w_j C_j$. Three groups [3, 4, 9] independently discovered an LP-based 2-approximation for $PD||\sum w_j C_j$ using the work of Queyranne [13]. The linear program in question has an exponential number of constraints, but can still be solved in polynomial time with a variant of the Ellipsoid method. Our "strong" algorithm for concurrent cluster scheduling refines the techniques contained therein, as well as those of Schulz [14, 15] (see Section 4).

Mastrolilli et al. [10] developed a primal-dual algorithm for $PD||\sum w_j C_j$ that does not use LP solvers. "MUSSQ"[3] is significant for both its speed and the strength of its performance guarantee : it achieves an approximation ratio of 2 in only $O(n^2 + nm)$ time. Although MUSSQ does not require an LP solver, its proof of correctness is based on the fact that it finds a feasible solution to the dual a particular linear program. Our "fast" algorithm for concurrent cluster scheduling uses MUSSQ as a subroutine (see Section 5).

Hung, Golubchik, and Yu [8] presented a framework designed to improve scheduling across geographically distributed data centers. The scheduling framework had a centralized scheduler (which determined a job ordering) and local dispatchers which carried out a schedule consistent with the controllers job ordering. Hung et al. proposed a particular algorithm for the controller called "SWAG." SWAG performed well in a wide variety of simulations where each data center was assumed to have the same number of identical parallel machines. We adopt a similar framework to Hung et al., but we show in Section 5.1 that SWAG has no constant-factor performance guarantee.

1.3 Paper Outline & Algorithmic Results

Although only one of our algorithms requires *solving* a linear program, both algorithms use the same linear program in their proofs of correctness; we introduce this linear program in Section 2 before discussing either algorithm. Section 3 establishes how an ordering of jobs can be processed to completely specify a schedule. This is important because the complex work in both of our algorithms is to generate an ordering of jobs for each cluster.

Section 4 introduces our "strong" algorithm: CC-LP. CC-LP can be applied to any instance of concurrent cluster scheduling, including those with non-zero release times r_{ji}. A key in CC-LP's strong performance guarantees lay in the fact that it allows different permutations of subjobs for different clusters. By providing additional structure to the problem (but while maintaining a generalization of concurrent open shop) CC-LP becomes a 2-approximation. This is significant because it is UGC-Hard to approximate concurrent open shop (and by extension, our problem) with ratio $2 - \epsilon$ for any $\epsilon > 0$ [2].

Our combinatorial algorithm ("CC-TSPT") is presented in Section 5. The algorithm is fast, provably accurate, and has the interesting property that it can schedule all clusters

[3] A permutation of the author's names: Mastrolilli, Queyranne, Schulz, Svensson, and Uhan.

using the same permutation of jobs.[4] After considering CC-TSPT in the general case, we show how fine-grained approximation ratios can be obtained in the "fully parallelizable" setting of Zhang et al. [17]. We conclude with an extension of CC-TSPT that maintains performance guarantees while offering improved empirical performance.

The following table summarizes our results for approximation ratios. For compactness, condition Id refers to identical machines (i.e. $v_{\ell i}$ constant over ℓ), condition A refers to $r_{ji} \equiv 0$, and condition B refers to p_{jit} constant over $t \in T_{ji}$.

	(Id, A, B)	$(Id, \neg A, B)$	$(Id, A, \neg B)$	$(Id, \neg A, \neg B)$	$(\neg Id, A)$	$(\neg Id, \neg A)$
CC-LP	2	3	3	4	$2 + R$	$3 + R$
CC-TSPT	3	–	3	–	$2 + R$	–

The term R is the maximum over i of R_i, where R_i is the ratio of fastest machine to *average* machine speed at cluster i.

The most surprising of all of these results is that our scheduling algorithms are remarkably simple. The first algorithm solves an LP, and then the scheduling can be done easily on each cluster. The second algorithm is again a rather surprising simple reduction to the case of one machine per cluster (the well understood concurrent open shop problem) and yields a simple combinatorial algorithm. The proof of the approximation guarantee is somewhat involved however.

In addition to algorithmic results, we demonstrate how our problem subsumes that of minimizing total weighted lateness on a bank of identical parallel machines (see Section 6). Section 7 provides additional discussion and highlights our more novel technical contributions.

2 The Core Linear Program

Our linear program has an unusual form. Rather than introduce it immediately, we conduct a brief review of prior work on similar LP's. All the LP's we discuss in this paper have objective function $\sum w_j C_j$, where C_j is a decision variable corresponding to the completion time of job j, and w_j is a weight associated with job j.

For the following discussion only, we adopt the notation in which job j has processing time p_j. In addition, if multiple machine problems are discussed, we will say that there are m *such machines (possibly with speeds $s_i, i \in \{1, \ldots, \mathsf{m}\}$).*

The earliest appearance of a similar linear program comes from Queyranne [13]. In his paper, Queyranne presents an LP relaxation for sequencing n jobs on a single machine where all constraints are of the form $\sum_{j \in S} p_j C_j \geq \frac{1}{2}\left[\left(\sum_{j \in S} p_j\right)^2 + \sum_{j \in S} p_j^2\right]$ where S is an arbitrary subset of jobs. Once a set of optimal $\{C_j^\star\}$ is found, the jobs are scheduled in increasing order of $\{C_j^\star\}$. These results were primarily theoretical, as it was known at his time of writing that sequencing n jobs on a single machine to minimize $\sum w_j C_j$ can be done optimally in $O(n \log n)$ time.

Queyranne's constraint set became particularly useful for problems with *coupling* across distinct machines (as occurs in concurrent open shop). Four separate groups [3, 4, 9, 10] saw

[4] We call such schedules "single-σ schedules." As we will see later on, CC-TSPT serves as a constructive proof of existence of near-optimal single-σ schedules for all instances of $CC||\sum w_j C_j$, *including* those instances for which single-σ schedules are strictly sub-optimal. This is addressed in Section 7.

this and used the following LP in a 2-approximation for concurrent open shop scheduling.

$$\text{(LP0)} \quad \min \sum_{j \in N} w_j C_j \quad \text{s.t.} \quad \textstyle\sum_{j \in S} p_{ji} C_j \geq \frac{1}{2}\left[\left(\sum_{j \in S} p_{ji}\right)^2 + \left(\sum_{j \in S} p_{ji}^2\right)\right] \ \forall \substack{S \subseteq N \\ i \in M}$$

In view of its tremendous popularity, we sometimes refer to the linear program above as the *canonical relaxation* for concurrent open shop.

Andreas Schulz's Ph.D. thesis developed Queyranne's constraint set in greater depth [14]. As part of his thesis, Schulz considered scheduling n jobs on m identical parallel machines with constraints of the form $\sum_{j \in S} p_j C_j \geq \frac{1}{2\mathsf{m}}\left(\sum_{j \in S} p_j\right)^2 + \frac{1}{2}\sum_{j \in S} p_j^2$. In addition, Schulz showed that the constraints $\sum_{j \in S} p_j C_j \geq \left[2\sum_{i=1}^{\mathsf{m}} s_i\right]^{-1}\left[\left(\sum_{j \in S} p_j\right)^2 + \sum_{j \in S} p_j^2\right]$ are satisfied by any schedule of n jobs on m uniform machines. In 2012, Schulz refined the analysis for several of these problems [15]. For constructing a schedule from the optimal $\{C_j^\star\}$, Schulz considered scheduling jobs by increasing order of $\{C_j^\star\}$, $\{C_j^\star - p_j/2\}$, and $\{C_j^\star - p_j/(2\mathsf{m})\}$.

2.1 Statement of LP1

The model we consider allows for more fine-grained control of the job structure than is indicated by the LP relaxations above. Inevitably, this comes at some expense of simplicity in LP formulations. In an effort to simplify notation, we define the following constants, and give verbal interpretations for each.

$$\mu_i \doteq \textstyle\sum_{\ell=1}^{m_i} v_{\ell i} \qquad q_{ji} \doteq \min\{|T_{ji}|, m_i\} \qquad \mu_{ji} \doteq \sum_{\ell=1}^{q_{ji}} v_{\ell i} \qquad p_{ji} \doteq \sum_{t \in T_{ji}} p_{jit} \tag{1}$$

From these definitions, μ_i is the processing power of cluster i. For subjob (j, i), q_{ji} is the maximum number of machines that could process the subjob, and μ_{ji} is the maximum processing power than can be brought to bear on the same. Lastly, p_{ji} is the total processing requirement of subjob (j, i). In these terms, the core linear program, LP1, is as follows.

$$\begin{aligned}
&\text{(LP1)} \ \min \textstyle\sum_{j \in N} w_j C_j \\
&\textit{s.t.} \quad (1A) \quad \textstyle\sum_{j \in S} p_{ji} C_j \geq \frac{1}{2}\left[\left(\sum_{j \in S} p_{ji}\right)^2 / \mu_i + \sum_{j \in S} p_{ji}^2 / \mu_{ji}\right] \qquad \forall S \subseteq N, i \in M \\
&\qquad\quad (1B) \quad C_j \geq p_{jit}/v_{1i} + r_{ji} \qquad \forall i \in M,\ j \in N,\ t \in T_{ji} \\
&\qquad\quad (1C) \quad C_j \geq p_{ji}/\mu_{ji} + r_{ji} \qquad \forall j \in N,\ i \in M
\end{aligned}$$

Constraints $(1A)$ are more carefully formulated versions of the polyhedral constraints introduced by Queyranne [13] and developed by Schulz [14]. The use of μ_{ji} term is new and allows us to provide stronger performance guarantees for our framework where subjobs are composed of *sets* of tasks. As we will see, this term is one of the primary factors that allows us to parametrize results under varying machine speeds in terms of maximum to *average* machine speed, rather than maximum to *minimum* machine speed. Constraints $(1B)$ and $(1C)$ are simple lower bounds on job completion time.

The majority of this section is dedicated to proving that LP1 is a valid relaxation of $CC|r|\sum w_j C_j$. Once this is established, we prove the that LP1 can be solved in polynomial time by providing a separation oracle with use in the Ellipsoid method. Both of these proofs use techniques established in Schulz's Ph.D. thesis [14].

2.2 Proof of LP1's Validity

The lemmas below establish the basis for both of our algorithms. Lemma 2 generalizes an inequality used by Schulz [14]. Lemma 3 relies on Lemma 2 and cites an inequality mentioned in the preceding section (and proven by Queyranne [13]).

▶ **Lemma 2.** *Let $\{a_1, \dots a_z\}$ be a set of non-negative real numbers. We assume that $k \le z$ of them are positive. Let b_i be a set of decreasing positive real numbers. Then*

$$\sum_{i=1}^{z} a_i^2/b_i \ge \left(\sum_{i=1}^{z} a_i\right)^2 / \left(\sum_{i=1}^{k} b_i\right).$$

The proof (found in the full version of this paper, [12]) cites the AM-GM inequality and proceeds with induction from $z = k = 2$.

▶ **Lemma 3** (Validity Lemma). *Every feasible schedule for an instance I of $CC|r|\sum w_j C_j$ has completion times that define a feasible solution to LP1(I).*

Proof. As constraints ($1B$) and ($1C$) are clear lower bounds on job completion time, it suffices to show the validity of constraint ($1A$). Thus, let S be a non-empty subset of N, and fix an arbitrary but feasible schedule "F" for I.

Define C^F_{ji} as the completion time of subjob (j, i) under schedule F. Similarly, define $C^F_{ji\ell}$ as the first time at which tasks of subjob (j, i) scheduled on machine ℓ of cluster i are finished. Lastly, define p^ℓ_{ji} as the total processing requirement of job j scheduled on machine ℓ of cluster i. Note that by construction, we have $C^F_{ji} = \max_{\ell \in \{1,\dots,m_i\}} C^F_{ji\ell}$ and $C^F_j = \max_{i \in M} C^F_{ji}$. Since $p_{ji} = \sum_{\ell=1}^{m_i} p^\ell_{ji}$, we can rather innocuously write

$$\textstyle\sum_{j\in S} p_{ji} C^F_{ji} = \sum_{j\in S} \left[\sum_{\ell=1}^{m_i} p^\ell_{ji}\right] C^F_{ji}. \tag{2}$$

But using $C^F_{ji} \ge C^F_{ji\ell}$, we can lower-bound $\sum_{j\in S} p_{ji} C^F_{ji}$. Namely,

$$\textstyle\sum_{j\in S} p_{ji} C^F_{ji} \ge \sum_{j\in S} \sum_{\ell=1}^{m_i} p^\ell_{ji} C^F_{ji\ell} = \sum_{\ell=1}^{m_i} v_{\ell i} \sum_{j\in S} \left[p^\ell_{ji}/v_{\ell i}\right] C^F_{ji\ell} \tag{3}$$

The next inequality uses a bound on $\sum_{j\in S} \left[p^\ell_{ji}/v_{\ell i}\right] C^F_{ji\ell}$ proven by Queyranne [13] for any subset S of N jobs with processing times $\left[p^\ell_{ji}/v_{\ell i}\right]$ to be scheduled on a single machine.[5]

$$\textstyle\sum_{j\in S} \left[p^\ell_{ji}/v_{\ell i}\right] C^F_{ji\ell} \ge \frac{1}{2}\left[\left(\sum_{j\in S} \left[p^\ell_{ji}/v_{\ell i}\right]\right)^2 + \sum_{j\in S} \left(\left[p^\ell_{ji}/v_{\ell i}\right]\right)^2\right] \tag{4}$$

Combining inequalities (3) and (4), we have the following.

$$\begin{align} \textstyle\sum_{j\in S} p_{ji} C^F_{ji} &\ge \frac{1}{2} \textstyle\sum_{\ell=1}^{m_i} v_{\ell i} \left[\left(\sum_{j\in S} \left[p^\ell_{ji}/v_{\ell i}\right]\right)^2 + \sum_{j\in S} \left(\left[p^\ell_{ji}/v_{\ell i}\right]\right)^2\right] \tag{5}\\ &\ge \frac{1}{2} \left[\textstyle\sum_{\ell=1}^{m_i} \left(\sum_{j\in S} p^\ell_{ji}\right)^2 / v_{\ell i} + \sum_{j\in S} \sum_{\ell=1}^{m_i} \left(p^\ell_{ji}\right)^2 / v_{\ell i}\right] \tag{6} \end{align}$$

Next, we apply Lemma 2 to the right hand side of inequality (6) a total of $|S| + 1$ times.

$$\textstyle\sum_{\ell=1}^{m_i} \left(\sum_{j\in S} p^\ell_{ji}\right)^2 / v_{\ell i} \ge \left(\sum_{\ell=1}^{m_i} \sum_{j\in S} p^\ell_{ji}\right)^2 / \sum_{\ell=1}^{m_i} v_{\ell i} = \left(\sum_{j\in S} p_{ji}\right)^2 / \mu_i \tag{7}$$

$$\textstyle\sum_{\ell=1}^{m_i} \left(p^\ell_{ji}\right)^2 / v_{\ell i} \ge \left(\sum_{\ell=1}^{m_i} p^\ell_{ji}\right)^2 / \sum_{\ell=1}^{q_{ji}} v_{\ell i} = p^2_{ji}/\mu_{ji} \quad \forall\, j \in S \tag{8}$$

[5] Here, our machine is machine ℓ on cluster i.

Citing $C_j^F \geq C_{ji}^F$, we arrive at the desired result.

$$\textstyle\sum_{j \in S} p_{ji} C_j^F \geq \frac{1}{2}\left[\left(\sum_{j \in S} p_{ji}\right)^2 / \mu_i + \sum_{j \in S} p_{ji}^2 / \mu_{ji}\right] \qquad \text{“constraint } (1A)\text{”} \tag{9}$$

◀

2.3 Theoretical Complexity of LP1

As the first of our two algorithms requires solving LP1 directly, we need to address the fact that LP1 has $m \cdot (2^n - 1) + n$ constraints. Luckily, it is still possible to such solve linear programs in polynomial time with the Ellipsoid method; we introduce the following separation oracle for this purpose.

▶ **Definition 4** (Oracle LP1). Define the *violation*

$$V(S, i) = \frac{1}{2}\left[\left(\textstyle\sum_{j \in S} p_{ji}\right)^2 / \mu_i + \sum_{j \in S} p_{ji}^2 / \mu_{ji}\right] - \textstyle\sum_{j \in S} p_{ji} C_j \tag{10}$$

Let $\{C_j\} \in \mathbb{R}^n$ be a *potentially* feasible solution to LP1. Let σ_i denote the ordering when jobs are sorted in increasing order of $C_j - p_{ji}/(2\mu_{ji})$. Find the most violated constraint in $(1A)$ for $i \in M$ by searching over $V(S_i, i)$ for S_i of the form $\{\sigma_i(1), \ldots, \sigma_i(j-1), \sigma_i(j)\}$, $j \in \{1, \ldots, n\}$. If any of maximal $V(S_i^*, i) > 0$, then return (S_i^*, i) as a violated constraint for $(1A)$. Otherwise, check the remaining n constraints ($(1B)$ and $(1C)$) directly in linear time.

For fixed i, Oracle-LP1 finds the subset of jobs that maximizes “violation” for cluster i. That is, Oracle-LP1 finds S_i^* such that $V(S_i^*, i) = \max_{S \subset N} V(S, i)$. We prove the correctness of Oracle-LP1 by establishing a necessary and sufficient condition for a job j to be in S_i^*.

▶ **Lemma 5.** *For $\mathbb{P}_i(A) \doteq \sum_{j \in A} p_{ji}$, we have $x \in S_i^* \Leftrightarrow C_x - p_{xi}/(2\mu_{xi}) \leq \mathbb{P}_i(S_i^*)/\mu_i$.*

Proof. For given S (not necessarily equal to S_i^*), it is useful to express $V(S, i)$ in terms of $V(S \cup x, i)$ or $V(S \setminus x, i)$ (depending on whether $x \in S$ or $x \in N \setminus S$). Without loss of generality, we restrict our search to $S : x \in S \Rightarrow p_{x,i} > 0$.

Suppose $x \in S$. By writing $\mathbb{P}_i(S) = \mathbb{P}_i(S \setminus x) + \mathbb{P}_i(x)$, and similarly decomposing the sum $\sum_{j \in S} p_{ji}^2/(2\mu_{ji})$, one can show the following.

$$V(S, i) = V(S \setminus x, i) + p_{xi}\left(\frac{1}{2}\left(\frac{2\mathbb{P}_i(S) - p_{xi}}{\mu_i} + \frac{p_{xi}}{\mu_{xi}}\right) - C_x\right) \tag{11}$$

Now suppose $x \in N \setminus S$. In the same strategy as above (this time writing $\mathbb{P}_i(S) = \mathbb{P}_i(S \cup x) - \mathbb{P}_i(x)$), one can show that

$$V(S, i) = V(S \cup x, i) + p_{xi}\left(C_x - \frac{1}{2}\left(\frac{2\mathbb{P}_i(S) + p_{xi}}{\mu_i} + \frac{p_{xi}}{\mu_{xi}}\right)\right). \tag{12}$$

Note that Equations (11) and (12) hold for all S, including $S = S_i^*$. Turning our attention to S_i^*, we see that $x \in S_i^*$ implies that the second term in Equation (11) is non-negative, i.e.

$$C_x - p_{xi}/(2\mu_{xi}) \leq \left(2\mathbb{P}_i(S_i^*) - p_{xi}\right)/(2\mu_i) < \mathbb{P}_i(S_i^*)/\mu_i. \tag{13}$$

Similarly, $x \in N \setminus S_i^*$ implies the second term in Equation (12) is non-negative.

$$C_x - p_{xi}/(2\mu_{xi}) \geq \left(2\mathbb{P}_i(S_i^*) + p_{xi}\right)/(2\mu_i) \geq \mathbb{P}_i(S_i^*)/\mu_i \tag{14}$$

It follows that $x \in S_i^*$ iff $C_x - p_{xi}/(2\mu_{xi}) < \mathbb{P}_i(S_i^*)/\mu_i$. ◀

Given Lemma 5, It is easy to verify that sorting jobs in increasing order of $C_x - p_{xi}/(2\mu_{xi})$ to define a permutation σ_i guarantees that S_i^* is of the form $\{\sigma_i(1), \ldots, \sigma_i(j-1), \sigma_i(j)\}$ for some $j \in N$. This implies that for fixed i, Oracle-LP1 finds S_i^* in $O(n\log(n))$ time. This procedure is executed once for each cluster, leaving the remaining n constraints in $(1B)$ and $(1C)$ to be verified in linear time. Thus Oracle-LP1 runs in $O(mn\log(n))$ time.

By the equivalence of separation and optimization, we have proven the following theorem:

▶ **Theorem 6.** *LP1(I) is a valid relaxation of $I \in \Omega_{CC}$, and is solvable in polynomial time.*

As was explained in the beginning of this section, linear programs such as those in [3, 4, 9, 13, 14, 15] are processed with an appropriate sorting of the optimal decision variables $\{C_j^\star\}$. It is important then to have bounds on job completion times for a particular ordering of jobs. We address this next in Section 3, and reserve our first algorithm for Section 4.

3 List Scheduling from Permutations

The complex work in both of our proposed algorithms is to generate a *permutation* of jobs. The procedure below takes such a permutation and uses it to determine start times, end times, and machine assignments for every task of every subjob.

List-LPT: Given a single cluster with m_i machines and a permutation of jobs σ, introduce $\text{List}(a,i) \doteq (p_{ai1}, p_{ai2}, \ldots, p_{ai|T_{ai}|})$ as an ordered set of tasks belonging to subjob (a,i), ordered by longest processing time first. Now define $\text{List}(\sigma) \doteq \text{List}(\sigma(1),i) \oplus \text{List}(\sigma(2),i) \oplus \cdots \oplus \text{List}(\sigma(n),i)$, where $\oplus$ is the concatenation operator.

Place the tasks of $\text{List}(\sigma)$ in order- from the largest task of subjob $(\sigma(1),i)$, to the smallest task of subjob $(\sigma(n),i)$. When placing a particular task, assign it whichever machine and start time results in the task being completed as early as possible (without moving any tasks which have already been placed). Insert idle time (on all m_i machines) as necessary if this procedure would otherwise start a job before its release time.

The following Lemma is essential to bound the completion time of a set of jobs processed by List-LPT. The proof is adapted from Gonzalez et al. [5].

▶ **Lemma 7.** *Suppose n jobs are scheduled on cluster i according to List-LPT(σ). Then for $\bar{v}_i \doteq \mu_i/m_i$, the completion time of subjob $(\sigma(j),i)$ (denoted $C_{\sigma(j)i}$) satisfies*

$$C_{\sigma(j)i} \leq \max_{1\leq k\leq j} r_{\sigma(k)i} + p_{\sigma(j)i1}/\bar{v}_i + \left(\textstyle\sum_{k=1}^{j} p_{\sigma(k)i} - p_{\sigma(j)i1}\right)/\mu_i \tag{15}$$

Proof. For now, assume all jobs are released at time zero. Let the task of subjob $(\sigma(j),i)$ to finish last be denoted t^*. If t^* is not the task in $T_{\sigma(j)i}$ with least processing time, then construct a new set $T'_{\sigma(j)i} = \{t : p_{\sigma(j)it^*} \leq p_{\sigma(j)it}\} \subset T_{\sigma(j)i}$. Because the tasks of subjob $(\sigma(j),i)$ were scheduled by List-LPT (i.e. longest-processing-time-first), the sets of potential start times and machines for task t^* (and hence the set of potential completion times for task t^*) are the same regardless of whether subjob $(\sigma(j),i)$ consisted of tasks $T_{\sigma(j)i}$ or the subset $T'_{\sigma(j)i}$. Accordingly, reassign $T_{\sigma(j)i} \leftarrow T'_{\sigma(j)i}$ without loss of generality.

Let D_ℓ^j denote the total demand for machine ℓ (on cluster i) once all tasks of subjobs $(\sigma(1),i)$ through $(\sigma(j-1),i)$ and all tasks in the set $T_{\sigma(j)i} \setminus \{t^*\}$ are scheduled. Using the fact that $C_{\sigma(j)i}v_{\ell i} \leq (D_\ell^j + p_{\sigma(j)it^*})\forall \ell \in \{1,\ldots,m_i\}$, sum the left and right and sides over ℓ. This implies $C_{\sigma(j)i}\left(\sum_{\ell=1}^{m_i} v_{\ell i}\right) \leq m_i p_{\sigma(j)it^*} + \sum_{\ell=1}^{m_i} D_\ell^j$. Dividing by the sum of machine speeds and using the definition of μ_i yields

$$C_{\sigma(j)i} \;\leq\; m_i p_{\sigma(j)it^*}/\mu_i + \textstyle\sum_{\ell=1}^{m_i} D_\ell^j/\mu_i \;\leq\; p_{\sigma(j)i1}/\bar{v}_i + \left(\sum_{k=1}^{j} p_{\sigma(k)i} - p_{\sigma(j)i1}\right)/\mu_i \tag{16}$$

where we estimated $p_{\sigma(j)it^*}$ upward by $p_{\sigma(j)i1}$. Inequality (16) completes our proof in the case when $r_{ji} \equiv 0$. The alternative case is addressed in the full version of this paper. ◀

Lemma 7 is cited directly in the proof of Theorem 8 and Lemma 13. Lemma 7 is used implicitly in the proofs of Theorems 9, 10, and 15.

4 An LP-based Algorithm

In this section we show how LP1 can be used to construct near optimal schedules for concurrent cluster scheduling both when $r_{ji} \equiv 0$ and when some $r_{ji} > 0$. Although solving LP1 is somewhat involved, the algorithm itself is quite simple:

Algorithm CC-LP: Let $I = (T, r, w, v)$ denote an instance of $CC|r|\sum w_j C_j$. Use the optimal solution $\{C_j^\star\}$ of LP1(I) to define m permutations $\{\sigma_i : i \in M\}$ which sort jobs in increasing order of $C_j^\star - p_{ji}/(2\mu_{ji})$. For each cluster i, execute List-LPT(σ_i).

Each theorem in this section can be characterized by how various assumptions help us cancel an additive term[6] in an upper bound for the completion time of an arbitrary subjob (x, i). Theorem 8 is the most general, while Theorem 10 is perhaps the most surprising.

4.1 CC-LP for Uniform Machines

▶ **Theorem 8.** *Let $\hat{C}_j$ be the completion time of job j using algorithm CC-LP, and let R be as in Section 1.3. If $r_{ji} \equiv 0$, then $\sum_{j \in N} w_j \hat{C}_j \leq (2 + R)\,OPT$. Otherwise, $\sum_{j \in N} w_j \hat{C}_j \leq (3 + R)\,OPT$.*

Proof. For $y \in \mathbb{R}$, define $y^+ = \max\{y, 0\}$. Now let $x \in N$ be arbitrary, and let $i \in M$ be such that $p_{xi} > 0$ (but otherwise arbitrary). Define t^* as the last task of job x to complete on cluster i, and let j_i be such that $\sigma_i(j_i) = x$. Lastly, denote the optimal LP solution $\{C_j\}$.[7] Because $\{C_j\}$ is a feasible solution to LP1, constraint $(1A)$ implies the following (set $S_i = \{\sigma_i(1), \ldots, \sigma_i(j_i - 1), x\}$)

$$\frac{\left(\sum_{k=1}^{j_i} p_{\sigma_i(k)i}\right)^2}{2\mu_i} \leq \sum_{k=1}^{j_i} p_{\sigma_i(k)i}\left(C_{\sigma_i(k)} - \frac{p_{\sigma_i(k)i}}{2\mu_{\sigma_i(k)i}}\right) \leq \left(C_x - \frac{p_{xi}}{2\mu_{xi}}\right)\sum_{k=1}^{j_i} p_{\sigma_i(k)i} \tag{17}$$

which in turn implies $\sum_{k=1}^{j_i} p_{\sigma_i(k)i}/\mu_i \leq 2C_x - p_{xi}/\mu_{xi}$.

If all subjobs are released at time zero, then we can combine this with Lemma 7 and the fact that $p_{xit^*} \leq p_{xi} = \sum_{t \in T_{xi}} p_{xit}$ to see the following (the transition from the first inequality the second inequality uses $C_x \geq p_{xit^*}/v_{1i}$ and $R_i = v_{1i}/\bar{v}_i$).

$$\hat{C}_{xi} \leq 2C_x - \frac{p_{xi}}{\mu_{xi}} + \frac{p_{xit^*}}{\bar{v}_i} - \frac{p_{xit^*}}{\mu_i} \leq C_x(2 + [R_i(1 - 2/m_i)]^+) \tag{18}$$

When one or more subjobs are released after time zero, Lemma 7 implies that it is sufficient to bound $\max\limits_{1 \leq k \leq j_i} \{r_{\sigma_i(k)i}\}$ by some constant multiple of C_x. Since σ_i is defined by increasing $L_{ji} \doteq C_j - p_{ji}/(2\mu_{ji})$, $L_{\sigma_i(a)i} \leq L_{\sigma_i(b)i}$ implies

$$r_{\sigma_i(a)i} + \frac{p_{\sigma_i(a)i}}{2\mu_{\sigma_i(a)i}} + \frac{p_{\sigma_i(b)i}}{2\mu_{\sigma_i(b)i}} C_{\sigma_i(a)} - \frac{p_{\sigma_i(a)i}}{2\mu_{\sigma_i(a)i}} + \frac{p_{\sigma_i(b)i}}{2\mu_{\sigma_i(b)i}} \leq C_{\sigma_i(b)} \;\forall\; a \leq b \tag{19}$$

[6] "$+p_{xit^*}$"; see associated proofs.
[7] We omit the customary $\star$ to avoid clutter in notation.

and so $\max_{1\le k\le j_i}\left\{r_{\sigma_i(l)i}\right\} + p_{xi}/(2\mu_{xi}) \le C_x$. As before, combine this with Lemma 7 and the fact that $p_{xit^*} \le p_{xi} = \sum_{t\in T_{xi}} p_{xit}$ to yield the following inequalities

$$\hat{C}_{xi} \le 3C_x - \frac{3p_{xi}}{2\mu_{xi}} + \frac{p_{xit^*}}{\bar{v}_i} - \frac{p_{xit^*}}{\mu_i} \le C_x(3 + [R_i(1 - 5/(2m_i))]^+) \tag{20}$$

which complete our proof. ◀

4.2 CC-LP for Identical Machines

▶ **Theorem 9.** *If machines are of unit speed, then CC-LP yields an objective that is ...*

	$r_{ji} \equiv 0$	*some* $r_{ji} > 0$
single-task subjobs	$\le$ *2 OPT*	$\le$ *3 OPT*
multi-task subjobs	$\le$ *3 OPT*	$\le$ *4 OPT*

Proof. Define $[\cdot]^+$, x, C_x, $\hat{C}_x$, i, σ_i, and t^* as in Theorem 8. When $r_{ji} \equiv 0$, one need only give a more careful treatment of the first inequality in (18) (using $\mu_{ji} = q_{ji}$).

$$\hat{C}_{x,i} \le 2C_x + p_{xit^*} - p_{xit^*}/m_i - p_{xi}/q_{xi} \le C_x(2 + [1 - 1/m_i - 1/q_{xi}]^+) \tag{21}$$

Similarly, when some $r_{ji} > 0$, the first inequality in (20) implies the following.

$$\hat{C}_{x,i} \le 3C_x + p_{xit^*} - p_{xit^*}/m_i - 3p_{xi}/(2q_{xi}) \le C_x(3 + [1 - 1/m_i - 3/(2q_{xi})]^+) \tag{22}$$

◀

The key in the refined analysis of Theorem 9 lay in how $-p_{xi}/q_{xi}$ is used to annihilate $+p_{xit^*}$. While $q_{xi} = 1$ (i.e. single-task subjobs) is sufficient to accomplish this, it is not strictly *necessary*. The theorem below shows that we can annihilate the $+p_{xit^*}$ term whenever all tasks of a given subjob are of the same length. Note that the tasks need not be *unit*, as the lengths of tasks across different subjobs can differ.

▶ **Theorem 10.** *Suppose $v_{\ell i} \equiv 1$. If p_{jit} is constant over $t \in T_{ji}$ for all $j \in N$ and $i \in M$, then algorithm CC-LP is a 2-approximation when $r_{ji} \equiv 0$, and a 3-approximation otherwise.*

Proof. The definition of p_{xi} gives $p_{xi}/q_{xi} = \sum_{t\in T_{xi}} p_{xit}/q_{xi}$. Using the assumption that p_{jit} is constant over $t \in T_{ji}$, we see that $p_{xi}/q_{xi} = (q_{xi} + |T_{xi}| - q_{xi})p_{xit^*}/q_{xi}$, where $|T_{xi}| \ge q_{xi}$. Apply this to Inequality (21) from the proof of Theorem 9; some algebra yields

$$\hat{C}_{xi} \le 2C_x - p_{xit^*}/m_i - p_{xit^*}\left(|T_{xi}| - q_{xi}\right)/q_{xi} \le 2C_x. \tag{23}$$

The case with some $r_{ji} > 0$ uses the same identity for p_{xi}/q_{xi}. ◀

Bansal and Khot [2] showed that is is UGC-Hard to approximate $CC|m_i \equiv 1|\sum w_j C_j$ with a constant factor less than 2. Theorem 10 is significant because it shows that CC-LP can attain the same guarantee for *arbitrary* m_i, provided $v_{\ell i} \equiv 1$ and p_{jit} is constant over t.

5 Combinatorial Algorithms

In this section, we introduce an extremely fast combinatorial algorithm with performance guarantees similar to CC-LP for "unstructured" inputs (i.e. those for which some $v_{\ell i} > 1$, or some T_{ji} have p_{jit} non-constant over t). We call this algorithm *CC-TSPT*. CC-TSPT uses the MUSSQ algorithm for concurrent open shop (from [10]) as a subroutine. As SWAG (from [8]) motivated development of CC-TSPT, we first address SWAG's worst-case performance.

5.1 A Degenerate Case for SWAG

The full version of this paper [12] contains a detailed explanation of SWAG's mechanics beside pseudocode as a necessary component of making claims on worst-case performance. In this version of the paper, we omit this reference material for space considerations.

▶ **Theorem 11.** *For an instance I of $PD||\sum C_j$, let $SWAG(I)$ denote the objective function value of SWAG applied to I, and let $OPT(I)$ denote the objective function value of an optimal solution to I. Then for all $L \geq 1$, there exists an $I \in \Omega_{PD||\sum C_j}$ such that $SWAG(I)/OPT(I) > L$.*

Proof. Let $L \in \mathbb{N}^+$ be fixed but otherwise arbitrary. Construct an instance I_L^m as follows:

$N = N_1 \cup N_2$ where N_1 is a set of m jobs, and N_2 is a set of L jobs. Job $j \in N_1$ has processing time p on cluster j and zero all other clusters. Job $j \in N_2$ has processing time $p(1-\epsilon)$ on all m clusters. ϵ is chosen so that $\epsilon < 1/L$.

It is easy to verify that SWAG will generate a schedule where all jobs in N_2 precede all jobs in N_1 (due to the savings of $p\epsilon$ for jobs in N_2). We propose an *alternative* solution in which all jobs in N_1 precede all jobs in N_2. Denote the objective value for this alternative solution $ALT(I_L^m)$, noting $ALT(I_L^m) \geq OPT(I_L^m)$.

By symmetry, and the fact that all clusters have a single machine, we can see that $SWAG(I_L^m)$ and $ALT(I_L^m)$ are given by the following

$$SWAG(I_L^m) = p(1-\epsilon)L(L+1)/2 + p(1-\epsilon)Lm + pm \tag{24}$$

$$ALT(I_L^m) = p(1-\epsilon)L(L+1)/2 + pL + pm \tag{25}$$

Since L is fixed, we can take the limit with respect to m.

$$\lim_{m\to\infty} \frac{SWAG(I_L^m)}{ALT(I_L^m)} = \lim_{m\to\infty} \frac{p(1-\epsilon)Lm + pm}{pm} = L(1-\epsilon) + 1 > L \tag{26}$$

The above implies the existence of a sufficiently large number of clusters $\overline{m}$, such that $m \geq \overline{m}$ implies $SWAG(I_L^m)/OPT(I_L^m) > L$. This completes our proof. ◀

Theorem 11 demonstrates that that although SWAG performed well in simulations, it may not be reliable. The rest of this section introduces an algorithm not only with superior runtime to SWAG (generating a permutation of jobs in $O(n^2 + nm)$ time, rather than $O(n^2 m)$ time), but also a constant-factor performance guarantee.

5.2 CC-TSPT : A Fast 2 + R Approximation

Our combinatorial algorithm for concurrent cluster scheduling exploits an elegant transformation to concurrent open shop. Once we consider this simpler problem, it can be handled with MUSSQ [10] and List-LPT. Our contributions are twofold: (1) we prove that this intuitive technique yields an approximation algorithm for a decidedly more general problem, and (2) we show that a *non-intuitive* modification can be made that maintains theoretical bounds while improving empirical performance. We begin by defining our transformation.

▶ **Definition 12** (The Total Scaled Processing Time (TSPT) Transformation)**.** Let Ω_{CC} be the set of all instances of $CC||\sum w_j C_j$, and let Ω_{PD} be the set of all instances of $PD||\sum w_j C_j$. Note that $\Omega_{PD} \subset \Omega_{CC}$. Then the Total Scaled Processing Time Transformation is a mapping

$$TSPT : \Omega_{CC} \to \Omega_{PD} \quad \text{with} \quad (T, v, w) \mapsto (X, w) \; : \; x_{ji} = \textstyle\sum_{t\in T_{ji}} p_{jit}/\mu_i$$

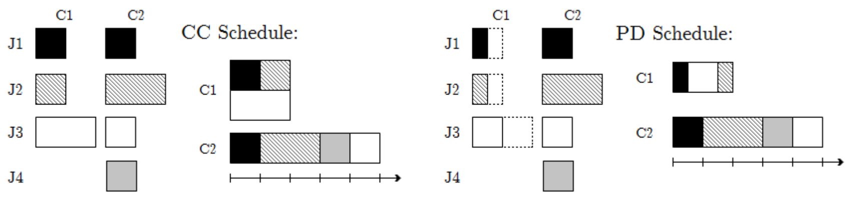


Figure 3 An instance I of $CC||\sum w_j C_j$, and its image $I' = TSPT(I)$. The schedules were constructed with List-LPT using the same permutation for I and I'.

i.e., x_{ji} is the total processing time required by subjob (j, i), scaled by the sum of machine speeds at cluster i. Throughout this section, we will use $I = (T, v, w)$ to denote an arbitrary instance of $CC||\sum w_j C_j$, and $I' = (X, w)$ as the image of I under TSPT. Figure 3 shows the result of TSPT applied to our baseline example.

We take the time to emphasize the simplicity of our reduction. Indeed, the TSPT transformation is perhaps the first thing one would think of given knowledge of the concurrent open shop problem. What is surprising is how one can attain constant-factor performance guarantees even after such a simple transformation.

Algorithm CC-TSPT: Execute MUSSQ on $I' = TSPT(I)$ to generate a permutation of jobs σ. List schedule instance I by σ on each cluster according to List-LPT.

Towards proving the approximation ratio for CC-TSPT, we will establish a critical inequality in Lemma 13. The intuition behind Lemma 13 requires thinking of every job j in I as having a corresponding representation in j' in I'. Job j in I will be scheduled in the CC environment, while job j' in I' will be scheduled in the PD environment. We consider what results when the same permutation σ is used for scheduling in both environments.

Now the definitions for the lemma: let $C^{CC}_{\sigma(j)}$ be the completion time of job $\sigma(j)$ resulting from List-LPT on an arbitrary permutation σ. Define $C^{CC\star}_{\sigma(j)}$ as the completion time of job $\sigma(j)$ in the CC environment in the optimal solution. Lastly, define $C^{PD,I'}_{\sigma(j')}$ as the completion time of job $\sigma(j')$ in I' when scheduling by List-LPT(σ) in the PD environment.

▶ **Lemma 13.** *For $I' = TSPT(I)$, let j' be the job in I' corresponding to job j in I. For an arbitrary permutation of jobs σ, we have $C^{CC}_{\sigma(j)} \leq C^{PD,I'}_{\sigma(j')} + R \cdot C^{CC\star}_{\sigma(j)}$.*

Proof. After list scheduling has been carried out in the CC environment, we may determine $C^{CC}_{\sigma(j)i}$ - the completion time of subjob $(\sigma(j), i)$. We can bound $C^{CC}_{\sigma(j)i}$ using Lemma 7 (which implies (27)), and the serial-processing nature of the PD environment (which implies (28)).

$$C^{CC}_{\sigma(j)i} \leq p_{\sigma(j)i1}\left(1/\bar{v} - 1/\mu_i\right) + \textstyle\sum_{\ell=1}^{j} p_{\sigma(\ell)i}/\mu_i \tag{27}$$

$$\textstyle\sum_{\ell=1}^{j} p_{\sigma(\ell)i}/\mu_i \leq C^{PD,I'}_{\sigma(j')} \quad \forall\ i \in M \tag{28}$$

If we relax the bound given in Inequality (27) and combine it with Inequality (28), we see that $C^{CC}_{\sigma(j)i} \leq C^{PD,I'}_{\sigma(j')} + p_{\sigma(j)i1}/\bar{v}$. The last step is to replace the final term with something more meaningful. Using $p_{\sigma(j)1}/\bar{v} \leq R \cdot C^{CC\star}_{\sigma(j)}$ (which is immediate from the definition of R) the desired result follows. ◀

While Lemma 13 is true for arbitrary σ, now we consider $\sigma = MUSSQ(X, w)$. The proof of MUSSQ's correctness established the first inequality in the chain of inequalities below. The second inequality can be seen by substituting p_{ji}/μ_i for x_{ji} in LP0(I') (this shows that the constraints in LP0(I') are weaker than those in LP1(I)). The third inequality follows from the Validity Lemma.

$$\textstyle\sum_{j\in N} w_{\sigma(j)} C^{PD,I'}_{\sigma(j)} \leq 2\sum_{j\in N} w_j C^{\text{LP0}(I')}_j \leq 2\sum_{j\in N} w_j C^{\text{LP1}(I)}_j \leq 2OPT(I) \tag{29}$$

Combining Inequality (29) with Lemma 13 allows us to bound the objective in a way that does not make reference to I'.

$$\textstyle\sum_{j\in N} w_{\sigma(j)} C^{CC}_{\sigma(j)} \leq \sum_{j\in N} w_{\sigma(j)} \left[C^{PD,I'}_{\sigma(j)} + R\cdot C^{CC\star}_{\sigma(j)}\right] \leq \ 2\cdot OPT(I) + R\cdot OPT(I) \tag{30}$$

Inequality (30) completes our proof of the following theorem.

▶ **Theorem 14.** *Algorithm CC-TSPT is a $2+R$ approximation for $CC||\sum w_jC_j$.*

5.3 CC-TSPT with Unit Tasks and Identical Machines

Consider concurrent cluster scheduling with $v_{\ell i} = p_{jit} = 1$ (i.e., all processing times are unit, although the size of the collections T_{ji} are unrestricted). In keeping with the work of Zhang, Wu, and Li [17] (who studied this problem in the single-cluster case), we call instances with these parameters "fully parallelizable," and write $\beta = fps$ for Graham's $\alpha|\beta|\gamma$ taxonomy.

Zhang et al. showed that scheduling jobs greedily by "Largest Ratio First" (decreasing w_j/p_j) results in a 2-approximation, where 2 is a tight bound. This comes as something of a surprise since the Largest Ratio First policy is *optimal* for $1||\sum w_jC_j$ - which their problem very closely resembles. We now formalize the extent to which $P|fps|\sum w_jC_j$ resembles $1||\sum w_jC_j$: define the *time resolution* of an instance I of $CC|fps|\sum w_jC_j$ as $\rho_I = \min_{j\in N, i\in M}\lceil p_{ji}/m_i\rceil$. Indeed, one can show that as the time resolution increases, the performance guarantee for LRF on $P|fps|\sum w_jC_j$ approaches that of LRF on $1||\sum w_jC_j$. We prove the analogous result for our problem.

▶ **Theorem 15.** *CC-TSPT for $CC|fps|\sum w_jC_j$ is a $(2+1/\rho_I)-$approximation.*

Proof. Applying techniques from the proof of Lemma 13 under the hypothesis of this theorem, we have $C^{CC}_{\sigma(j),i} \leq C^{PD,I'}_{\sigma(j)} + 1$. Next, use the fact that for all $j \in N$, $C^{CC,OPT}_{\sigma(j)} \geq \rho_I$ by the definition of ρ_I. These facts together imply $C^{CC}_{\sigma(j),i} \leq C^{PD,I'}_{\sigma(j)} + C^{CC,OPT}/\rho_I$. Thus

$$\textstyle\sum_{j\in N} w_j C^{CC}_{\sigma(j)} \leq \sum_{j\in N} w_j \left[C^{PD,I'}_{\sigma(j)} + C^{CC,OPT}/\rho_I\right] \leq 2\cdot OPT + OPT/\rho_I. \tag{31}$$

◀

5.4 CC-ATSPT : Augmenting the LP Relaxation

The proof of Theorem 14 appeals to a trivial lower bound on $C^{CC\star}_{\sigma(j)}$, namely $p_{\sigma(j)1}/\bar{v} \leq R\cdot C^{CC\star}_{\sigma(j)}$. We attain constant-factor performance guarantees in spite of this, but it is natural to wonder how the *need* for such a bound might come hand-in-hand with empirical weaknesses. Indeed, TSPT can make subjobs consisting of many small tasks look the same as subjobs consisting of a single very long task. Additionally, a cluster hosting a subjob with a single extremely long task might be identified as a bottleneck by MUSSQ, even if that cluster has more machines than it does tasks to process.

We would like to mitigate these issues by introducing the simple lower bounds on C_j as seen in constraints ($1B$) and ($1C$). This is complicated by the fact that MUSSQ's proof of correctness only allows constraints of the form in $(1A)$. For $I \in \Omega_{PD}$ this is without loss of generality, since $|S| = 1$ in LP0 implies $C_j \geq p_{ji}$, but since we apply LP0 to $I' = TSPT(I)$, $C_j \geq x_{ji}$ is equivalent to $C_j \geq p_{ji}/\mu_i$ (a much weaker bound than we desire).

Nevertheless, we can bypass this issue by introducing additional clusters and appropriately defined subjobs. We formalize this with the "Augmented Total Scaled Processing Time" (ATSPT) transformation. Conceptually, ATSPT creates n "imaginary clusters", where each imaginary cluster has nonzero processing time for exactly one job.

▶ **Definition 16** (The Augmented TSPT Transformation)**.** Let Ω_{CC} and Ω_{PD} be as in the definition for TSPT. Then the Augmented TSPT Transformation is likewise a mapping

$$ATSPT : \Omega_{CC} \to \Omega_{PD} \quad \text{with} \quad (T, v, w) \mapsto (X, w) \ : \ X = \left[\ X_{TSPT(I)} \ \middle| \ D \ \right].$$

Where $D \in \mathbb{R}^{n \times n}$ is a diagonal matrix with d_{jj} as any valid lower bound on the completion time of job j (such as the right hand sides of constraints $(1B)$ and $(1C)$ of LP1).

Given that d_{jj} is a valid lower bound on the completion time of job j, it is easy to verify that for $I' = ATSPT(I)$, LP1(I') is a valid relaxation of I. Because MUSSQ returns a permutation of jobs for use in list scheduling by List-LPT, these "imaginary clusters" needn't be accounted for beyond the computations in MUSSQ.

6 A Reduction for Minimizing Total Weighted Lateness on Identical Parallel Machines

The problem of minimizing total weighted lateness on a bank of identical parallel machines is typically denoted $P||\sum w_j L_j$, where the lateness of a job with deadline d_j is $L_j \doteq \max\{C_j - d_j, 0\}$. The reduction we offer below shows that $P||\sum w_j L_j$ can be stated in terms of $CC||\sum w_j C_j$ *at optimality*. Thus while a Δ approximation to $CC||\sum w_j C_j$ does not imply a Δ approximation to $P||\sum w_j L_j$, the reduction below nevertheless provides new insights on the structure of $P||\sum w_j L_j$.

▶ **Definition 17** (Total Weighted Lateness Reduction)**.** Let $I = (p, d, w, m)$ denote an instance of $P||\sum w_j L_j$. p is the set of processing times, d is the set of deadlines, w is the set of weights, and m is the number of identical parallel machines. Given these inputs, we transform $I \in \Omega_{P||\sum w_j L_j}$ to $I' \in \Omega_{CC}$ in the following way.

Create a total of $n + 1$ clusters. Cluster 0 has m machines. Job j has processing time p_j on this cluster, and $|T_{j0}| = 1$. Clusters 1 through n each consist of a single machine. Job j has processing time d_j on cluster j, and zero on all clusters other than cluster 0 and cluster j. Denote this problem I'.

We refer the reader to Figure 2 for an example output of this reduction.

▶ **Theorem 18.** *Let I be an instance of $P||\sum w_j L_j$. Let I' be an instance of $CC||\sum w_j C_j$ resulting from the transformation described above. Any list schedule σ that is optimal for I' is also optimal for I.*

The proof can be found in the full version of this paper.

7 Closing Remarks

We now take a moment to address a subtle issue in the concurrent cluster problem: what price do we pay for using the same permutation on all clusters (i.e. single-σ schedules)?

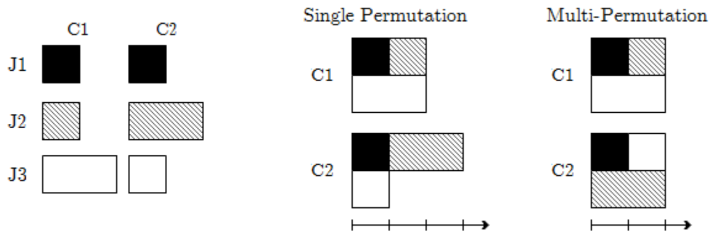


Figure 4 An instance of $CC||\sum C_j$ (i.e. $w_j \equiv 1$) for which there does not exist a single-σ schedule which attains the optimal objective value. In the single-σ case, one of the jobs necessarily becomes delayed by one time unit compared to the multi-σ case. As a result, we see a 20% optimality gap even when $v_{\ell i} \equiv 1$.

For concurrent open shop, it has been shown ([16, 10]) that single-σ schedules may be assumed without loss of optimality. As is shown in Figure 4, this does *not* hold for concurrent cluster scheduling in the general case. In fact, that is precisely why the strong performance guarantees for algorithm CC-LP rely on clusters having possibly unique permutations.

Our more novel contributions came in our analysis for CC-TSPT and CC-ATSPT. First, we could not rely on the processing time of the last task for a job to be bounded above by the job's completion time variable C_j in LP0(I'), and so we appealed to a lower bound on C_j that was not stated in the LP itself. The need to incorporate this second bound is critical in realizing the strength of algorithm CC-TSPT, and uncommon in LP rounding schemes. Second, CC-ATSPT is novel in that it introduces constraints that would be redundant for LP0(I) when $I \in \Omega_{PD}$, but become relevant when viewing $LP0(I')$ as a relaxation for $I \in \Omega_{CC}$. This approach has potential for more broad applications since it represented effective use of a limited constraint set supported by a known primal-dual algorithm.

We now take a moment to state some open problems in this area. One topic of ongoing research is developing a factor 2 purely combinatorial algorithm for the special case of concurrent cluster scheduling considered in Theorem 10. In addition, it would be of broad interest to determine the worst-case loss to optimality incurred by assuming single-permutation schedules for $CC|v \equiv 1|\sum w_j C_j$. The simple example above shows that an optimal single-σ schedule can have objective 1.2 times the globally optimal objective. Meanwhile, Theorem 14 shows that there always exists a single-σ schedule with objective no more than 3 times the globally optimal objective. Thus, we know that the worst-case performance ratio is in the interval $[1.2, 3]$, but we do not know its precise value. As a matter outside of scheduling theory, it would be valuable to survey primal-dual algorithms with roots in LP relaxations to determine which have constraint sets that are amenable to implicit modification, as in the fashion of CC-ATSPT.

Acknowledgments. Special thanks to Andreas Schulz for sharing some of his recent work with us [15]. His thorough analysis of a linear program for $P||\sum w_j C_j$ drives the LP-based results in this paper. Thanks also to Chien-Chung Hung and Leana Golubchik for sharing [8] while it was under review, and to Ioana Bercea and Manish Purohit for their insights on SWAG's performance. Lastly, our sincere thanks to William Gasarch for organizing the REU which led to this work, and to the 2015 CAAR-REU cohort for making the experience an unforgettable one; in the words of Rick Sanchez *wubalubadubdub!*

References

1 Inc Amazon Web Services. *AWS Lambda - Serverless Compute*, 2016 (accessed April 3, 2016). URL: `https://aws.amazon.com/lambda/`.

2 Nikhil Bansal and Subhash Khot. Inapproximability of Hypergraph Vertex Cover and Applications to Scheduling Problems. *Automata, Languages and Programming*, 6198:250–261, 2010. `doi:10.1007/978-3-642-14165-2`.

3 Zhi-Long Chen and Nicholas G. Hall. Supply chain scheduling: Assembly systems. Working paper., 2000. `doi:10.1007/978-3-8349-8667-2`.

4 Naveen Garg, Amit Kumar, and Vinayaka Pandit. Order Scheduling Models: Hardness and Algorithms. *FSTTCS 2007: Foundations of Software Technology and Theoretical Computer Science*, 4855:96–107, 2007. `doi:10.1007/978-3-540-77050-3\_8`.

5 Teofilo Gonzalez, Oscar Ibarra, and Sartaj Sahni. Bounds for LPT Schedules on Uniform Processors. *SIAM Journal on Computing*, 6(1):155–166, 1977.

6 Ronald L Graham, Eugene L Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, 5:287–326, 1979.

7 Mohammad Hajjat, Shankaranarayanan P N, David Maltz, Sanjay Rao, and Kunwadee Sripanidkulchai. Dealer : Application-aware Request Splitting for Interactive Cloud Applications. *CoNEXT 2012*, pages 157–168, 2012.

8 Chien-Chun Hung, Leana Golubchik, and Minlan Yu. Scheduling jobs across geo-distributed datacenters. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*, pages 111–124. ACM, 2015.

9 J. Y T Leung, Haibing Li, and Michael Pinedo. Scheduling orders for multiple product types to minimize total weighted completion time. *Discrete Applied Mathematics*, 155(8):945–970, 2007. `doi:10.1016/j.dam.2006.09.012`.

10 Monaldo Mastrolilli, Maurice Queyranne, Andreas S. Schulz, Ola Svensson, and Nelson A. Uhan. Minimizing the sum of weighted completion times in a concurrent open shop. *Operations Research Letters*, 38(5):390–395, 2010. `doi:10.1016/j.orl.2010.04.011`.

11 Microsoft. *Azure Service Fabric*, 2016 (accessed April 3, 2016). URL: `https://azure.microsoft.com/en-us/services/service-fabric/`.

12 Riley Murray, Megan Chao, and Samir Khuller. Scheduling distributed clusters of parallel machines [full version], 2016. Unpublished. URL: `http://www.cs.umd.edu/users/samir/grant/ESA2016Full.pdf`.

13 Maurice Queyranne. Structure of a simple scheduling polyhedron. *Mathematical Programming*, 58(1-3):263–285, 1993. `doi:10.1007/BF01581271`.

14 Andreas S. Schulz. Polytopes and scheduling. *PhD Thesis*, 1996.

15 Andreas S Schulz. From linear programming relaxations to approximation algorithms for scheduling problems : A tour d'horizon. Working paper; available upon request., 2012.

16 C. Sriskandarajah and E. Wagneur. Openshops with jobs overlap. *European Journal of Operations Research*, 71:366–378, 1993.

17 Qiang Zhang, Weiwei Wu, and Minming Li. Resource Scheduling with Supply Constraint and Linear Cost. *COCOA 2012 Conference*, 2012. `arXiv:9780201398298`, `doi:10.1007/3-540-68339-9\_34`.

Finding Large Set Covers Faster via the Representation Method

Jesper Nederlof[*]

Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands
j.nederlof@tue.nl

Abstract

The worst-case fastest known algorithm for the Set Cover problem on universes with n elements still essentially is the simple $O^*(2^n)$-time dynamic programming algorithm, and no non-trivial consequences of an $O^*(1.01^n)$-time algorithm are known. Motivated by this chasm, we study the following natural question: Which instances of Set Cover *can* we solve faster than the simple dynamic programming algorithm? Specifically, we give a Monte Carlo algorithm that determines the existence of a set cover of size σn in $O^*(2^{(1-\Omega(\sigma^4))n})$ time. Our approach is also applicable to Set Cover instances with exponentially many sets: By reducing the task of finding the chromatic number $\chi(G)$ of a given n-vertex graph G to Set Cover in the natural way, we show there is an $O^*(2^{(1-\Omega(\sigma^4))n})$-time randomized algorithm that given integer $s = \sigma n$, outputs NO if $\chi(G) > s$ and YES with constant probability if $\chi(G) \leq s - 1$.

On a high level, our results are inspired by the 'representation method' of Howgrave-Graham and Joux [EUROCRYPT'10] and obtained by only evaluating a randomly sampled subset of the table entries of a dynamic programming algorithm.

1998 ACM Subject Classification G.2.2 [Graph Algorithms] Hypergraphs

Keywords and phrases Set Cover, Exact Exponential Algorithms, Fine-Grained Complexity

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.69

1 Introduction

The Set Cover problem is, after determining satisfiability of CNF formulas or Boolean circuits, one of the canonical NP-complete problems. It not only directly models many applications in practical settings, but also algorithms for it routinely are used as tools for theoretical algorithmic results (e.g., [17]). It is a problem 'whose study has led to the development of fundamental techniques for the entire field' of approximation algorithms.[1] However, the exact exponential time complexity of Set Cover is still somewhat mysterious: We know algorithms need to use super-polynomial time assuming $P \neq NP$ and (denoting n for the universe size) $O^*(2^{\Omega(n)})$ time assuming the Exponential Time Hypothesis, but how large the exponential should be is not clear. In particular, no non-trivial consequences of an $O^*(1.01^n)$-time algorithm are currently known.

Even though it is one of the canonical NP-complete problems, the amount of studies of exact algorithms for Set Cover pales in comparison with the amount of literature on exact algorithms for CNF-Sat: Many works focus on finding $O^*(c^n)$-time algorithms for $c < 2$

[*] Funded by the NWO VENI project 639.021.438. This work was partly done while the author was visiting the Simons Institute for the Theory of Computing during the program 'Fine-Grained Complexity and Algorithm Design' in the fall of 2015.

[1] As the Wikipedia page on Set Cover quotes the textbook by Vazirani [32, p15].

for CNF-Sat on n-variable CNF-formulas in special cases such as, among others, bounded clause width [31, 17, 12], bounded clause density [11, 25] or few projections [27, 29, 30]. Improved exponential time algorithms for special cases of problems other than CNF-Sat were also studied for e.g. Graph Coloring or Traveling Salesman on graphs bounded degree/average degree [8, 9, 15, 20].

In this paper we are interested in the exponential time complexity of Set Cover, and study which properties are sufficient to have improved exponential time algorithms. Our interest in finding faster exponential time algorithms for Set Cover does not only stem from it being a canonical NP-complete problem, but also from its unclear relation with CNF-Sat. Intriguingly, on one hand Set Cover has some similarities with the CNF-Sat: 1. Both problems take an (annotated) hypergraph as input. 2. The improvability of the worst-case complexity of CNF-Sat is essentially equivalent to the improvability of the worst-case complexity of Hitting Set [14], which is just a reparametrization[2] of Set Cover. But, on the other hand the problems are quite different to our understanding: 1. Most algorithms for Set Cover use dynamic programming or some variant of inclusion exclusion, while most algorithms for CNF-Sat are based on branching. 2. No connection between the exponential time complexities of both problems is known (see [14]). One hope would be that a better understanding of the exact complexity of Set Cover might shed more light on this unclarity. Moreover, Cygan et al. [14] also show that if we would like to improve the run time $O^*(f(k))$ of several parameterized algorithms to $O^*(f(k)^{1-\Omega(1)})$, we first need to find an $O^*(2^{(1-\Omega(1))n})$-time algorithm for Set Cover. These parameterized algorithms include the classic algorithm for Subset Sum, as well as more recent algorithms for Connected Vertex Cover and Steiner Tree.

Relevant previous work. The algorithmic results on Set Cover that are the most relevant to our work are as follows: The folklore dynamic programming algorithm runs in $O^*(2^n)$ time. A notable special case of Set Cover that can be solved in $O^*(2^{(1-\Omega(1))n})$ time is due to Koivisto [28]: He gives an algorithm that runs in time $O^*(2^{(1-\frac{1}{O(r)})n})$-time algorithm if all sets are at most of size r. Björklund et al. [10] show that the problem can be solved in $2^n \operatorname{poly}(n)$ time (which is faster if the number of sets is exponentially large in n). Björklund et al. [7] give a randomized algorithm that assumes all sets are of size q and determines whether there exist p pairwise disjoint sets in $O^*(2^{(1-\epsilon)pq})$ time where $\epsilon > 0$ depends on q.

Our Main Results. We investigate what are sufficient structural properties of instances of Set Cover, and the closely related Set Partition (in which the picked sets need to be disjoint), problems to be solvable in time significantly faster than the currently known algorithms. We will outline our main results now:

▶ **Theorem 1.1.** *There is a Monte Carlo algorithm that takes an instance of* Set Cover *on n elements and m sets and an integer s as input and determines whether there exists a set cover of size s in $O(2^{(1-\Omega(\sigma^4))n}m)$ time, where $\sigma = s/n$.*

We remark that this generalizes the result of Koivisto [28] in the sense that it solves a larger class of instances in $O^*(2^{(1-\Omega(1))n})$ time: If all set sizes are bounded by a constant r, a set partition needs to consist of at least n/r sets and Theorem 1.1 applies with $\sigma = 1/r$

[2] One way of stating Hitting Set in this context, is that we have an instance of the Set Cover problem but aim to find an $O^*(2^{(1-\Omega(1))m})$ time algorithm, where m denotes the number of sets.

(although this gives a slower algorithm than Koivisto's in this special case). Moreover, it seems hard to extend the approach of Koivisto to our more general setting.

The second result demonstrates that our techniques are also applicable to Set Cover instances with exponentially many sets, a canonical example of which being graph coloring:

▶ **Theorem 1.2.** *There is a randomized algorithm that given graph G and integer $s = \sigma n$, in $O^*(2^{(1-\Omega(\sigma^4))n})$ time outputs* **yes** *with constant probability, if $\chi(G) < s$, and* **no**, *if $\chi(G) > s$.*

Representation method for Set Cover. We feel the main technique used in this paper is equally interesting as the result, and will therefore elaborate on its origin here. Our technique is on a high level inspired by the following simple observation ingeniously used by Howgrave-Graham and Joux [23]: Suppose $\mathcal{S} \subseteq 2^{[m]}$ is a set of solutions implicitly given and we seek for a solution $X \in \mathcal{S}$ with $|X| = s$ by listing all sets of $\binom{[m]}{s/2}$ and performing pairwise checks to see which two combine to an element of $\mathcal{S}$. Then we can restrict our search in various ways since there will be as many as $\binom{s}{s/2}$ pairs guiding us to X. In [23] and all subsequent works (including [3, 4, 1, 2]), this idea was used to speed up 'meet-in-the-middle attacks' (also called 'birthday attacks' [26, Chapter 6]). We will refer to uses of this idea as the 'representation method' since it crucially relies on the fact that X has many representations as pairs. To indicate the power of this technique in the context of Set Cover and Set Partition we show that without changes it already gives an $O^*(2^{0.3399m})$-time Monte Carlo algorithm for the Set Partition problem with m sets, and even for a more general linear satisfiability problem on m variables. For the latter problem this improves the $O^*(2^{m/2})$ time algorithm based on the meet-in-the-middle attack that was the fastest known before.

At first sight the representation method seemed to be inherently only useful for improving algorithms based on the meet-in-the-middle attack. However, the main conceptual contribution of this work is to show that it is also useful in other settings, or at least for improving the dynamic programming algorithm for the Set Cover and Set Partition problems if the solution size is large. On a high level, we show this as follows in the case of Set Partition:[3] for a subset W of the elements of the Set Partition instance, define $T[W]$ to be the minimum number of disjoint sets needed to cover all elements of W. Stated slightly oversimplified, we argue that if a minimal set partition of size s is large, we have that $T[W] + T[[n] \setminus W] = s$ for $\binom{s}{s/2}$ sets W with $|W|$ close to $n/2$. To relate this to later sections, let us remark we refer to such a set W as a *witness halve*. Subsequently, we exploit the presence of many witness halves by using a dynamic programming algorithm that samples a set of the subsets with size close to $n/2$ and only evaluates table entries from this sample plus the table entries required to compute the table entries from the sample.

Organization. This paper is organized as follows: In Section 2, we recall preliminaries and introduce notation. In Section 3, we discuss new observations and basic results that we feel are useful for developing a better understanding of the complexity of Set Cover with respect to several structural properties of instances. In Section 4 we formally present the notion of witness halves and prepare tools for exploiting the existence of many witness halves. In Section 5 we prove our main results and in Section 6 we suggest further research.

[3] The algorithm for Set Cover actually reduces to Set Partition.

2 Preliminaries and Notation

For a Boolean predicate p, we let $[p]$ denote 1 if p is true and 0 otherwise. On the other hand, if p is an integer we let $[p]$ denote $\{1,\dots,p\}$. As usual, $\mathbb{N}$ denotes all positive integers. Running times of algorithms are often stated using $O^*(\cdot)$ notation which suppresses factors polynomial in the input size. To avoid superscript, we sometimes use $\exp(x)$ to denote e^x. We denote lg for the base-2 logarithm. If $G=(V,E)$ and $v \in V$ we denote $N(v)=\{w \in V : (v,w) \in E\}$ and for $X \subseteq V$ we extended this notation to $N(X)=\bigcup_{v \in X} N(v)$. For reals $a,b>0$ we let $a \pm b$ denote the interval $[a-b,a+b]$. A false positive (negative) of an algorithm is an instance on which it incorrectly outputs YES (respectively, NO). In this work we call an algorithm Monte Carlo if it has no false positives and if any instance is a false negative with probability at most 1/4. We denote vectors with boldface for clarity. For a real number $x \in [0,1]$, $h(x)=-x\lg x-(1-x)\lg(1-x)$ denotes the binary entropy of x, where $0\lg 0$ should be thought of as 0. It is well known that $\binom{b}{a} \le 2^{h(a/b)b}$ (and this can for example be proved using Stirling's approximation). It is easy to see from the definition that $h(\cdot)$ is symmetric in the sense that $h(x)=h(1-x)$.

▶ **Lemma 2.1.** *The following can be verified using standard calculus:*
1. $h(1/2-x)=h(1/2+x) \le 1-x^2$ *for all* $x \in (0,1/2)$,
2. $h(x) \le x\lg(4/x)$ *for all* $x \in (0,1)$,
3. $(1-1/n)^n \le 1/e$.

▶ **Lemma 2.2** (Hoeffding bound [21]). *If* $X_1,\dots,X_s$ *are independent,* $Y=\sum_{i=1}^s X_i$ *and* $a_i \le X_i \le b_i$ *for* $i=1,\dots,s$ *then* $\Pr[|Y-\mathbb{E}[Y]| \ge t] \le 2\cdot\exp\left(\frac{-2t^2}{\sum_{i=1}^s (b_i-a_i)^2}\right)$.

Set Cover / Set Partition. In the Set Cover problem we are given a bipartite graph $G=(F\dot{\cup}U,E)$ (where F and U shorthand 'Family' and 'Universe' respectively), together with an integer s and the task is to determine whether there exists a *solution* $S \subseteq F$ such that $N(S)=U$ and $|S| \le s$. In the Set Partition problem we are given the same input as in the Set Cover problem, but we are set to determine whether there exists $S \subseteq F$ with $N(S)=U$, $|S|=s$ and additionally $N(f)\cap N(f')=\emptyset$ for every $f,f' \in S$ with $f \ne f'$. We will refer to solutions of both problems as set covers and set partitions.

Throughout this paper, we let n,m respectively denote $|U|$ and $|F|$, and refer to instances of Set Cover or Set Partition as (n,m,s)-instances to quantify their parameters. Since this work concerns Set Cover or Set Partition with large solutions we record the following basic observation that follows by constructing for each[4] c-tuple $t=(f_1,\dots,f_c) \in F^c$ of sets in the original instance a set f^t with $N(f^t)=\bigcup_{i=1}^t f_i$ in the output instance:

▶ **Observation 2.3** ([14]). *There is a polynomial time algorithm that takes a constant* $c \ge 1$ *dividing* s*, and a* (n,m,s)*-instance of* Set Cover *(resp.* Set Partition*) as input and outputs an equivalent* $(n,m^c,s/c)$*-instance of* Set Cover *(resp.* Set Partition*).*

Often it will be useful dispense with linear sized sets. To this end, the following can be achieved by simply iterating over all $f \in F$ with $|N(f)| \ge \epsilon n$ and checking for each such set whether there is a solution containing it using the $2^n\operatorname{poly}(n)$ algorithm for Set Cover [10].

[4] For Set Partition only do this for c-tuples $(f_1,\dots,f_c)$ with $N(f_i)$ disjoint.

▶ **Observation 2.4.** *There is an algorithm that, given a real number $\epsilon > 0$, takes an (n, m, s)-instance of* Set Cover *as input and outputs an equivalent (n, m', s)-instance with $m' \le m$ satisfying $|N(f)| \le \epsilon n$ for every $f \in F$. The algorithm runs in $O(m2^{(1-\epsilon)n} \operatorname{poly}(n))$ time.*

As we will see in Theorem 3.4, it makes a difference in the Set Partition problem whether empty sets are allowed since we need to find a set partition of size exactly s. To exclude such sets, we will simply say that an instance is 'without empty sets'.

3 Observations and Basic Results on Set Cover and Set Partition

To improve our understanding of which properties of instances of Set Cover and Set Partition allow faster algorithms, and which techniques are useful for obtaining such faster algorithms, we will record some observations and basic results in this section. To stress that the proof techniques in this section are *not our main technical contribution*, we postpone all proofs to full version.

We prefer to state our results in terms of Set Cover because it is slightly more natural and common, but since Set Partition often is easier to deal with for our purposes we will sometimes use the following easy reduction, all of whose steps are contained in [14]:

▶ **Theorem 3.1.** *There is an algorithm that, given a real $0 < \epsilon < 1/2$, takes an (n, m, s)-instance of* Set Cover *as input and outputs an equivalent (n, m', s)-instance of* Set Partition *with $m' \le m2^{\epsilon n}$ sets in time $O(m2^{(1-\epsilon)n})$.*

For completeness, we show that in fact Set Cover and Set Partition are equivalent with respect to being solvable in time $O^*(2^{(1-\Omega(1))n})$. This was never stated in print to the best of our knowledge, but the proof uses standard ideas and is found in the full version.

▶ **Theorem 3.2.** *For some $\epsilon > 0$ there is an $O^*(2^{(1-\epsilon)n})$ time algorithm for* Set Cover *if and only if for some $\epsilon' > 0$ there is an $O^*(2^{(1-\epsilon')n})$ time algorithm for* Set Partition*.*

The following natural result is a rather direct consequence of a paper by Koivisto [28]. It reveals some more similarity with the k-CNF-Sat problem: Koivisto shows[5] that for maximum set size r, Set Cover can be solved in $O^*(2^{(1-\Omega(\frac{1}{r}))n})$ which is analogous to k-CNF-Sat being in $O^*(2^{(1-\Omega(\frac{1}{k}))n})$ time [31, 17, 12], and similarly the following result is the counterpart of $O^*(2^{(1-\Omega(\frac{1}{\delta}))n})$-time algorithms for CNF-formula's of density δ (i.e. at most δn clauses) [11, 25]. Again, this result was never explicitly stated in print to the best of our knowledge, and therefore is proved in the full version.

▶ **Theorem 3.3.** *There is an algorithm solving (n, m, s)-instances of* Set Cover *or* Set Partition *in time $m \cdot \mathsf{poly}(n)2^{n - \frac{n}{O(\lg(m/n))}}$.*

Relevant to our work is the following subtlety on solution sizes in Set Partition. It shows that for Set Partition with empty sets, finding large solutions is as hard as the general case. The proof is postponed to the full version.

▶ **Theorem 3.4.** *Suppose there exist $0 < \epsilon_1, \epsilon_2 < 1/2$ and an algorithm solving $(n, m, \epsilon_1 n)$-instances of* Set Partition *in time $O^*(2^{(1-\epsilon_2)n})$. Then there exists an $O^*(2^{(1-\epsilon_2/2)n})$-time algorithm for* Set Partition*.*

[5] Koivisto only showed this for Set Partition, but the straightforward reductions in this section carry this result over to Set Cover.

Finally, it is insightful to see how well the representation method performs on the Set Partition problem with few sets (e.g., we consider running times of the $O^*(2^m)$, where m is the number of sets). A straightforward approach of the meet-in-the-middle attack leads directly to an $O^*(2^{m/2})$ time algorithm. We show that the representation method combined with the analysis of [2, 1] in fact solves the more general Linear Sat problem. In Linear Sat, one is given an integer t, matrix $A \in \mathbb{Z}_2^{n\times m}$ and vectors $\boldsymbol{b} \in \mathbb{Z}_2^n$ and $\boldsymbol{\omega} \in \mathbb{N}^m$ the task is to find $\boldsymbol{x} \in \mathbb{Z}_2^m$ satisfying $A\boldsymbol{x} \equiv \boldsymbol{b}$ and $\boldsymbol{\omega} \cdot \boldsymbol{x} \leq t$.

▶ **Theorem 3.5.** *There is an $O^*(2^{0.3399m})$-time Monte Carlo algorithm solving* Linear Sat.

To our best knowledge no $O^*(2^{(0.5-\Omega(1))m})$-time algorithm for Linear Sat was known before. We get as a corollary that given a bipartite graph $G = (F\dot{\cup}U, E)$ we can determine the smallest size of a set partition in time $O^*(2^{0.3399m})$, which we take as a clear first signal that the representation method is useful for solving Set Partition (and Set Cover) for instances with small universe. To see this consequence, note we can reduce this problem to Linear Sat as follows: for every $f \in F$ add the incidence vector of $N(f)$ as a column to A, and set cost ω_i of picking this column to be $n|N(f)| + 1$. Then the minimum of $\boldsymbol{\omega} \cdot \boldsymbol{x}$ subject to $A\boldsymbol{x} \equiv \mathbf{1}$ will be $n^2 + s$ where s is the number of sets in a minimum set partition. Let us remark that [16, Page 130] solves (a counting version) of Set Partition in time $O^*(1.2561^m) = O^*(2^{0.329m})$, and Drori and Peleg [18] solve the problem in $O^*(2^{0.3212m})$ time,[6] so by no means our algorithm is the fastest in this setting. However, both use sophisticated branching and we find it intriguing that the representation method does work quite well even for the more general Linear Sat problem.

4 Exploiting the Presence of Many Witness β-halves

For convenience we will work with Set Partition in this section; the results straightforwardly extend to Set Cover but we will not need this in the subsequent section.

▶ **Definition 4.1.** Given an (n, m, s) instance of Set Partition, a subset $W \subseteq U$ is said to be a *witness β-halve* if $|W| \in (\frac{1}{2} \pm \beta)n$ and there exist disjoint subsets $S_1, S_2 \subseteq F$ such that $N(S_1 \cup S_2) = U$, $\sum_{f \in S_1 \cup S_2} |N(f)| = n$, $N(S_1) = W$, $N(S_2) = U \setminus W$ and $|S_1| + |S_2| = s$.

Note that this is similar to the intuitive definition outlined in Section 1, except that we require $|W| \in (\frac{1}{2} \pm \beta)|U|$ and we adjusted the definition to the Set Partition problem. Since $S_1 \cup S_2$ is a set partition of size s we see that if a witness β-halve exists, we automatically have a yes instance.

In this section we will give randomized algorithms that solve promise-variants of Set Partition with the promise that, if the instance is a yes-instance, there will be an exponential number of witness halves that are sufficiently balanced (i.e. of size close to $n/2$). In the first subsection we outline the basic algorithm and in the second subsection we show how tools from the literature can be combined with our approach to also give a faster algorithm if the number of sets is exponential in n.

[6] We attempted to find any more recent faster algorithm, but did not find this. Though, we would not be surprised if using more recent tools in branching algorithms as [19] one should be able to more significantly outperform our algorithm for Set Partition.

```
Algorithm A1($G = (F \dot\cup U, E), s, \zeta, \beta$).
Output: An estimate of whether there exists a set partition of size $s$.
1: for integer $l$ satisfying $\lfloor(1/2-\beta)n\rfloor < l < \lceil(1/2+\beta)n\rceil$ do
2:     Sample $\mathcal{W} \subseteq \binom{U}{l}$ by including every set of $\binom{U}{l}$ with probability $2^{-\zeta n}$.
3:     For every $W \in \mathcal{W}$ and $i \in [n]$, compute $c_i(W)$ and $c_i(U \setminus W)$.
4:     if $\exists i \in [n] : c_i(W) \wedge c_{s-i}(U \setminus W)$ then return yes.
5: return no.
```

Figure 1 High level description of the Algorithm implementing Theorem 4.2.

4.1 The basic algorithm

▶ **Theorem 4.2.** *There exists an algorithm* A1 *that takes an (n, m, s)-instance of* SET PARTITION *and real numbers $\beta, \zeta > 0$ satisfying $2\sqrt{\beta} \leq \zeta < 1/4$ as input, runs in time $2^{(1-(\zeta/2)^4)n}\text{poly}(n)m$, and has the following property: if there exist at least $\Omega(2^{\zeta n})$ witness β-halves it returns* **yes** *with at least constant probability, and if there does not exist a set partition of size s it returns* **no**.

Note that the theorem does not guarantee anything on Algorithm A1 if a partition of s sets exists and there are only few witness halves, but we will address this later. A high level description of the Algorithm A1 is given in Figure 1:

Here, we define $c_i(W)$ to be true if and only if there exists $S_1 \subseteq F$ with $|S_1| = i$, $N(S_1) = W$, and for every $f, f' \in S_1$ with $f \neq f'$, $N(f) \cap N(f') = \emptyset$. Given a set family $\mathcal{W}$, we denote $\downarrow\mathcal{W} = \{X : \exists W \in \mathcal{W} \wedge X \subseteq W\}$ for the *down-closure* of $\mathcal{W}$, and $\uparrow\mathcal{W} = \{X : \exists W \in \mathcal{W} \wedge X \supseteq W\}$ for the *up-closure* of $\mathcal{W}$. The following lemma concerns the sub-routine invoked in Algorithm 1 and can be proved via known dynamic programming techniques, and is postponed to the full version.

▶ **Lemma 4.3.** *There exists an algorithm that given a bipartite graph $G = (F \dot\cup U, E)$ and $\mathcal{W} \subseteq 2^U$ with $|U| = n$ and $|F| = m$, computes $c_i(W)$ for all $W \in \mathcal{W}$ and $i \in n$ in $O(\text{poly}(n)|\downarrow\mathcal{W}|m)$ time.*

Thus, for further preparation of the proof of Theorem 4.2, we need to analyze the maximum size of the (down/up)-closure of $\mathcal{W}$ in Algorithm A1 in Figure 1:

▶ **Lemma 4.4.** *Let ζ, β be positive real numbers satisfying $2\sqrt{\beta} \leq \zeta < 1/4$ and $|U| = n$. Suppose $\mathcal{W} \subseteq \binom{U}{(1/2+\beta)n}$ with $|\mathcal{W}| \leq 2^{(1-\zeta)n}$. Then $|\uparrow\mathcal{W}|, |\downarrow\mathcal{W}| \leq n2^{(1-(\zeta/2)^4)n}$.*

Proof. The upper bound on the up-closure is directly obtained by using Part 1 of Lemma 2.1:

$$|\uparrow\mathcal{W}| \leq n\binom{n}{1/2+\beta} \leq n2^{h(1/2+\beta)n} \leq n2^{(1-\beta^2)n} \leq n2^{(1-(\zeta/2)^4)n}.$$

We continue with upper bounding the down closure. Let $\lambda \leq \beta$ and $w_\lambda = |\{W \in \downarrow\mathcal{W} : |W| = \lambda n\}|$, so $|\downarrow\mathcal{W}| \leq n \cdot \max_\lambda w_\lambda$. Then we have the following upper bounds:

$$w_\lambda \leq \binom{n}{\lambda n} \leq 2^{h(\lambda)n}, \qquad w_\lambda \leq |\mathcal{W}|\binom{(1/2+\beta)n}{\lambda n} \leq 2^{\left((1-\zeta)+h\left(\frac{\lambda}{1/2+\beta}\right)(1/2+\beta)\right)n}.$$

To see the second upper bound, note that any set $W \in \mathcal{W}$ can have at most $\binom{\beta n}{\lambda n}$ subsets of size λn. Thus, we see that $|\downarrow\mathcal{W}|/n$ is upper bounded by $2^{f(\zeta,\beta)n}$, where

$$f(\zeta, \beta) = \max_{\lambda \leq 1/2+\beta} \min\left\{h(\lambda), (1-\zeta) + h\left(\frac{\lambda}{1/2+\beta}\right)(1/2+\beta)\right\}.$$

The remainder of the proof is therefore devoted to upper bounding $f(\zeta,\beta)$. We establish this by evaluating both terms of the minimum, setting λ to be $\lambda' = (1-\zeta^2)(1/2+\beta)$. First note that by our assumption

$$\begin{aligned} \lambda' = (1-\zeta^2)(1/2+\beta) = 1/2 - \zeta^2/2 + \beta - \zeta^2\beta \leq 1/2 - \zeta^2/2 + \zeta^2/4 - \zeta^2\beta & \quad < 1/2, \\ \lambda'/(1/2+\beta) = 1-\zeta^2 & \quad > 1/2. \end{aligned}$$

Therefore, since $h(x)$ is increasing for $x < 1/2$, $h(\lambda) \leq h(\lambda')$ for $\lambda \leq \lambda'$. Similarly, $h\left(\frac{\lambda}{1/2+\beta}\right)$ is at most $h\left(\frac{\lambda'}{1/2+\beta}\right)$ for $\lambda \geq \lambda'$, and we may upper bound $f(\zeta,\beta)$ by the maximum of the two terms of the minimum in $f(\zeta,\beta)$ obtained by setting $\lambda = \lambda'$. For the first term of the minimum, note that by Lemma 2.1, Item 1:

$$\begin{aligned} h(\lambda') = h((1-\zeta^2)(1/2+\beta)) &\leq 1 - (1/2 - (1-\zeta^2)(1/2+\beta))^2 \\ &= 1 - \left(\zeta^2/2 - \beta + \beta\zeta^2\right)^2 \\ &\leq 1 - \left(\zeta^2/2 - \beta\right)^2 \\ &\leq 1 - (\zeta^2/4)^2 = 1 - (\zeta/2)^4. \end{aligned}$$

For the second term we have

$$\begin{aligned} 1 - \zeta + h\left(\frac{\lambda'}{1/2+\beta}\right)(1/2+\beta) &= 1 - \zeta + h(1-\zeta^2)(1/2+\beta) && \text{by Lemma 2.1, Item 2} \\ &= 1 - \zeta + h(\zeta^2)(1/2+\beta) && \beta < \tfrac{1}{64} \text{ by assumption} \\ &\leq 1 - \zeta + \zeta^2 \lg\left(\frac{4}{\zeta^2}\right)\frac{33}{64} && \zeta \lg\left(\tfrac{4}{\zeta^2}\right) \leq \tfrac{3}{2} \\ &\leq 1 - \zeta + \zeta\tfrac{3}{2}\cdot\tfrac{33}{64} \\ &\leq 1 - \zeta/10. \end{aligned}$$

note for the penultimate inequality that $\zeta\lg(\frac{4}{\zeta^2})$ is monotone increasing for $0 \leq \zeta \leq 1/4$ and substituting $\zeta = 1/4$ in this expression thus upper bounds it with $3/2$. ◀

Now we are ready to wrap up this section with the proof of Theorem 4.2:

Proof of Theorem 4.2. We can implement Line 3 by invoking the algorithm of Lemma 4.3 with both $|\mathcal{W}|$ and $\mathcal{W}' = \{W : [n]\setminus W \in \mathcal{W}\}$. Since $|{\downarrow}\mathcal{W}'| = |\{X : \exists W \in \mathcal{W} \wedge X \subseteq [n]\setminus W\}| = |{\uparrow}\mathcal{W}|$, this will take time $O(\text{poly}(n)(|{\downarrow}W| + |{\uparrow}\mathcal{W}|)m)$. This is clearly the bottleneck of the algorithm, so it remains to upper bound (the expectation of) $|{\downarrow}W| + |{\uparrow}\mathcal{W}|$ by applying Lemma 4.4. To do this, note that $W \subseteq \binom{n}{l}$, and we may assume $l \geq n/2$ since we could either apply the lemma using $\mathcal{W}$ or $\mathcal{W}'$. Moreover, we have that $l \leq (1/2+\beta)n$ and $2\sqrt{\beta} \leq \zeta$ by assumption so indeed Lemma 4.4 applies. On expectation $|\mathcal{W}| \leq \binom{n}{(1/2+\beta)n}2^{-\zeta n} \leq 2^{(1-\zeta)n}$, thus the running time[7] indeed is as claimed.

For the correctness, it is easily checked that the algorithm never returns false positives. Moreover, if there exist at least $\Omega(2^{\zeta n})$ witness β-halves then for some l in the loop of Line 1, there are at least $\Omega(2^{\zeta n}/n)$ witness halves of size l. Thus in this iteration we see by Lemma 2.1, Part 3 that

$$\Pr[\nexists \text{ witness halve } W \in \mathcal{W}] \leq \left(1 - \frac{1}{2^{\zeta n}}\right)^{\Omega(2^{\zeta n}/n)} \leq e^{-1/n}. \tag{4.1}$$

[7] Due to the sampling in Line 2, we actually only get an upper bound on the expectation of the running time, but by Markov's inequality we can simply ignore iterations where $\mathcal{W}$ exceeds twice the expectation.

and if a witness halve $W \in \mathcal{W}$ exists the algorithm returns **yes** since $c_i(W) \wedge c_{s-i}(U \setminus W)$ holds for some i by the definition of witness halve. Therefore, if we perform n independent trials of Algorithm $\mathtt{A}_1$ it return **yes** with probability at least $1 - 1/e$. ◀

4.2 Improvement in the case with exponentially many input sets

In this section we show that under some mild conditions, the existence of many witness halves can also be exploited in the presence of exponentially many sets. This largely builds upon machinery developed by Björklund et al. [10, 8]. To state our result as general as possible we assume the sets are given via an oracle so our running can be sublinear in the input if the number of sets is close to 2^n.

▶ **Theorem 4.5.** *There exists an algorithm that, given oracle access to an (n, m, s)-instance of* Set Partition *and real numbers $\beta, \zeta > 0$ satisfying $2\sqrt{\beta} \leq \zeta < 1/4$, runs in time $2^{(1-(\zeta/2)^4)n}\operatorname{poly}(n)T$ and has the following property: if there exist at least $\Omega(2^{\zeta n})$ witness β-halves, it outputs* **yes** *with constant probability and if there does not exist a set partition of size s it outputs* **no**.

Here the oracle algorithm accepts $X \subseteq U$ as input, and decides whether there exists $f \in F$ with $N(f) = X$ in time T.

The proof of Theorem 4.5 is identical to the proof of Theorem 4.2 (and therefore omitted), except that here we use the following lemma instead of Lemma 4.3:

▶ **Lemma 4.6.** *There exists an algorithm that, given $\mathcal{W} \subseteq 2^U$ and oracle access to a bipartite graph $G = (F \dot\cup U, E)$, computes the values $c_i(W)$ for all $W \in \mathcal{W}$ in $O(T|{\downarrow}\mathcal{W}|\operatorname{poly}(n))$ time. Here the oracle algorithm accepts $X \subseteq U$ as input, and decides whether there exists $f \in F$ with $N(f) = X$ in time T.*

This lemma mainly reiterates previous work developed by Björklund et al. [10, 8], but since they did not prove this lemma as such we include a proof here in the full version.

5 Finding Large Set Covers Faster

In this section we will use the tools of the previous sections to prove our main results, Theorems 1.1 and 1.2. We first connect the existence of large solutions to the existence of many witness halves in the following lemma:

▶ **Lemma 5.1.** *If an (n, m, s)-instance of* Set Partition *has no empty sets and satisfies $s \geq \sigma_0 n$ and $|N(f)| \leq \sigma_0^4 n/8$ for every $f \in F$, there is a solution if and only if there exist at least $2^{\sigma_0 n}/4$ witness $(\sigma_0^2/4)$-halves.*

Proof. Note that the backward direction is trivial since by definition the existence of a witness halve implies the existence of a solution.

For the other direction, suppose $S = \{f_1, \ldots, f_s\}$ is a set partition, and denote $d_i = |N(f_i)|$. Suppose $S' \subseteq S$ is obtained by including every element of S with probability $1/2$ in S'. since $N(f_i) \cap N(f_j) = \emptyset$ for $i \neq j$, we have that the random variable $|N(S')|$ is a sum of s independent random variables that equal 0 and d_i with probability $1/2$. By the Hoeffding bound (Lemma 2.2) we see that

$$\Pr\left[\Big| |N(S')| - \mathbb{E}[|N(S')|]\Big| \geq n\sigma_0^2/4\right] \leq 2 \cdot \exp\left(\frac{-n^2\sigma_0^4/8}{\sum_{e \in S} d_e^2}\right) \leq 2 \cdot \exp\left(\frac{-n^2\sigma_0^4/8}{n^2\sigma_0^4/8}\right) < \tfrac{3}{4},$$

```
Algorithm A2($G = (F \dot\cup U, E), \sigma$).
1: Ensure $|N(f)| \leq \sigma^4 n/1000$ using Observation 2.4.
2: for every integer $s$ satisfying $\lfloor \sigma n/2 \rfloor \leq s \leq \sigma n$ do
3:     Create an $(n, m', s)$-instance $((F' \dot\cup U, E), s)$ of Set Partition where $F'$ is con-
       structed by adding a vertex $f'$ with $N(f') = X$ for all $f \in F, X \subseteq N(f)$.
4:     Let $\sigma_0 = s/n$.
5:     if A1$((F' \dot\cup U, E), s, \sigma_0, \sigma_0^2/4)$ = yes then return yes.
6: Pick an arbitrary subset $X \in \binom{U}{n/2}$.
7: Find the sizes $l$ and $r$ of the smallest set covers in the instances induced by elements
   $X$ and respectively elements $U \setminus X$ in $O(2^{n/2}\,\mathrm{poly}(n)m)$ time with standard dynamic
   programming.
8: if $l + r <= \sigma n$ then return yes else return no.
```

Figure 2 Algorithm for Set Cover large solutions (implementing Theorem 1.1).

where the second inequality follows from $d_e \leq \sigma_0^4 n/8$ and $\sum_{e \in S} d_e = n$. So for at least $2^{|S|}/4 \geq 2^{\sigma_0 n}/4$ subsets $S' \subseteq S$ we have that $|N(S')| \in (\frac{1}{2} \pm \sigma_0^2/4)n$. Thus, since for each such S', $N(S')$ determines S' and thus gives rise to a distinct witness halve, there are at least $2^{\sigma_0 n}/4$ witness $(\sigma_0^2/4)$-halves. ◀

Now we are ready to prove the first main theorem, which we recall here for convenience.

▶ **Theorem 1.1** (restated). *There is a Monte Carlo algorithm that takes an instance of* Set Cover *on n elements and m sets and an integer s as input and determines whether there exists a set cover of size s in $O(2^{(1-\Omega(\sigma^4))n}m)$ time, where $\sigma = s/n$.*

Proof. The algorithm implementing Theorem 1.1 is given in Figure 2.

We first focus on the correctness of this algorithm. It is clear that the algorithm never returns false positives on Line 5 since Algorithm A1 also has this property. If **yes** is returned on Line 8 it is also clear there exists a solution.

Now suppose that a set cover S of size at most σn exists. First suppose $\sigma n/2 \leq |S| \leq \sigma n$. We consider $s = |S|$ in some iteration of the loop on Line 2. Notice that now in Line 3 we have reduced the problem to a yes-instance of Set Partition without empty sets satisfying $|N(f)| \leq \sigma^4 n/1000$ for every $f \in F$. Therefore Lemma 5.1 applies with $\sigma_0 \geq \sigma/2$ and we see there are at least $2^{\sigma_0 n}/4$ witness $(\sigma_0^2/4)$-halves. Thus, we can apply Theorem 4.2 with $\zeta = \sigma_0$ and $\beta = \sigma_0^2/4$ to find the set S with constant probability, since $\beta \leq (\zeta/2)^2$.

Now suppose $|S| \leq \sigma n/2$. Then picking every element in S twice is a solution (as a multiset), and it implies that for every $X \subseteq U$ the sizes of the smallest set covers l and r (as defined in the algorithm) satisfy $l + r \leq \sigma n$. Thus Lines 6-8 find such a set and the algorithm returns **yes**.

For the running time, Line 1 takes at most $O(2^{(1-\sigma^4/1000)n}\,\mathrm{poly}(n)m)$ due to Observation 2.4. For Line 5, due to Theorem 4.2 this runs in time

$$\begin{aligned} O(2^{(1-(\zeta/2)^4)n}\,\mathrm{poly}(n)m') &= O(2^{(1-(\sigma_0/2)^4)n}\,\mathrm{poly}(n)2^{\sigma^4 n/1000}m) \\ &\leq O(2^{(1-(\sigma/4)^4)n}\,\mathrm{poly}(n)2^{\sigma^4 n/1000}m) \\ &= O(2^{(1+\sigma^4(1/1000-1/4^4))n}\,\mathrm{poly}(n)m) \\ &\leq O(2^{(1-\Omega(\sigma^4))n}\,\mathrm{poly}(n)m). \end{aligned}$$

as claimed in the theorem statement. ◀

As a more direct consequence of the tools of the previous section we also get the following result for Set Partition:

▶ **Theorem 5.2.** *There exists a Monte Carlo algorithm for* Set Partition *that, given oracle access to an $(n, m, \sigma n)$-instance satisfying $0 < |N(f)| \leq \sigma^4 n/8$ for every $f \in F$, runs in $2^{(1-\Omega(\sigma^4))n}\,\mathrm{poly}(n)T$ time.*

Here the oracle algorithm accepts $X \subseteq U$ as input, and decides whether there exists $f \in F$ with $N(f) = X$ in time T.

Proof. Lemma 5.1 implies the instance is a YES-instance if and only if there exist $2^{\sigma n}/4$ witness $(\sigma^2/4)$-halves. Thus Theorem 4.5 implies the theorem statement. ◀

Note that this theorem also implies an $O((m + 2^{(1-\Omega(\sigma^4))n})\,\mathrm{poly}(n))$ time algorithm for Set Partition where the sets are given explicitly because we can construct a binary search tree after which we can implement the oracle to run in $T = n$ query time. We remark that it would be interesting to see whether the assumption $|N(f)| \leq \sigma^2 n/4$ is needed, but removing this assumption seems to require more ideas than the ones from this work: For example if the solution has three sets of size $3n/10$ there will be no witness halve that is sufficiently balanced, and alternatively using Observation 2.4 seems to be too slow.

However, if we settle for a additive 1-approximation we can deal with this issue in a simple way and have as a particular consequence the second result mentioned in the beginning of this paper:

▶ **Theorem 1.2** (restated)**.** *There is a randomized algorithm that given graph G and integer $s = \sigma n$, in $O^*(2^{(1-\Omega(\sigma^4))n})$ time outputs* **yes** *with constant probability, if $\chi(G) < s$, and* **no***, if $\chi(G) > s$.*

Proof. Let $G = (V, E)$ and define a Set Partition instance where for every independent set $I \subseteq V$ of G there is an element $f \in F$ with $N(f) = I$. It is easy to see that this instance of Set Partition has a solution of size s if and only if $\chi(G) \leq s$.

Check in $\binom{n}{\sigma^4 n/8}$ time whether G has an independent set of size $\sigma^4 n/8$. If such an independent set is found, remove this set from the graph and return yes if the obtained graph has a $(k-1)$-coloring and no otherwise. Using the $O^*(2^n)$ time algorithm by Björklund et al. [10] in the second step, this procedure clearly runs in time $O^*(2^{(1-\Omega(\sigma^4))n})$, and always finds a coloring using at most one more color than the minimum number of colors if a large enough independent set exists.

On the other hand, if the maximum independent set of G is of size at most $\sigma^4 n/8$, we may apply Theorem 5.2 with $T = \mathrm{poly}(n)$ since it can be verified in polynomial time whether a given $X \subseteq V$ is an independent set, and the theorem follows. ◀

6 Directions for Further Research

In this section, we relate the work presented to some notorious open problems. The obvious open question is to determine the exact complexity of the Set Cover problem:

▶ **Open Problem 1.** *Can* Set Cover *be solved in time $O^*((2-\Omega(1))^n)$?*

This question was already stated at several places. It is known that if a version of Set Cover where the number of solutions modulo 2 is counted can be solved in $(2-\Omega(1))^n$ the Strong Exponential Time Hypothesis fails. We refer to [14], for more details.

Less ambitiously, it is natural to wonder whether our dependency on σ can be improved. Our algorithm and analysis seem loose, but we feel the gain of a sharpening this analysis does not outweigh the technical effort currently: For a better dependence, we need both a better bound in Lemma 4.4 and to reduce the set sizes more efficiently than in Observation 2.4. As further research we suggest to find a different algorithmic way to deal with the case where many witness halves are unbalanced. But this alone will not suffice to give linear dependence in σ since in Lemma 4.4 we do not expect to get linear dependence on ζ even if $\beta = 0$. It would also be interesting to see which other instances of SET COVER can be solved in $O^*((2 - \Omega(1))^n)$ time. One that might be worthwhile studying is whether this includes instances with optimal set covers in which the sum of the set sizes is at least $(1 + \Omega(1))n$; one may hope to find exponentially many (balanced) witness halves here as well.

In [14], the authors also give a reduction from SUBSET SUM to SET PARTITION. The exact complexity of SUBSET SUM with small integers is also something we explicitly like to state as open problem here, especially since the $O^*(t)$ time algorithm (where t is the target integer) is perhaps one of the most famous exponential time algorithms:

▶ **Open Problem 2.** *Can* SUBSET SUM *with target t be solved in time $O^*(t^{1-\Omega(1)})$, or can we exclude the existence of such an assuming the Strong Exponential Time Hypothesis?*

Note this question was before asked in [24] by the present author. It would be interesting to study the complexity of SUBSET SUM in a similar vein as we did in this paper: are there some special properties allowing a faster algorithm? For example, a faster algorithm for instances of high ‘density’ (e.g., $n/\lg t$) may be used for improving an algorithm of Horowitz&Sahni [22]. Note that here the ‘density’ of a SUBSET SUM instance is the inverse of what one would expect when relating to the definition of density of k-CNF formula.

Another question that has already open for a while is:

▶ **Open Problem 3.** *Can* GRAPH COLORING *be solved in time $O^*(2^{(1-\Omega(1))n})$?*

Could the techniques of this paper be used to make progress towards resolving this question? While our algorithm seems to benefit from the existence of many optimal colorings, in an interesting paper Björklund [5] actually shows that the existence of *few* optimal colorings can be exploited in graphs of small pathwidth. Related to this is also the Hamiltonicity problem. In our current understanding this problem becomes easier when there is a promise that there are few Hamiltonian cycles (see [6], but also e.g. [13] allows derandomizations of known probabilistic algorithms in this case), so a natural approach would be to deal explicitly with instances with many solutions by sampling dynamic programming table in a similar vein as done in this paper.

Acknowledgements. The author would like to thank Per Austrin, Petteri Kaski, Mikko Koivisto for their collaborations resulting in [1, 2] that mostly inspired this work, Karl Bringmann for discussions on applications of [2] to SET PARTITION, and anonymous reviewers for their useful comments.

References

1 Per Austrin, Petteri Kaski, Mikko Koivisto, and Jesper Nederlof. Subset sum in the absence of concentration. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, volume 30 of *LIPIcs*, pages 48–61. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2015. URL: `http://www.dagstuhl.de/dagpub/978-3-939897-78-1`, `doi:10.4230/LIPIcs.STACS.2015.48`.

2 Per Austrin, Petteri Kaski, Mikko Koivisto, and Jesper Nederlof. Dense subset sum may be the hardest. In Nicolas Ollinger and Heribert Vollmer, editors, *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, volume 47 of *LIPIcs*, pages 13:1–13:14. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2016. `doi:10.4230/LIPIcs.STACS.2016.13`.

3 Anja Becker, Jean-Sébastien Coron, and Antoine Joux. Improved generic algorithms for hard knapsacks. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011 – 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*, pages 364–385. Springer, 2011. `doi:10.1007/978-3-642-20465-4_21`.

4 Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. Decoding random binary linear codes in 2 n/20: How 1 + 1 = 0 improves information set decoding. In *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 520–536. Springer, 2012. Talk at `http://www.iacr.org/cryptodb/data/paper.php?pubkey=24271`. `doi:10.1007/978-3-642-29011-4_31`.

5 Andreas Björklund. Uniquely coloring graphs over path decompositions. *CoRR*, abs/1504.03670, 2015. URL: `http://arxiv.org/abs/1504.03670`.

6 Andreas Björklund, Holger Dell, and Thore Husfeldt. The parity of set systems under random restrictions with applications to exponential time problems. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming – 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, volume 9134 of *Lecture Notes in Computer Science*, pages 231–242. Springer, 2015. `doi:10.1007/978-3-662-47672-7_19`.

7 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Narrow sieves for parameterized paths and packings. *CoRR*, abs/1007.1161, 2010. URL: `http://arxiv.org/abs/1007.1161`.

8 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Trimmed moebius inversion and graphs of bounded degree. *Theory Comput. Syst.*, 47(3):637–654, 2010. `doi:10.1007/s00224-009-9185-7`.

9 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. The traveling salesman problem in bounded degree graphs. *ACM Transactions on Algorithms*, 8(2):18, 2012. `doi:10.1145/2151171.2151181`.

10 Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set partitioning via inclusion-exclusion. *SIAM J. Comput.*, 39(2):546–563, 2009. `doi:10.1137/070683933`.

11 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. A duality between clause width and clause density for SAT. In *21st Annual IEEE Conference on Computational Complexity (CCC 2006), 16-20 July 2006, Prague, Czech Republic*, pages 252–260. IEEE Computer Society, 2006. `doi:10.1109/CCC.2006.6`.

12 Timothy M. Chan and Ryan Williams. Deterministic apsp, orthogonal vectors, and more: Quickly derandomizing razborov-smolensky. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1246–1255. SIAM, 2016. `doi:10.1137/1.9781611974331.ch87`.

13 Suresh Chari, Pankaj Rohatgi, and Aravind Srinivasan. Randomness-optimal unique element isolation with applications to perfect matching and related problems. *SIAM J. Comput.*, 24(5):1036–1050, 1995. `doi:10.1137/S0097539793250330`.

14 Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On problems as hard as CNF-SAT. In *Proceedings of the 27th Conference on Computational Complexity, CCC 2012, Porto, Portugal, June 26-29, 2012*, pages 74–84. IEEE, 2012. doi:10.1109/CCC.2012.36.

15 Marek Cygan and Marcin Pilipczuk. Faster exponential-time algorithms in graphs of bounded average degree. *Inf. Comput.*, 243:75–85, 2015. doi:10.1016/j.ic.2014.12.007.

16 Vilhelm Dahllöf. *Exact Algorithms for Exact Satisfiability Problems.* PhD thesis, Linköping University, TCSLAB, The Institute of Technology, 2006.

17 Evgeny Dantsin, Andreas Goerdt, Edward A Hirsch, Ravi Kannan, Jon Kleinberg, Christos Papadimitriou, Prabhakar Raghavan, and Uwe Schöning. A deterministic $(2-2/(k+1))n$ algorithm for k-SAT based on local search. *Theoretical Computer Science*, 289(1):69–83, 2002. doi:10.1016/S0304-3975(01)00174-8.

18 Limor Drori and David Peleg. Faster exact solutions for some NP-hard problems. *Theoretical Computer Science*, 287(2):473–499, 2002. Algorithms. doi:10.1016/S0304-3975(01)00257-2.

19 Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. A measure & conquer approach for the analysis of exact algorithms. *J. ACM*, 56(5), 2009. doi:10.1145/1552285.1552286.

20 Alexander Golovnev, Alexander S. Kulikov, and Ivan Mihajlin. Families with infants: A general approach to solve hard partition problems. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming – 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 551–562. Springer, 2014. doi:10.1007/978-3-662-43948-7_46.

21 Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

22 Ellis Horowitz and Sartaj Sahni. Computing partitions with applications to the knapsack problem. *J. ACM*, 21(2):277–292, 1974. doi:10.1145/321812.321823.

23 Nick Howgrave-Graham and Antoine Joux. New generic algorithms for hard knapsacks. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 – June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 235–256. Springer, 2010. doi:10.1007/978-3-642-13190-5_12.

24 Thore Husfeldt, Ramamohan Paturi, Gregory B. Sorkin, and Ryan Williams. Exponential Algorithms: Algorithms and Complexity Beyond Polynomial Time (Dagstuhl Seminar 13331). *Dagstuhl Reports*, 3(8):40–72, 2013. doi:10.4230/DagRep.3.8.40.

25 Russell Impagliazzo, Shachar Lovett, Ramamohan Paturi, and Stefan Schneider. 0-1 integer linear programming with a linear number of constraints. *CoRR*, abs/1401.5512, 2014. URL: http://arxiv.org/abs/1401.5512.

26 Antoine Joux. *Algorithmic Cryptanalysis.* Chapman & Hall/CRC, 1st edition, 2009.

27 Petteri Kaski, Mikko Koivisto, and Jesper Nederlof. Homomorphic hashing for sparse coefficient extraction. In Dimitrios M. Thilikos and Gerhard J. Woeginger, editors, *Parameterized and Exact Computation – 7th International Symposium, IPEC 2012, Ljubljana, Slovenia, September 12-14, 2012. Proceedings*, volume 7535 of *Lecture Notes in Computer Science*, pages 147–158. Springer, 2012. doi:10.1007/978-3-642-33293-7_15.

28 Mikko Koivisto. Partitioning into sets of bounded cardinality. In Jianer Chen and Fedor V. Fomin, editors, *Parameterized and Exact Computation, 4th International Workshop, IWPEC 2009, Copenhagen, Denmark, September 10-11, 2009, Revised Selected Papers*, volume 5917 of *Lecture Notes in Computer Science*, pages 258–263. Springer, 2009. doi:10.1007/978-3-642-11269-0_21.

29 Daniel Paulusma, Friedrich Slivovsky, and Stefan Szeider. Model counting for cnf formulas of bounded modular treewidth. *Algorithmica*, pages 1–27, 2015. doi:10.1007/s00453-015-0030-x.

30 Sigve Hortemo Sæther, Jan Arne Telle, and Martin Vatshelle. Solving maxsat and #sat on structured CNF formulas. In Carsten Sinz and Uwe Egly, editors, *Theory and Applications of Satisfiability Testing – SAT 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8561 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2014. doi:10.1007/978-3-319-09284-3_3.

31 Uwe Schöning. A probabilistic algorithm for k-SAT based on limited local search and restart. *Algorithmica*, 32(4):615–623, 2002. doi:10.1007/s00453-001-0094-7.

32 Vijay V. Vazirani. *Approximation Algorithms*. Springer-Verlag New York, Inc., New York, NY, USA, 2001.

Graph Isomorphism for Unit Square Graphs

Daniel Neuen

RWTH Aachen University, Aachen, Germany
neuen@informatik.rwth-aachen.de

Abstract

In the past decades for more and more graph classes the Graph Isomorphism Problem was shown to be solvable in polynomial time. An interesting family of graph classes arises from intersection graphs of geometric objects. In this work we show that the Graph Isomorphism Problem for unit square graphs, intersection graphs of axis-parallel unit squares in the plane, can be solved in polynomial time. Since the recognition problem for this class of graphs is NP-hard we can not rely on standard techniques for geometric graphs based on constructing a canonical realization. Instead, we develop new techniques which combine structural insights into the class of unit square graphs with understanding of the automorphism group of such graphs. For the latter we introduce a generalization of bounded degree graphs which is used to capture the main structure of unit square graphs. Using group theoretic algorithms we obtain sufficient information to solve the isomorphism problem for unit square graphs.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

Keywords and phrases graph isomorphism, geometric graphs, unit squares

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.70

1 Introduction

The Graph Isomorphism Problem is one of the most famous open problems in theoretical computer science. In the past three decades the problem was intensively studied but only recently the upper bound on the complexity could be improved to quasipolynomial time [2]. However, it is still open whether the Graph Isomorphism Problem can be solved in polynomial time. In this work we focus on geometric graph classes, that is, graph classes that arise as intersection graphs of geometric objects. In an intersection graph the vertices are identified with geometric objects and two vertices are connected if the corresponding objects intersect.

One of the most basic geometric graph classes is the class of interval graphs, intersection graphs of intervals on the real line. Although this graph class is quite restrictive there are a number of practical applications and specialized algorithms for interval graphs (see e.g. [15]). The Graph Isomorphism Problem on interval graphs can be solved in linear time [11] as well as in logarithmic space [18]. However, for several generalizations of interval graphs the complexity of the Graph Isomorphism Problem is unknown. This includes for example circular arc graphs (see [12]) and triangle graphs (see [30]). On the other hand a graph class is GI-complete if the Graph Isomorphism Problem for this class is as difficult as the general problem under polynomial time reductions. An example of a GI-complete geometric class is the class of grid intersection graphs, bipartite intersection graphs of horizontal and vertical line segments in the plane [29]. As an immediate consequence the class of intersection graphs of axis-parallel rectangles is also GI-complete. Unit square graphs, intersection graphs of axis-parallel unit squares, are a natural restriction for the rectangle graphs. This raises the question for the complexity of the isomorphism problem for unit square graphs. In this work we prove that the Graph Isomorphism Problem for unit square graphs can be solved in polynomial time. Besides being a natural restriction to rectangle graphs, another central motivation to study this problem comes from unit disk graphs, intersection graphs of unit circles in the plane. Unit disk graphs where first studied by Clark, Colbourn and Johnson in [10] and for several problems specialized algorithms have been proposed (see e.g. [13]). Practical applications arise for example from broadcast networks where each broadcast station is represented by a vertex and two stations communicate with each other if the distance between them does not exceed the broadcast range. In the work of Clark et al. two problems, namely the recognition problem and the isomorphism problem, were left open. While recognition of unit disk graphs proved to be NP-hard [8], the isomorphism problem for unit disk graphs is still open. Unit square graphs present a natural variant to unit disk graphs as we just replace the Euclidean norm by the Manhattan norm. Also, going from unit disks to unit squares removes geometric intricacies and tends to simplify the structure of graphs but maintains several key aspects of the problem. In particular, for unit disk as well as unit square graphs vertices only have a bounded number of independent neighbors (set of pairwise non-adjacent neighbors) and the structure of graphs seems to be a mixture of bounded degree and planarity. Hence, solving the isomorphism problem for unit square graphs might be a step towards solving the same problem for unit disk graphs. Furthermore, in [29, 30] Uehara also asked for the complexity of graph isomorphism for unit grid intersection graphs. Unit grid intersection graphs can be seen as bipartite versions of unit square graphs in the following sense: A bipartite graph is a unit grid intersection graph if and only if it is the intersection graph of unit squares where intersections between squares belonging to vertices on the same side of the bipartition are ignored. Hence, the result presented in this work shows that in some sense the difficulty for unit grid intersection lies in recreating the information which lines are close to each other.

Another interesting point arises from the fact that, like for unit disk graphs, recognition of unit square graphs is NP-hard [7]. Hence, we obtain an example of a natural graph class with the interesting property that isomorphism tests can be performed in polynomial time whereas recognition is NP-hard. Also, the hardness result rules out the classical approach to attack the isomorphism problem. Typically, isomorphism tests for geometric graphs are based on constructing a canonical geometric representation of the graph (see e.g. [18, 20]) but, as this would also solve the recognition problem, such an approach is not possible here. Instead, our algorithm combines group theoretic techniques with geometric properties of unit square graphs. For the group theoretic machinery we extend the results developed by Luks [22] to decide isomorphism for bounded degree graphs by also allowing for example large cycles in the neighborhood of a vertex. On a geometric level this coincides in some sense with the intuition that vertices in the neighborhood of some fixed vertex are cyclically arranged around the central vertex. Using geometric properties of unit square graphs and known algorithms for other geometric graph classes such as proper circular arc graphs we can canonically (in an isomorphism-invariant way) extract such circular orderings, which can then be used by the group theoretic machinery. For this, we show a series of results giving a deep insight into the structure of neighborhoods of single vertices and neighborhoods of cliques within unit square graphs. These results not only help us to understand the structure of unit square graphs, but also we obtain significant knowledge about the structure of the automorphism group of a unit square graph. However, an obvious obstacle to this approach comes from large cliques which are connected in a uniform way to the rest of the graph and do not contain any significant structure. More precisely, such cliques may be responsible for large symmetric groups which are subgroups of the automorphism group of the whole graph. Since large symmetric groups form a clear obstacle to the group theoretic machinery and can not be handled by Luks' algorithm we have to cope with these parts of the graph in a different way. Our second main result on the structure of unit square graphs characterizes connections between cliques which are stable with respect to the color refinement algorithm (see e.g. [6, 24]), and also establishes a close connection to interval graphs. Building on this characterization we show that the color refinement algorithm can be used to cope with the symmetric parts of the graph containing no significant structure. Finally, combining the color refinement algorithm with the group theoretic machinery, we obtain an algorithm to solve the isomorphism problem for unit square graphs.

2 Preliminaries

2.1 Graphs

A *graph* is a pair $G = (V, E)$ with vertex set $V = V(G)$ and edge set $E = E(G)$. In this paper all graphs are undirected, so $E(G)$ is always irreflexive and symmetric. The *(open) neighborhood* of a vertex $v \in V(G)$ is the set $N_G(v) = N(v) = \{w \in V(G) \mid vw \in E(G)\}$ and the size of $N(v)$ is the degree of v. The *closed neighborhood* is the set $N[v] = N(v) \cup \{v\}$. Two vertices $v, w \in V(G)$ are *connected twins* if $N[v] = N[w]$. The corresponding equivalence relation, where two vertices are related if they are connected twins, will be called the *connected twins relation*. A *path* from v to w of length m is a sequence $u_0, \ldots, u_m$ of distinct vertices with $u_0 = v$ and $u_m = w$, such that $u_{i-1}u_i \in E(G)$ for each $i \in [m] := \{1, \ldots, m\}$. The *distance* between v and w, $d(v, w)$, is the length of a shortest path from v to w. A colored graph is a tuple $G = (V, E, c)$ where $c\colon V(G) \to \mathbb{N}$ assigns each vertex a unique color. For each color $i \in \mathbb{N}$ let $V_i(G) = \{v \in V(G) \mid c(v) = i\}$. Two graphs G and H are *isomorphic* ($G \cong H$) if there is a bijection $\varphi\colon V(G) \to V(H)$, such that $vw \in E(G)$ if and only if $\varphi(v)\varphi(w) \in E(H)$ for all $v, w \in V(G)$. In this case the mapping φ is an *isomorphism* from G to H. In case the input graphs are colored it is demanded that the isomorphism also preserves the coloring of the vertices. The Graph Isomorphism Problem asks whether two given graphs G and H are isomorphic. An isomorphism from a graph to itself is called an *automorphism*. The set of automorphisms of a graph G, denoted by $\mathrm{Aut}(G)$, forms a subgroup of the symmetric group over the vertex set.

2.2 Color Refinement

A very basic and fundamental method, which is a basic building block of many isomorphism tests, is the color refinement algorithm (see e.g. [24]). The basic idea is to iteratively distinguish vertices if they have a different number of neighbors in some color. A partition $\mathcal{P}$ of the vertices is *stable* if for all $X, Y \in \mathcal{P}$ and all $v, w \in X$ it holds that $|N(v) \cap Y| = |N(w) \cap Y|$. Further a partition $\mathcal{P}$ *refines* another partition $\mathcal{Q}$ if for each $X \in \mathcal{P}$ there is some $Y \in \mathcal{Q}$ with $X \subseteq Y$. The color refinement algorithm computes the unique coarsest stable partition refining the initial color partition (i.e. the partition of the vertices according to their color). The coarsest stable partition can be computed in almost linear time (see [24, 6]). We say color refinement distinguishes two graphs if there is some class in the coarsest stable partition on the disjoint union of the graphs that contains a different number of vertices from the two graphs. In this case the two input graphs are not isomorphic.

The k-dimensional Weisfeiler-Leman algorithm is a generalization of the color refinement algorithm (cf. [9]). Instead of coloring only single vertices, the Weisfeiler-Leman algorithm colors k-tuples of vertices. Initially each tuple is colored with the isomorphism type of the underlying induced subgraph and then the coloring is refined in a similar way as by the color refinement algorithm (see [9] for a detailed description). We say k-dimensional Weisfeiler-Leman *identifies* a graph class $\mathcal{C}$ if for every pair of non-isomorphic graphs G, H with $G \in \mathcal{C}$ the k-dimensional Weisfeiler-Leman distinguishes between G and H.

2.3 Group Theory

In this subsection we briefly introduce the main group theoretic tools used in this work. For a general introduction to group theory we refer to [25] whereas several group theoretic algorithms are given in [16, 27]. Since we only deal with automorphism groups of graphs we can restrict ourselves to permutation groups which throughout this work will always be represented by generating sets of size polynomial in the size of the permutation domain. For a set Ω let $\mathrm{Sym}(\Omega)$ be the symmetric group over the set Ω. In particular we require a certain subclass of permutation groups, namely groups with bounded non-abelian composition factors. Let Γ be a group. A *normal series* is a sequence of subgroups $\Gamma = \Lambda_0 \trianglerighteq \Lambda_1 \trianglerighteq \dots \trianglerighteq \Lambda_k = \{1\}$. The length of the series is k and the groups Λ_{i-1}/Λ_i are the factor groups of the series, $i \in [k]$. A *composition series* is a strictly decreasing normal series of maximal length. For every finite group Γ all composition series have the same family of factor groups considered as a multi-set (cf. [25]). A *composition factor* of a finite group Γ is a factor group of a composition series of Γ.

▶ **Definition 2.1.** Let $d \in \mathbb{N}$. The family Γ_d contains all finite groups Γ for which all non-abelian composition factors are isomorphic to subgroups of $S_d = \mathrm{Sym}([d])$.

The class of Γ_d-groups is closed under subgroups and homomorphic images. Furthermore, for groups $N \trianglelefteq \Gamma$ it holds that $\Gamma \in \Gamma_d$ if and only if $N \in \Gamma_d$ and $\Gamma/N \in \Gamma_d$ (cf. [22]). A group Γ is *solvable* if every composition factor is abelian. Two examples of solvable groups, that are particularly important for this work, are cyclic groups and dihedral groups which are the automorphism groups of directed cycles and undirected cycles. Note that every solvable group is a Γ_d-group for every $d \in \mathbb{N}$.

The Setwise Stabilizer Problem asks, given a permutation group $\Gamma \leq \mathrm{Sym}(\Omega)$ and $A \subseteq \Omega$, for a generating set of the group $\mathrm{Stab}_\Gamma(A) := \{\gamma \in \Gamma \mid A = A^\gamma\}$, where $A^\gamma = \{\alpha^\gamma \mid \alpha \in A\}$ and α^γ is the image of α under the permutation γ. A central motivation to consider Γ_d-groups is the following result.

▶ **Theorem 2.2.** *Let $d \in \mathbb{N}$. The Setwise Stabilizer Problem for groups in Γ_d can be solved in polynomial time.*

A weaker version of this statement was proved by Luks in [22] considering only groups where all composition factors are isomorphic to subgroups of S_d. For the more general version stated above we refer to [1]. This result is for example used by Luks to solve graph isomorphism for graphs of bounded degree [22], but it can also be applied to more general graph classes such as t-bounded graphs (see e.g. [4]). For this work we introduce a slight variation, namely graphs which we call *t-circle-bounded* graphs. For a graph G and a set $X \subseteq V(G)$ we write $G[X]$ to denote the induced subgraph of G with vertex set X.

▶ **Definition 2.3.** A colored graph $G = (V, E, c)$ with $c\colon V(G) \to [k]$ is *t-circle-bounded* if for each $i \in [k]$ and $X \subseteq V_{<i} := \bigcup_{j<i} V_j(G)$ the graph

$$G_{i,X} := G[\{v \in V_i(G) \mid N(v) \cap V_{<i} = X\}]$$

is the disjoint union of at most t connected graphs of maximum degree two.

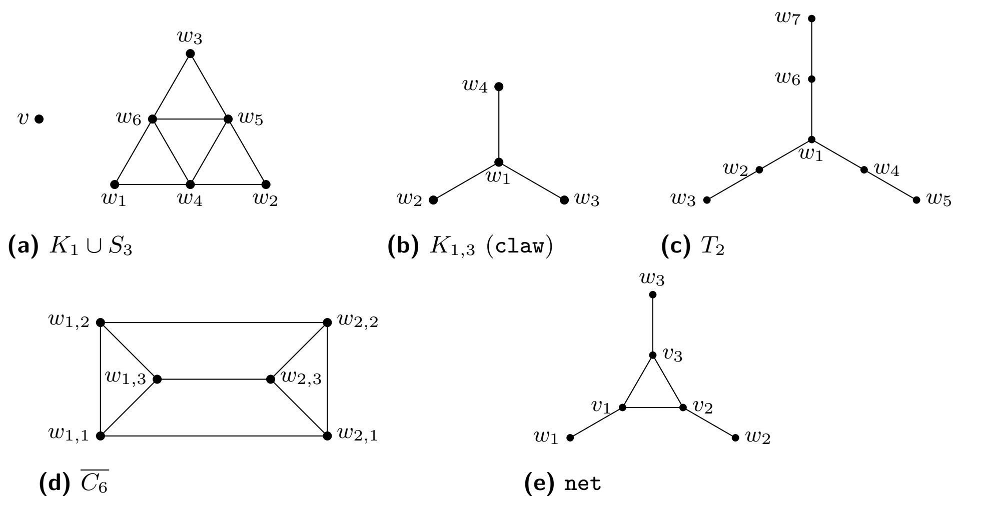


Figure 1 Some forbidden induced subgraphs of proper circular arc graphs.

The t-circle-bounded graphs are closely related to t-bounded graphs which are similarly defined with the size of $G_{i,X}$ bounded by t (see [4]). From an algorithmic point of view we can use very similar methods for the isomorphism problem on t-circle-bounded graphs as for t-bounded graphs.

▶ **Theorem 2.4.** *Let G be a t-circle-bounded graph. Then* $\text{Aut}(G) \in \Gamma_t$.

▶ **Theorem 2.5.** *The Graph Isomorphism Problem for t-circle-bounded graphs can be solved in polynomial time.*

Both theorems can be proved in very similar fashion as the respective statements for t-bounded graphs. In fact the only additional argument required for t-circle-bounded graphs is that the automorphism groups of connected graphs of maximum degree two are solvable and thus elements of Γ_t.

2.4 Proper circular arc graphs

A graph G is a *unit interval graph* if G is the intersection graph of unit intervals on the real line. A graph G is a *proper circular arc graph* if G is the intersection graph of arcs on a circle, such that for no two arcs one is properly contained in the other. A characterization of unit interval graphs and proper circular arc graphs in terms of forbidden induced subgraphs is given in [28]. For our purposes the following statements are sufficient. Some relevant forbidden induced subgraphs are also depicted in Figure 1.

▶ **Theorem 2.6.** *A graph G is a unit interval graph if and only if there are no induced subgraphs isomorphic to C_{n+4} for $n \geq 0$, S_3, $K_{1,3}$ and* `net`.

Here, C_n denotes a cycle of length n. Furthermore we denote by $G \cup H$ the disjoint union of G and H and the graph $\overline{G}$ is the complement graph of G.

▶ **Lemma 2.7.** *Let G be a graph, such that $N_G[v]$ induces a unit interval graph for every $v \in V(G)$ and there are no induced subgraphs isomorphic to $K_1 \cup C_{n+4}$ for $n \geq 0$, $K_1 \cup S_3$, $\overline{T_2}$, $\overline{C_6}$ and* `net`. *Then G is a proper circular arc graph.*

We also require the following characterization for proper circular arc graphs. A vertex is *universal* if it is adjacent to all other vertices. A graph G is *twin-free* if it does not contain connected twins and G is *co-bipartite* if $\overline{G}$ is bipartite.

▶ **Theorem 2.8** ([19], Theorem 3)**.** *Let G be a graph without universal vertices. Then G is a proper circular arc graph if and only if there is a cycle H with $V(G) = V(H)$, such that*

1. *$N_G[v]$ induces a connected subgraph in H for each $v \in V(G)$*
2. *For all $v, w \in V(G)$ it holds that if $N_G[v] \subseteq N_G[w]$ then the two paths share an endpoint in H.*

Furthermore, if G is connected, twin-free and not co-bipartite, the cycle H is unique [17]. Additionally, given some proper circular arc graph, a cycle H can be computed in polynomial time (see [19]). This gives us the following result.

▶ **Theorem 2.9.** *Let G be a connected proper circular arc graph, such that $\overline{G}$ is not bipartite. Further let $\sim_G$ be the connected twins relation and $\mathcal{P}$ the corresponding partition into equivalence classes. Then one can compute in polynomial time a canonical connected graph H, such that $V(H) = \mathcal{P}$ and H has maximum degree two.*

3 Basic Properties

For unit square graphs there are several possible definitions. The most obvious one is to describe vertices by axis-parallel unit squares with edges connecting two vertices if the unit squares intersect. Alternatively it can also be demanded that vertices represented by unit squares are connected if the center of the first square is contained in the other square. Another possibility is to describe vertices by points in the plane. Note that two squares with unit side length intersect if and only if the distance between their centers using the maximum norm is at most one. Thus, two vertices are connected if the distance between the points is at most one using the maximum norm. Furthermore, a unit square contains the center point of another unit square if and only if the distance between both centers is at most one half using the maximum norm. By applying a scaling argument this also gives us the equivalence to the second definition. In this paper we work with the last definition, that is we represent vertices by points in the plane. For a point $p \in \mathbb{R}^k$ we denote by p_i the i-th component of p, $i \in [k]$. The L_∞-norm is defined as $\|p\|_\infty = \max_{i\in[k]} |p_i|$.

▶ **Definition 3.1.** A *k-dimensional L_∞-realization* of a graph G is a mapping $f : V(G) \to \mathbb{R}^k$ such that $vw \in E(G)$ if and only if $\|f(v) - f(w)\|_\infty \leq 1$ for all $v, w \in V(G)$. A *unit square graph* is a graph having a two-dimensional L_∞-realization.

Observe that graphs with 1-dimensional L_∞-realization are exactly the unit interval graphs. For the remainder of this paper we focus on unit square graphs and just use the term realization for a two-dimensional L_∞-realization. Following our previous notation, for a realization $f : V(G) \to \mathbb{R}^2$ and a vertex $v \in V(G)$, we denote by $(f(v))_i$ the i-th component of $f(v)$. This is also abbreviated by $f(v)_i$, $i \in [2]$. We start by listing some basic properties for unit square graphs.

▶ **Observation 3.2.** *Let G be a unit square graph and $f\colon V(G) \to \mathbb{R}^2$ a realization. Further let $X \subseteq V(G)$, such that there are $a_1, b_1, a_2, b_2 \in \mathbb{R}$ with $a_1 \leq b_1 \leq a_1 + 1$, $a_2 \leq b_2$ and $f(v) \in [a_1, b_1] \times [a_2, b_2]$ for every $v \in X$. Then $G[X]$ is a unit interval graph.*

▶ **Lemma 3.3.** *Let G be a unit square graph. Then the following properties hold:*

1. *There is some $v \in V(G)$, such that $G[N[v]]$ is a unit interval graph.*

2. *For every two non-adjacent $u, v \in V(G)$ the set $N(u) \cap N(v)$ induces a unit interval graph with at most two independent vertices.*
3. *G has no induced subgraph isomorphic to $K_{1,5}$, $K_{2,3}$ or $\overline{3K_2}$ ($3K_2$ is the disjoint union of three K_2).*

Proof. Let $f\colon V(G) \to \mathbb{R}^2$ be a realization. Pick $v = \operatorname{argmin}_{v \in V(G)} f(v)_1$. Further let $a_1 = f(v)_1$, $b_1 = a_1 + 1$, $a_2 = f(v)_2 - 1$ and $b_2 = f(v)_2 + 1$. Then $f(w) \in [a_1, b_1] \times [a_2, b_2]$ for every $w \in N[v]$. So $G[N[v]]$ is a unit interval graph by Observation 3.2.

Now let $u, v \in V(G)$ be two non-adjacent vertices. Without loss of generality assume $f(u)_1 + 1 \leq f(v)_1$. For $w \in N(u) \cap N(v)$ we obtain $f(u)_1 \leq f(w)_1 \leq f(v)_1$. Without loss of generality let $f(u) = (0, 0)$. Then $f(w) \in [0, 1] \times [-1, 1]$ for every $w \in N(u) \cap N(v)$. So $N(u) \cap N(v)$ defines a unit interval graph according to Observation 3.2. Furthermore in this area there can be at most two independent vertices.

For the third item first observe that the class of unit square graphs is hereditary (i.e. it is closed under taking induced subgraphs) so it suffices to show that the listed graphs are not unit square graphs. We first consider the graph $K_{1,5}$. Suppose towards a contradiction that there is a realization $f : V(G) \to \mathbb{R}^2$. Without loss of generality let $f(v) = (0, 0)$ where v is the center vertex connected to the other vertices $w_1, \ldots, w_5$. Then there is some quadrant containing two vertices w_i and w_j for distinct $i, j \in [5]$. But then $w_i w_j \in E(G)$ which is a contradiction.

For $K_{2,3}$ the two vertices on the left side have three independent common neighbors. The graph $\overline{3K_2}$ contains two non-adjacent vertices whose common neighborhood is a 4-cycle. So in both cases it follows from the second statement that the graph is not a unit square graph. ◀

We also require some properties of maximal cliques. A clique is a set $C \subseteq V(G)$, such that $vw \in E(G)$ for every two distinct $v, w \in C$. A maximal clique is a clique so that there is no larger clique containing it. For a graph G the set of maximal cliques of G is denoted by $\mathcal{M}(G)$.

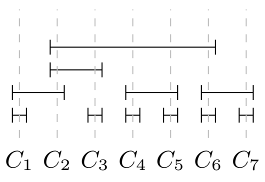
▶ **Lemma 3.4.** *Let G be a unit square graph and C be a maximal clique of G. Then there are $v_1, \ldots, v_4 \in V(G)$, such that $C = \bigcap_{i \in [4]} N[v_i]$.*

Proof. Let $f : V(G) \to \mathbb{R}^2$ be a realization of G. Let $v_{2i-1} = \operatorname{argmin}_{v \in C} f(v)_i$ and $v_{2i} = \operatorname{argmax}_{v \in C} f(v)_i$ for $i \in [2]$. Clearly $C \subseteq \bigcap_{i \in [4]} N[v_i]$. So let $w \in \bigcap_{i \in [4]} N[v_i]$ and $v \in C$. In order to prove $w \in C$ it suffices to show that $\|f(v) - f(w)\|_\infty \leq 1$, since v is chosen arbitrarily from C. For $i \in [2]$ it holds that $f(v_{2i-1})_i \leq f(v_{2i})_i$. If $f(w)_i \leq f(v_{2i})_i$ then $-1 \leq f(v_{2i})_i - 1 - f(v)_i \leq f(w)_i - f(v)_i \leq f(v_{2i})_i - f(v)_i \leq 1$. Otherwise $f(w)_i \geq f(v_{2i-1})_i$ and $-1 \leq f(v_{2i-1})_i - f(v)_i \leq f(w)_i - f(v)_i \leq f(v_{2i-1})_i + 1 - f(v)_i \leq 1$. ◀

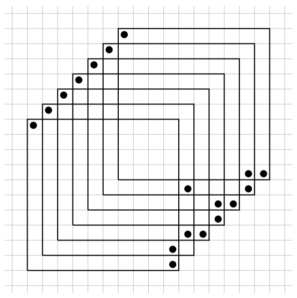
In particular all maximal cliques can be computed in polynomial time.

4 Local structure

The basic approach for our algorithm is group-theoretic. A main obstacle for group theoretic approaches comes from large symmetric or alternating groups that appear in the automorphism group of the given graph. For unit square graphs the central observation is that these groups can in a way only arise from cliques. In this section we show how to cope with possibly very symmetric parts of the graphs and give a corresponding reduction to get rid of them.



(a) An interval graph G



(b) The constructed realization for $G_{\mathcal{M}}$

Figure 2 From interval to unit square graphs.

We start by giving a central class of examples to obtain a better understanding of how the symmetric parts may look like. Let G be a graph. Define the colored graph $G_{\mathcal{M}} = (V(G) \cup \mathcal{M}(G), E(G_{\mathcal{M}}), c_{G_{\mathcal{M}}})$ with

$$E(G_{\mathcal{M}}) = \{vC \mid C \in \mathcal{M}(G), v \in C\} \cup \{vw \mid v \neq w \in V(G)\} \cup \{CD \mid C \neq D \in \mathcal{M}(G)\}$$

and $c_{G_{\mathcal{M}}}(v) = c_G(v) + 1$ for $v \in V(G)$, $c_{G_{\mathcal{M}}}(C) = 1$ for $C \in \mathcal{M}(G)$. For a group $\Gamma \leq \mathrm{Sym}(\Omega)$ and a set $A \subseteq \Omega$ the *pointwise stabilizer* is the group $\mathrm{Stab}^{\bullet}_{\Gamma}(A) := \{\gamma \in \Gamma \mid \forall \alpha \in A \colon \alpha^{\gamma} = \alpha\}$. If A is invariant under Γ (i.e. $A^{\gamma} = A$ for every $\gamma \in \Gamma$) we define the restriction of Γ to A as $\Gamma|_A := \{\gamma|_A \mid \gamma \in \Gamma\}$ where $\gamma|_A \colon A \to A$ with $\gamma|_A(\alpha) = \gamma(\alpha)$.

▶ **Observation 4.1.** *For every two graphs G and H it holds that*
1. *$G \cong H$ if and only if $G_{\mathcal{M}} \cong H_{\mathcal{M}}$,*
2. *$\mathrm{Stab}^{\bullet}_{\mathrm{Aut}(G_{\mathcal{M}})}(V(G)) = \{1\}$ (here 1 denotes the identify mapping),*
3. *$\mathrm{Aut}(G_{\mathcal{M}})|_{V(G)} = \mathrm{Aut}(G)$.*

In particular $\mathrm{Aut}(G) \cong \mathrm{Aut}(G_{\mathcal{M}})$ by combining the second and third part of the observation. We now show that for each interval graph G the graph $G_{\mathcal{M}}$ is a unit square graph. For this we use the following characterization of interval graphs: A graph G is an interval graph if and only if there is a linear order on the maximal cliques, such that each vertex appears in consecutive maximal cliques [14]. For a vertex $v \in V(G)$ let $\mathcal{M}_G(v) = \mathcal{M}(v) = \{C \in \mathcal{M}(G) \mid v \in C\}$.

▶ **Lemma 4.2.** *Let G be an interval graph. Then $G_{\mathcal{M}}$ is a colored unit square graph.*

Proof. Let $<$ be a linear order on $\mathcal{M}(G)$, such that each vertex appears in consecutive maximal cliques. Let $k = |\mathcal{M}(G)|$ and $C_1 < C_2 < \cdots < C_k$ be the maximal cliques of G. For each $v \in V(G)$ define $a_v, b_v \in [k]$ in such a way that $\mathcal{M}(v) = \{C_i \mid a_v \leq i \leq b_v\}$. Consider the following realization $f : V(G_{\mathcal{M}}) \to \mathbb{R}^2$ with $f(C_i) = (\frac{i}{k} - 1, \frac{i}{k})$ and $f(v) = (\frac{a_v}{k}, \frac{b_v}{k} - 1)$ for all $v \in V(G)$. Clearly all vertices are connected to each other as well as all maximal cliques. So let $v \in V(G)$. Then $|\frac{a_v}{k} - \frac{i}{k} + 1| = |\frac{a_v - i}{k} + 1| \leq 1$ if and only if $a_v \leq i$. Further $|\frac{i}{k} - \frac{b_v}{k} + 1| = |\frac{i - b_v}{k} + 1| \leq 1$ if and only if $i \leq b_v$. So there is an edge between $v \in V(G)$ and $C_i \in \mathcal{M}$ if and only if $a_v \leq i \leq b_v$ if and only if $v \in C_i$. ◀

A visualization of the presented realization is also given in Figure 2. In Figure 2b the vertices of $G_{\mathcal{M}}$ are represented by points. The squares are only for visualization purposes and indicate which vertices are connected. Each square describes a maximal clique of the original interval graph and contains exactly the vertex which corresponds to the given clique and the vertices being in the clique.

▶ **Corollary 4.3.** *For each colored interval graph G one can compute in polynomial time some colored unit square graph H with* $\mathrm{Aut}(G) \cong \mathrm{Aut}(H)$.

In particular this construction implies that there are twin-free unit square graphs where the automorphism group contains arbitrarily large symmetric groups which can not be handled by the group theoretic approach due to Luks. For example consider the following graph $G_{n,k}$ for $n, k \in \mathbb{N}$. The vertex set $V(G_{n,k}) = [n]^{\leq k}$ is the set of all words over the alphabet $[n]$ of length at most k and there is an edge $vw \in E(G_{n,k})$ if v is a prefix of w (this is interpreted for an undirected graph). First, $G_{n,k}$ is an interval graph. To verify this consider the set $[n]^k$ of words of length k with the natural lexicographic order. Then each vertex $v \in V(G_{n,k})$ can be represented by the interval $I_{n,k}(v) = \{w \in [n]^k \mid v \text{ is prefix of } w\}$. It is easy to check that two vertices are connected if and only if the corresponding intervals intersect. The automorphism group of $G_{n,k}$ is a wreath product of the automorphism group of $G_{n,k-1}$ by a symmetric group on n points.

One of the main contributions of this work is to show that within automorphism groups of unit square graphs large symmetric groups only appear in a local setting. Here, local refers to a small area in the realization of a unit square graph G. In the presented example the vertices of each color class are close together and in particular they induce a clique. The main target for this section is to present a method how to cope with the local parts of the graph that may admit large symmetric groups in the automorphism group. For this purpose we have to analyze the structure of clique-partitions of the vertices.

▶ **Definition 4.4.** Let G be a graph and $\mathcal{P}$ be a partition of the vertices. We call $\mathcal{P}$ a *clique-partition* if X is a clique for each $X \in \mathcal{P}$.

Let G be a unit square graph with realization $f : V(G) \to \mathbb{R}^2$. We first define the graph $G^*_{\mathcal{M}} = (V(G) \cup \mathcal{M}(G), E(G^*_{\mathcal{M}}), c_{G^*_{\mathcal{M}}})$ with $E(G^*_{\mathcal{M}}) = \{vC \mid C \in \mathcal{M}(G), v \in C\} \cup E(G)$ and $c_{G^*_{\mathcal{M}}} = c_{G_{\mathcal{M}}}$ as defined above. Let $\mathcal{P}$ be a clique-partition of $V(G)$, which is refined by the color refinement algorithm applied to the graph $G^*_{\mathcal{M}}$. More precisely let $\mathcal{P}^*$ be the unique coarsest partition of $V(G) \cup \mathcal{M}(G)$ that is stable with respect to $G^*_{\mathcal{M}}$ and refines the partition $\mathcal{P} \cup \{\mathcal{M}(G)\}$. The partition $\mathcal{P}$ is called *clique-stable* if $\mathcal{P}^* \cap 2^{V(G)} = \mathcal{P}$.

▶ **Lemma 4.5.** *Let G be a twin-free unit square graph and let $\mathcal{P}$ be a clique-stable partition. Further let $f : V(G) \to \mathbb{R}^2$ be a realization. Then the following properties hold:*

1. *For each $X \in \mathcal{P}$ there exists $b \in \{-1, 1\}$, such that*
$$f(v)_1 \leq f(w)_1 \;\Leftrightarrow\; b \cdot f(v)_2 \leq b \cdot f(w)_2 \tag{1}$$
for all $v, w \in X$. The value b is called the orientation of X, *which is denoted by* $\mathrm{ori}_{G,f}(X)$ *(the value b is unique unless $|X| = 1$, in this case we define* $\mathrm{ori}_{G,f}(X) = 1$*).*
2. *Let $X, Y \in \mathcal{P}$ with* $\mathrm{ori}_{G,f}(X) \neq \mathrm{ori}_{G,f}(Y)$. *Then either $xy \in E(G)$ for all $x \in X, y \in Y$ or there is no $x \in X, y \in Y$ with $xy \in E(G)$.*
3. *Let $X, Y \in \mathcal{P}$ with* $\mathrm{ori}_{G,f}(X) = \mathrm{ori}_{G,f}(Y)$ *and suppose $k = |\{xy \in X \times Y \mid xy \in E(G)\}| \geq 1$. Further let $X = \{x_1, \dots, x_s\}$, such that $f(x_i)_1 \leq f(x_{i+1})_1$ for all $i \in [s]$, and $Y = \{y_1, \dots, y_t\}$, such that $f(y_j)_1 \leq f(y_{j+1})_1$ for all $j \in [t]$. Then*
$$x_i y_j \in E(G) \;\Leftrightarrow\; \left\lceil \frac{i \cdot t}{k} \right\rceil = \left\lceil \frac{j \cdot s}{k} \right\rceil. \tag{2}$$

Now let $\mathcal{P}$ be a canonical, clique-stable partition of the graph G. We define the vertex and edge-colored graph $G[\mathcal{P}] = (\mathcal{P}, E, c_V, c_E)$ with $E = \{XY \mid X \neq Y \in \mathcal{P}\}$,

$c_V : \mathcal{P} \to \mathbb{N} \colon X \mapsto |X|$

and

$$c_E \colon E \to \mathbb{N} \colon XY \mapsto |\{xy \in X \times Y \mid xy \in E(G)\}|.$$

For $\gamma \in \mathrm{Aut}(G)$ define the permutation $\gamma^{\mathcal{P}} := \varphi(\gamma) \in \mathrm{Sym}(\mathcal{P})$ where $\varphi\colon \mathrm{Aut}(G) \to \mathrm{Sym}(\mathcal{P})$ is the natural action of $\mathrm{Aut}(G)$ on $\mathcal{P}$. Note that φ is well-defined since the partition $\mathcal{P}$ is canonical.

▶ **Theorem 4.6.** *Let G be a twin-free unit square graph and let $\mathcal{P}$ be a canonical, clique-stable partition. Further let $\delta \in \mathrm{Aut}(G[\mathcal{P}])$. Then there is some $\gamma \in \mathrm{Aut}(G)$ with $\gamma^{\mathcal{P}} = \delta$.*

Intuitively, the last theorem states that each automorphism of $G[\mathcal{P}]$ naturally extends to an automorphism of G. In particular, the graph G can be uniquely reconstructed from the graph $G[\mathcal{P}]$. This is the main result on the local structure of unit square graphs which allows us, for a canonical, clique-stable partition $\mathcal{P}$, to restrict to the graph $G[\mathcal{P}]$. For the remainder of this work the goal is to compute a canonical, clique-stable partition $\mathcal{P}$, such that the automorphism group of $G[\mathcal{P}]$ is a Γ_t-group for some constant t. To achieve this goal we require the graph G to have some singleton vertex v_0 (a vertex with a unique color). More precisely, for such a graph we construct the desired partition $\mathcal{P}$ and a canonical, t-circle-bounded graph H, such that $\mathcal{P} \subseteq V(H)$ and $\mathcal{P}$ is invariant under $\mathrm{Aut}(H)$. This results in a good supergroup of the group $\mathrm{Aut}(G[\mathcal{P}])$ which can be used by Luks' algorithm to compute the real automorphism group. To compute the graph H we devise an algorithm that iteratively extends H taking vertices with larger and larger distances to v_0 into account. While doing so the crucial subproblem is to compute canonical clique-partitions for neighborhoods of cliques. This problem is addressed in the next section.

5 Neighborhoods

In order to obtain canonical clique-partitions for neighborhoods we essentially proceed in two steps. First, we use some combinatorial partitioning techniques to obtain some initial coloring of the vertices. Then, considering each color class separately, the main contribution is to prove that each color class can either be described by a co-bipartite graph or a proper circular arc graph. In both cases it is easy to compute a canonical clique-partition.

5.1 Neighborhood graphs

Before considering neighborhoods of cliques we first restrict to neighborhoods of vertices. This occurs as a subcase when analyzing neighborhood of cliques. Also the structure of neighborhoods tends to be simpler than for neighborhoods of cliques.

▶ **Definition 5.1.** A unit square graph is a *neighborhood graph* if there is a realization $f\colon V(G) \to [-1,1]^2$.

Note that every graph induced on a neighborhood of a vertex is indeed a neighborhood graph and every neighborhood graph can be turned into the neighborhood of a vertex by adding a universal vertex located at the origin. Let G be a neighborhood unit square graph. The goal is to prove that, after performing the k-dimensional Weisfeiler-Leman algorithm for sufficiently large k, each color class of vertices is co-bipartite or proper circular arc. We build on the characterization of proper circular arc graphs in terms of forbidden induced subgraphs. We start by giving two general graph-theoretic lemmas.

▶ **Lemma 5.2.** *Let G be a graph, such that*
1. *$G[N[v]]$ is a unit interval graph for each $v \in V(G)$,*
2. *G has no induced subgraph isomorphic to $C_4 \cup K_1$.*
Further let $X = \{w_1 \in V(G) \mid \exists w_2, \dots, w_6 \colon G[w_1, \dots, w_6] \cong \overline{C_6}\}$. Then $G[X]$ is co-bipartite.

▶ **Lemma 5.3.** *Let G be a graph, such that*
1. *$G[N[v]]$ is a unit interval graph for each $v \in V(G)$,*
2. *G has no induced subgraph isomorphic to $C_{n+4} \cup K_1$ for $n \geq 0$,*
3. *there are no $v, w \in V(G)$, such that $N[v] \subsetneq N[w]$.*
Then G has no induced subgraph isomorphic to `net`*.*

Now let G be a neighborhood unit square graph. In order to apply Lemma 2.7 we still need to consider $C_{n+4} \cup K_1$ and $S_3 \cup K_1$.

▶ **Lemma 5.4.** *Let G be a neighborhood unit square graph. Let $X = \{v \in V(G) \mid \exists \ell \geq 4\, \exists w_1, \dots, w_\ell : vw_i \notin E(G) \wedge G[w_1, \dots, w_\ell] \cong C_\ell\}$. Then $X \neq V(G)$.*

Proof. Let $f \colon V(G) \to [-1,1]^2$ be a realization and let $v = \text{argmin}_{v \in V(G)} |f(v)_1|$. Suppose towards a contradiction that $v \in X$. Then there is some $\ell \geq 4$ and $w_1, \dots, w_\ell \in V(G)$, such that $vw_i \notin E(G)$ for all $i \in [\ell]$ and $w_iw_j \in E(G)$ if and only if $i - j \equiv \pm 1 \mod \ell$ for all $i, j \in [\ell]$. Without loss of generality assume that $f(v) \in [-1,0] \times [-1,0]$. Let $i = \text{argmin}_{i \in [\ell]} f(w_i)_2$. Since $G[w_1, \dots, w_\ell]$ is not a unit interval graph it holds that $f(w_i) \in [0,1] \times [-1,0]$ by Observation 3.2. Without loss of generality assume $i = 2$. Now consider the two neighbors w_1 and w_3. Note that $w_1w_3 \notin E(G)$ since $\ell \geq 4$. Then there is some $j \in \{1,3\}$, such that $f(w_j) \in [-1,0) \times [0,1]$. So in particular $f(w_j)_1 < 0$. Further $f(w_j)_1 + 1 \geq f(w_2)_1$ and $f(v)_1 + 1 < f(w_2)_1$. Altogether this means that $f(v)_1 < f(w_j)_1 < 0$ contradicting the definition of v. ◀

▶ **Lemma 5.5.** *Let G be a neighborhood unit square graph. Let $X = \{v \in V(G) \mid \exists w_1, \dots, w_6 : vw_i \notin E(G) \wedge G[w_1, \dots, w_6] \cong S_3\}$. Then $X \neq V(G)$.*

This lemma is proved in a similar fashion to Lemma 5.4. In order to prove the main partitioning result for neighborhood graphs we also require that for sufficiently large k the k-dimensional Weisfeiler-Leman algorithm identifies all interval graphs (cf. [21]).

▶ **Corollary 5.6.** *There is some $k \in \mathbb{N}$, such that for each neighborhood unit square graph the following holds: After performing k-dimensional Weisfeiler-Leman each equivalence class of vertices induces a graph which is co-bipartite or proper circular arc with at most four connected components.*

Proof. Choose k sufficiently large and let $X \subseteq V(G)$ be an equivalence class. Then $G[X]$ is a neighborhood unit square graph. By Lemma 3.3 there is some $v \in X$, such that $N_{G[X]}[v]$ induces a unit interval graph. Since k-dimensional Weisfeiler-Leman identifies all interval graphs this is true for all $v \in X$. From Lemma 5.4 it follows that there exists a vertex $v \in X$, such that every induced cycle contains at least one vertex being a neighbor of v. Again by stability of the set X with respect to k-dimensional Weisfeiler-Leman this is true for all $v \in X$ (note that the maximal length of an induced cycle is at most 8). So there is no induced subgraph isomorphic to $C_{n+4} \cup K_1$. Combining the same argument with Lemma 5.5 we get that $G[X]$ also has no induced subgraph isomorphic to $S_3 \cup K_1$. Since $G[X]$ is regular there are no vertices $v, w \in X$, such that $N_{G[X]}[v] \subsetneq N_{G[X]}[w]$. So we can apply Lemma 5.3 and obtain that there is no induced subgraph isomorphic to `net`. Furthermore $G[X]$ has no induced subgraph isomorphic to $\overline{3K_2}$ by Lemma 3.3 and therefore it has also no induced subgraph $\overline{T_2}$. Now suppose there is an induced subgraph isomorphic to $\overline{C_6}$. Then, by stability, every vertex is part of an induced subgraph $\overline{C_6}$ and thus, $G[X]$ is co-bipartite by Lemma 5.2. Otherwise $G[X]$ is proper circular arc by Lemma 2.7. The bound on the number of components follows from the fact that $K_{1,5}$ is not a unit square graph (cf. Lemma 3.3). ◀

▶ **Theorem 5.7.** *Let G be a neighborhood unit square graph. Then one can compute in polynomial time a canonical clique-partition $\mathcal{P}$ and a canonical colored graph H, such that*
1. $\mathcal{P} = V(H)$,
2. *H is 4-circle-bounded,*
3. $\operatorname{im}(\varphi) \le \operatorname{Aut}(H)$ *where* $\varphi\colon \operatorname{Aut}(G) \to \operatorname{Sym}(\mathcal{P})$ *is the natural action of* $\operatorname{Aut}(G)$ *on* $\mathcal{P}$.

Proof. Choose k according to Corollary 5.6 and let $X \subseteq V(G)$ be an equivalence class after performing k-dimensional Weisfeiler-Leman. Further let c be the color of the equivalence class. First suppose $G[X]$ is co-bipartite. Let t be the number of non-trivial connected components of $\overline{G[X]}$. Then $t \le 2$ by Lemma 3.3. Let $Y_{j,1}, Y_{j,2}$ be the unique bipartition of the j-th connected component, $j \in [t]$. Further let Y be the set of isolated vertices in $\overline{G[X]}$ and $\mathcal{Y} = \{Y\}$ if $Y \neq \emptyset$ and $\mathcal{Y} = \emptyset$ otherwise. Define $\mathcal{P}_X = \{Y_{j,j'} \mid j \in [t], j' \in [2]\} \cup \mathcal{Y}$. Further let $H_X = \{\mathcal{P}_X, E(H_X), c_X\}$ with $YZ \in E(H_X)$ if there are $v \in Y, w \in Z$ with $vw \in E\left(\overline{G[X]}\right)$ and $c_X(Y) = c$.

Otherwise $G[X]$ is proper circular arc according to Corollary 5.6. Let $X_1, \ldots, X_t$ be the connected components of $G[X]$. Then $t \le 4$ by Corollary 5.6. Let $i \in [t]$ and let $\mathcal{P}_{X,i}$ be the partition containing the equivalence classes of the connected twins relation for $G[X_i]$. Further let $H_{X,i}$ be the graph computed by Theorem 2.9 where each vertex is colored by c. Define $\mathcal{P}_X = \bigcup_{i\in[t]} \mathcal{P}_{X,i}$ and $H_X = \bigcup_{i\in[t]} H_{X,i}$.

Finally let $\mathcal{P} = \bigcup_X \mathcal{P}_X$ and $H = \bigcup_X H_X$. It can easily be checked that $\mathcal{P}$ and H have the desired properties. ◀

5.2 Clique neighborhoods graphs

Remember, that our goal is to compute a canonical clique-partition of a given unit square graph with singleton vertex v_0. We first group the vertices according to their distance to v_0. Then, for the first level of vertices which are all the neighbors of v_0, we use the previous theorem to compute a canonical clique-partition. For all other levels we want to build up on the partition computed in the previous level. More precisely, for a given clique in the partition of the previous level we want to partition its neighbors in the current level. Hence, we need to consider neighborhoods of cliques and extend the results of the previous subsection accordingly.

Let G be a colored unit square graph and let $X \subseteq V(G)$ be a clique, such that $V(G) = N[X] = \bigcup_{v\in X} N[v]$. Further suppose there is some color i, such that $X = V_i(G)$, and there is some $k \in [|X|]$, such that $|N[v] \cap X| = k$ for all $v \in V(G) \setminus X$. In this case G is called a *simple clique neighborhood graph with respect to X*. The next theorem extends the result of the previous subsection to simple clique neighborhood graphs. Note that we have to pay a price here, namely the constant for the circle-bounded graph increases from four to eight. This can be explained by the fact that a single vertex can have at most four independent neighbors whereas a clique can have eight independent neighbors (cf. Figure 3).

▶ **Theorem 5.8.** *Let G be a simple clique neighborhood graph with respect to $X \subseteq V(G)$. Then one can compute in polynomial time a canonical clique-partition $\mathcal{P}$ and a canonical colored graph H, such that*

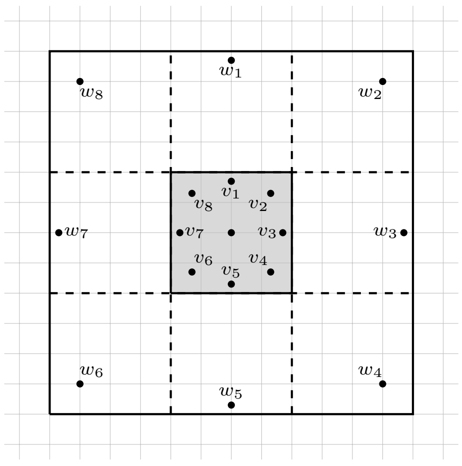


Figure 3 Neighborhood of a clique and realization for the graph G_8

1. $\mathcal{P} \subseteq V(H)$ *and* $\mathcal{P}$ *is* $\mathrm{Aut}(H)$*-invariant,*
2. H *is* 8*-circle-bounded,*
3. $\mathrm{im}(\varphi) \leq \mathrm{Aut}(H)|_{\mathcal{P}}$ *where* $\varphi\colon \mathrm{Aut}(G) \to \mathrm{Sym}(\mathcal{P})$ *is the natural action of* $\mathrm{Aut}(G)$ *on* $\mathcal{P}$*.*

The basic idea for the proof is similar to Theorem 5.7 but the technical details are far more involved. In particular Corollary 5.6 does not hold for neighborhoods of cliques. To circumvent this problem the basic idea is to consider an initial partition which is based on whether two vertices have the same neighbors in X. Then the single sets all define neighborhood graphs whereas on the sets considered as single elements we can define an auxiliary graph in a canonical way so that this auxiliary graph is again proper circular arc. From this point we can use similar arguments as for neighborhoods of single vertices. We omit the details here.

6 Global structure

In this section we are ready construct a canonical, clique-stable partition $\mathcal{P}$ together with some canonical 8-circle-bounded graph H, such that $\mathcal{P} \subseteq V(H)$ and $\mathcal{P}$ is $\mathrm{Aut}(H)$-invariant. This method is the central part of our algorithm and gives us a good supergroup of the natural action of the automorphism group on the computed partition. The computed supergroup is then given to the subroutine, that computes setwise stabilizers for groups in Γ_8, to obtain the automorphism group of $G[\mathcal{P}]$.

▶ **Theorem 6.1.** *Let G be a connected unit square graph with singleton vertex. Then one can compute in polynomial time a canonical, clique-stable partition $\mathcal{P}$ and a canonical colored graph H, such that*
1. $\mathcal{P} \subseteq V(H)$ *and* $\mathcal{P}$ *is invariant under* $\mathrm{Aut}(H)$*,*
2. H *is* 8*-circle-bounded,*
3. $\mathrm{im}(\varphi) \leq \mathrm{Aut}(H)|_{\mathcal{P}}$ *where* $\varphi\colon \mathrm{Aut}(G) \to \mathrm{Sym}(\mathcal{P})$ *is the natural action of* $\mathrm{Aut}(G)$ *on* $\mathcal{P}$*.*

The basic idea for the algorithm is to proceed in two steps. First, we compute a clique-partition $\mathcal{P}$, which is only canonical but not necessarily clique-stable, together with a corresponding graph H. For this part of the algorithm we make use of the partitioning algorithm for neighborhoods of cliques. More precisely we first group the vertices according to their distance to the singleton vertex v_0 and then we iteratively consider vertices with larger and larger distances to v_0. In the first iteration we only consider the neighbors of v_0 and compute a clique-partition and a canonical graph using Theorem 5.7. In the i-th iteration we partition the vertices with distance i to v_0 based on the partition of the vertices in the previous level. For each clique in the partition of the previous level we partition its neighbors in the current level using Theorem 5.8. Then we combine the computed partitions into one partition for the current level and use the computed graphs (which we obtained from Theorem 5.8) to update the graph H.

Then, in a second step, we refine the computed partition using the color refinement algorithm while simultaneously extending the graph H. The crucial idea for extending the graph H is to use additional layers which model the iterations of the color refinement algorithm.

▶ **Corollary 6.2.** *Let G be a connected unit square graph with singleton vertex. Then one can compute in polynomial time a canonical clique-partition $\mathcal{P}$, such that $\operatorname{im}(\varphi) \in \Gamma_8$ where $\varphi\colon \operatorname{Aut}(G) \to \operatorname{Sym}(\mathcal{P})$ is the natural action of $\operatorname{Aut}(G)$ on $\mathcal{P}$.*

▶ Remark. The constant $d = 8$ is tight for the previous corollary. In particular the graph G_8 with $V(G_8) = \{v_i \mid i \in [9]\} \cup \{w_i \mid i \in [8]\}$ and $E(G_8) = \{v_i v_j \mid i \neq j \in [9]\} \cup \{v_i w_i \mid i \in [8]\}$ is a unit square graph (the vertex v_9 may be a singleton vertex). A possible realization of G_8 is depicted in Figure 3.

Together with Theorem 4.6 this gives us sufficient structure to compute the natural action of the automorphism group on the computed partition. This can also be used to solve the isomorphism problem.

▶ **Theorem 6.3.** *Let G be a connected, twin-free unit square graph with a singleton vertex. Then one can compute in polynomial time a canonical, clique-stable partition $\mathcal{P}$ and a set $S \subseteq \operatorname{Sym}(\mathcal{P})$, such that $\langle S \rangle = \operatorname{im}(\varphi) \in \Gamma_8$ where $\varphi\colon \operatorname{Aut}(G) \to \operatorname{Sym}(\mathcal{P})$ is the natural action of $\operatorname{Aut}(G)$ on $\mathcal{P}$.*

Proof. Let $\mathcal{P}$ be the canonical, clique-stable partition and H the canonical, 8-circle-bounded graph obtained from Theorem 6.1. Then $\operatorname{Aut}(H)$ can be computed in polynomial time and $\operatorname{Aut}(H) \in \Gamma_8$ by Theorem 2.5 and 2.4. Further $\mathcal{P}$ is invariant under $\operatorname{Aut}(H)$. Since H is canonical this implies $\operatorname{im}(\varphi) \leq \operatorname{Aut}(H)|_{\mathcal{P}} \in \Gamma_8$. Furthermore $\operatorname{im}(\varphi) = \operatorname{Aut}(G[\mathcal{P}])$ by Theorem 4.6. A generating set for $\operatorname{Aut}(G[\mathcal{P}])$ can be computed in polynomial time using Theorem 2.2. ◀

▶ **Theorem 6.4.** *The Graph Isomorphism Problem for unit square graphs can be solved in polynomial time.*

Proof. Let G_1, G_2 be two unit square graphs. First, it can be assumed that G_1 and G_2 are connected by considering the connected components separately. Furthermore, the graphs can be assumed to be twin-free using modular decompositions of graphs (cf. [26]). Let $c \in \mathbb{N}$ be a fresh color (i.e. a color which does not appear in G_1 or G_2). For a graph G and a vertex $v \in V(G)$ we denote by $G^{v \mapsto c}$ the graph where vertex v is colored by c. Pick $v_1 \in V(G_1)$. For each $v_2 \in V(G_2)$ test whether $G_1^{v_1 \mapsto c} \cong G_2^{v_2 \mapsto c}$ by the following procedure. For $i \in [2]$ let $\mathcal{P}_i$ be the partition and H_i be the graph computed by Theorem 6.1 for the graph $G_i^{v_i \mapsto c}$. Let H be the disjoint union of H_1 and H_2. Note that $H_1 \cong H_2$ if $G_1^{v_1 \mapsto c} \cong G_2^{v_2 \mapsto c}$ because the graph H_i is canonical. Compute a generating set for $\operatorname{Aut}(H) \in \Gamma_8$. This can be done in polynomial time according to Theorem 2.5. Let G be the disjoint union of $G_1^{v_1 \mapsto c}[\mathcal{P}_1]$ and $G_2^{v_2 \mapsto c}[\mathcal{P}_2]$. Then $\operatorname{Aut}(G) \leq \operatorname{Aut}(H)|_{\mathcal{P}_1 \cup \mathcal{P}_2}$ and hence a generating set for $\operatorname{Aut}(G)$ can be computed in polynomial time using Theorem 2.2 (note that $\operatorname{Aut}(G)$ is the set of permutations which stabilize the edge set). By Theorem 4.6 it holds that $G_1^{v_1 \mapsto c} \cong G_2^{v_2 \mapsto c}$ if and only if there is an automorphism $\gamma \in \mathrm{Aut}(G)$ that maps $G_1^{v_1 \mapsto c}[\mathcal{P}_1]$ to $G_2^{v_2 \mapsto c}[\mathcal{P}_2]$. Since G is the disjoint union of of $G_1^{v_1 \mapsto c}[\mathcal{P}_1]$ and $G_2^{v_2 \mapsto c}[\mathcal{P}_2]$ and both of these graphs are connected it holds that if such an automorphism exists then there will also be one present in the generating set of $\mathrm{Aut}(G)$. Thus it can be checked in polynomial time whether $G_1^{v_1 \mapsto c} \cong G_2^{v_2 \mapsto c}$. ◀

▶ Remark. The running time of the presented algorithm is dominated by the running time for the subroutine computing setwise stabilizers for groups in Γ_8, which in turn depends on the maximal size of primitive Γ_8-groups.

The latter was analyzed by Babai, Cameron and Pálfy in [3] and proven to be polynomially bounded in the size of the permutation domain. For a complexity analysis of the setwise stabilizer subroutine we refer to [22, 23, 5]. Note that the setwise stabilizer subroutine is also used for computing the automorphism group of H and the graph H might be much larger than the original graph G.

▶ Remark. The presented algorithm also gives us some insight about the structure of the automorphism group of a unit square graph with singleton vertex. There is an invariant clique-partition, such that the natural action on the partition forms a Γ_8-group.

An interesting question is whether a similar statement still holds if the given graph does not have a singleton vertex. We leave this question open.

7 Discussion

We presented a polynomial time algorithm solving the Graph Isomorphism Problem for unit square graphs. Overall the presented algorithm heavily depends on group theoretic methods. This raises the question whether the problem can also be solved without the use of such methods. In fact, it might be that the k-dimensional Weisfeiler-Leman algorithm can identify every unit square graph for sufficiently large k. This is left as an open question.

Furthermore it is an interesting question whether the methods presented in this work can be adapted to other geometric classes for which the isomorphism problem is still open. At first glance a natural candidate seems to be the class of unit disk graphs. However, it turns out that there are some crucial structural differences to unit square graphs. In particular, there are unit disk graphs with singleton vertex, such that for each canonical clique-partition the natural action of the automorphism group contains a large symmetric group.

Finally we would like to address two natural generalizations of unit square graphs. The first one concerns the dimension of the realization, that is, what is the complexity of graph isomorphism for graphs with d-dimensional L_∞-realization for any constant number d. The second extension concerns squares of arbitrary size. This is still a natural restriction for the class of intersection graphs of rectangles, which is GI-complete, and the reduction does not directly extend to square graphs because it requires large complete bipartite graphs as induced subgraphs (cf. [29]). However, developing an efficient algorithm for this class of graphs would require some new ideas since the number of independent neighbors of a vertex is unbounded.

Acknowledgements. I want to thank Martin Grohe and Pascal Schweitzer for several helpful discussions and comments throughout this work. I also thank the anonymous referees for many comments improving the presentation of the results.

References

1 László Babai. On the automorphism groups of strongly regular graphs I. In Moni Naor, editor, *Innovations in Theoretical Computer Science, ITCS'14, Princeton, NJ, USA, January 12-14, 2014*, pages 359–368. ACM, 2014. URL: `http://dl.acm.org/citation.cfm?id=2554797`, `doi:10.1145/2554797.2554830`.

2 László Babai. Graph isomorphism in quasipolynomial time. *CoRR*, abs/1512.03547, 2015. URL: `http://arxiv.org/abs/1512.03547`.

3 László Babai, Peter J. Cameron, and Péter P. Pálfy. On the orders of primitive groups with restricted nonabelian composition factors. *Journal of Algebra*, 79(1):161–168, 1982. `doi:10.1016/0021-8693(82)90323-4`.

4 László Babai, Xi Chen, Xiaorui Sun, Shang-Hua Teng, and John Wilmes. Faster canonical forms for strongly regular graphs. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 157–166. IEEE Computer Society, 2013. `doi:10.1109/FOCS.2013.25`.

5 László Babai, William M. Kantor, and Eugene M. Luks. Computational complexity and the classification of finite simple groups. In *24th Annual Symposium on Foundations of Computer Science, Tucson, Arizona, USA, 7-9 November 1983*, pages 162–171. IEEE Computer Society, 1983. `doi:10.1109/SFCS.1983.10`.

6 Christoph Berkholz, Paul S. Bonsma, and Martin Grohe. Tight lower and upper bounds for the complexity of canonical colour refinement. In Hans L. Bodlaender and Giuseppe F. Italiano, editors, *Algorithms – ESA 2013 – 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings*, volume 8125 of *Lecture Notes in Computer Science*, pages 145–156. Springer, 2013. `doi:10.1007/978-3-642-40450-4_13`.

7 Heinz Breu. *Algorithmic Aspects of Constrained Unit Disk Graphs.* PhD thesis, University of British Columbia, Vancouver, Canada, 1996.

8 Heinz Breu and David G. Kirkpatrick. Unit disk graph recognition is np-hard. *Comput. Geom.*, 9(1-2):3–24, 1998. `doi:10.1016/S0925-7721(97)00014-X`.

9 Jin-yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identifications. *Combinatorica*, 12(4):389–410, 1992. `doi:10.1007/BF01305232`.

10 Brent N. Clark, Charles J. Colbourn, and David S. Johnson. Unit disk graphs. *Discrete Mathematics*, 86(1-3):165–177, 1990. `doi:10.1016/0012-365X(90)90358-O`.

11 Charles J. Colbourn and Kellogg S. Booth. Linear time automorphism algorithms for trees, interval graphs, and planar graphs. *SIAM J. Comput.*, 10(1):203–225, 1981. `doi:10.1137/0210015`.

12 Andrew R. Curtis, Min Chih Lin, Ross M. McConnell, Yahav Nussbaum, Francisco J. Soulignac, Jeremy P. Spinrad, and Jayme Luiz Szwarcfiter. Isomorphism of graph classes related to the circular-ones property. *Discrete Mathematics & Theoretical Computer Science*, 15(1):157–182, 2013. URL: `http://www.dmtcs.org/dmtcs-ojs/index.php/dmtcs/article/view/2298`.

13 Fedor V. Fomin, Daniel Lokshtanov, and Saket Saurabh. Bidimensionality and geometric graphs. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1563–1575. SIAM, 2012. `doi:10.1137/1.9781611973099`.

14 Delbert R. Fulkerson and Oliver A. Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15(3):835–855, 1965.

15 Martin C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs (Annals of Discrete Mathematics, Vol 57).* North-Holland Publishing Co., Amsterdam, The Netherlands, 2004.

16 Derek F. Holt, Bettina Eick, and Eamonn A. O'Brien. *Handbook of Computational Group Theory.* Discrete Mathematics and Its Applications. CRC Press, 2005.

17 Jing Huang. On the structure of local tournaments. *J. Comb. Theory, Ser. B*, 63(2):200–221, 1995. doi:10.1006/jctb.1995.1016.

18 Johannes Köbler, Sebastian Kuhnert, Bastian Laubner, and Oleg Verbitsky. Interval graphs: Canonical representations in logspace. *SIAM J. Comput.*, 40(5):1292–1315, 2011. doi:10.1137/10080395X.

19 Johannes Köbler, Sebastian Kuhnert, and Oleg Verbitsky. Solving the canonical representation and star system problems for proper circular-arc graphs in logspace. In Deepak D'Souza, Telikepalli Kavitha, and Jaikumar Radhakrishnan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012, December 15-17, 2012, Hyderabad, India*, volume 18 of *LIPIcs*, pages 387–399. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2012. URL: http://www.dagstuhl.de/dagpub/978-3-939897-47-7, doi:10.4230/LIPIcs.FSTTCS.2012.387.

20 Johannes Köbler, Sebastian Kuhnert, and Oleg Verbitsky. Helly circular-arc graph isomorphism is in logspace. In Krishnendu Chatterjee and Jirí Sgall, editors, *Mathematical Foundations of Computer Science 2013 – 38th International Symposium, MFCS 2013, Klosterneuburg, Austria, August 26-30, 2013. Proceedings*, volume 8087 of *Lecture Notes in Computer Science*, pages 631–642. Springer, 2013. doi:10.1007/978-3-642-40313-2_56.

21 Bastian Laubner. Capturing polynomial time on interval graphs. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*, pages 199–208. IEEE Computer Society, 2010. doi:10.1109/LICS.2010.42.

22 Eugene M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *J. Comput. Syst. Sci.*, 25(1):42–65, 1982. doi:10.1016/0022-0000(82)90009-5.

23 Eugene M. Luks. Permutation groups and polynomial-time computation. In Larry Finkelstein and William M. Kantor, editors, *Groups And Computation, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, October 7-10, 1991*, volume 11 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 139–176. DIMACS/AMS, 1991. URL: http://dimacs.rutgers.edu/Volumes/Vol11.html.

24 Brendan D. McKay. Practical Graph Isomorphism. *Congressus Numerantium*, 30:45–87, 1981.

25 Joseph J. Rotman. *An Introduction to the Theory of Groups*. Graduate Texts in Mathematics. Springer, 1995.

26 Pascal Schweitzer. Towards an isomorphism dichotomy for hereditary graph classes. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, volume 30 of *LIPIcs*, pages 689–702. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2015. URL: http://www.dagstuhl.de/dagpub/978-3-939897-78-1, doi:10.4230/LIPIcs.STACS.2015.689.

27 Ákos Seress. *Permutation Group Algorithms:*. Cambridge Tracts in Mathematics. Cambridge University Press, 2003.

28 Alan Tucker. Structure theorems for some circular-arc graphs. *Discrete Math.*, 7(1-2):167–195, 1974. doi:10.1016/S0012-365X(74)80027-0.

29 Ryuhei Uehara. Simple geometrical intersection graphs. In Shin-Ichi Nakano and Md. Saidur Rahman, editors, *WALCOM: Algorithms and Computation, Second International Workshop, WALCOM 2008, Dhaka, Bangladesh, February 7-8, 2008.*, volume 4921 of *Lecture Notes in Computer Science*, pages 25–33. Springer, 2008. doi:10.1007/978-3-540-77891-2_3.

30 Ryuhei Uehara. The graph isomorphism problem on geometric graphs. *Discrete Mathematics & Theoretical Computer Science*, 16(2):87–96, 2014. URL: http://www.dmtcs.org/dmtcs-ojs/index.php/dmtcs/article/view/2528.

The Alternating Stock Size Problem and the Gasoline Puzzle

Alantha Newman[*1], **Heiko Röglin**[†2], **and Johanna Seif**[3]

1 **CNRS-Université Grenoble Alpes and G-SCOP, France**
alantha.newman@grenoble-inp.fr
2 **Department of Computer Science, University of Bonn, Bonn, Germany**
roeglin@cs.uni-bonn.de
3 **ENS Lyon, Lyon, France**
johanna.seif@ens-lyon.fr

Abstract

Given a set S of integers whose sum is zero, consider the problem of finding a permutation of these integers such that: (i) all prefixes of the ordering are non-negative, and (ii) the maximum value of a prefix sum is minimized. Kellerer et al. referred to this problem as the *stock size problem* and showed that it can be approximated to within 3/2. They also showed that an approximation ratio of 2 can be achieved via several simple algorithms.

We consider a related problem, which we call the *alternating stock size problem*, where the number of positive and negative integers in the input set S are equal. The problem is the same as above, but we are additionally required to alternate the positive and negative numbers in the output ordering. This problem also has several simple 2-approximations. We show that it can be approximated to within 1.79.

Then we show that this problem is closely related to an optimization version of the gasoline puzzle due to Lovász, in which we want to minimize the size of the gas tank necessary to go around the track. We present a 2-approximation for this problem, using a natural linear programming relaxation whose feasible solutions are doubly stochastic matrices. Our novel rounding algorithm is based on a transformation that yields another doubly stochastic matrix with special properties, from which we can extract a suitable permutation.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases approximation algorithms, stock size problem, scheduling with non-renewable resources

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.71

1 Introduction

Suppose there is a set of jobs that can be processed in any order. Each job requires a specified amount of a particular resource, e.g. gasoline, which can be supplied in an amount chosen from a specified set of quantities. The limitation is that the storage space for this resource is bounded, so it must be replenished as it is used. The goal is to order the jobs and the replenishment amounts so that the required quantity of the resource is always available for the job being processed and so that the storage space is never exceeded.

[*] Supported in part by LabEx PERSYVAL-Lab (ANR-11-LABX-0025).
[†] Supported by ERC Starting Grant 306465 (BeyondWorstCase).

More formally, we are given a set of integers $Z = \{z_1, z_2, \dots z_n\}$ whose sum is zero. For a permutation σ, a prefix sum is $\sum_{i=1}^{t} z_{\sigma(i)}$ for $t \in [1, n]$. Our goal is to find a permutation of the elements in Z such that (i) each prefix sum is non-negative, and (ii) the maximum prefix sum is minimized. (Placing the elements with positive values in front of the elements with negative values satisfies (i) and therefore yields a feasible – although possibly far from optimal – solution.) This problem is known as the *stock size problem.* Kellerer, Kotov, Rendl and Woeginger presented a simple algorithm with a guarantee of $\mu_x + \mu_y$, where μ_x is the largest number in Z, and μ_y is the absolute value of the negative number with the largest absolute value in Z. (We sometimes use $\mu = \max\{\mu_x, \mu_y\}$.) Since both μ_x and μ_y are lower bounds on the value S^* of an optimal solution, this shows that the problem can be approximated to within a factor of 2. Additionally, they presented algorithms with approximation guarantees of 8/5 and 3/2 [11].

1.1 The Alternating Stock Size Problem

In this paper we first consider a restricted version of the stock size problem in which we require that the positive and negative numbers in the output permutation alternate. We refer to this problem as the *alternating stock size problem.* A motivation for this problem is that we could schedule tasks in advance of knowing the input data. For example, suppose we want to stock and remove items from a warehouse and each task occupies a certain time slot. If we want to plan ahead, we may want to designate each slot as a stocking or a removing slot in advance, e.g. all odd time slots will be used for stocking and all even time slots for de-stocking. This could be beneficial in situations where some preparation is required for each type of time slot.

The input for our new problem is two sets of positive integers, $X = \{x_1 \geq \dots \geq x_n\}$ and $Y = \{y_1 \geq \dots \geq y_n\}$, such that $|X| = |Y|$, and the two sets have equal sums. The elements of X represent the elements to be "added" and the elements of Y are those to be "removed". Note that, here, $\mu_y = y_1$ and $\mu_x = x_1$. We now formally define the new problem.

▶ **Definition 1.** The goal of the *alternating stock size problem* is to find permutations σ and ν such that

(i) for $t \in [1, n]$, $\sum_{i=1}^{t} x_{\sigma(i)} - y_{\nu(i)} \geq 0$,

(ii) $\max\limits_{1 \leq t \leq n} \sum_{i=1}^{t} (x_{\sigma(i)} - y_{\nu(i-1)})$ is minimized, where $y_{\nu(0)} = 0$.

Although this problem is a variant of the stock size problem, the algorithms found in [11] do not provide approximation guarantees since they do not necessarily produce feasible solutions for the alternating problem. Indeed, even the optimal solutions for these two problems on the same instance can differ greatly. The following example illustrates this.

$$\begin{aligned} X &= \{\underbrace{p-1, \dots, p-1}_{p \text{ entries}}, 2, \underbrace{1, \dots, 1}_{p(p-1) \text{ entries}}\}, \\ Y &= \{\underbrace{p, \dots, p}_{p-1 \text{ entries}}, \underbrace{1, 1, 1, \dots, 1}_{p(p-1)+2 \text{ entries}}\}. \end{aligned}$$

For this instance, the optimal value for the alternating problem is at least $2p - 3$, while it is p for the original stock size problem. Thus, this example exhibits a gap arbitrarily close to 2 between the optimal solutions for the two problems.

We can show the following facts about the alternating problem. (i) There is always a feasible solution. (ii) The problem is NP-hard (as is the stock size problem). (iii) It is still the case that 2μ is an upper bound on the value of an optimal solution. Our main result for this problem is to give an algorithm with an approximation guarantee of 1.79 in Section 2.

1.2 Connections to the Gasoline Puzzle

The following well-known puzzle appears on page 31 in [12]:

> Along a speed track there are some gas stations. The total amount of gasoline available in them is equal to what our car (which has a very large tank) needs for going around the track. Prove that there is a gas station such that if we start there with an empty tank, we shall be able to go around the track without running out of gasoline.

Suppose that the capacity of each gas station is represented by a positive integer and the distance of each road segment is represented by a negative integer. For simplicity, suppose that it takes one unit of gas to travel one unit of road. Then the assumption of the puzzle implies that the sum of the positive integers equals the absolute value of the sum of the negative integers. In fact, if we are allowed to permute the gas stations and the road segments (placing exactly one gas station between every pair of consecutive road segments), and our goal is to minimize the size of the gas tank required to go around the track (beginning from a feasible starting point), then this is exactly the alternating stock size problem.

This leads to the following natural problem: Suppose the road segments are fixed and we are only allowed to rearrange (i.e. permute) the gas stations. In other words, between each pair of consecutive road segments (represented by negative integers), there is a spot for exactly one gas station (represented by positive integers, the capacities), and we can choose which gas station to place in each spot. The goal is to minimize the size of the tank required to get around the track, assuming we can choose our starting gas station. What is the complexity of this problem?

We show in the full version of this paper that this problem is NP-hard [16]. Our algorithm for the alternating stock size problem specifically requires that there is flexibility in placing both the x-values and the y-values. Therefore, it does not appear to be applicable to this problem, where the y-values are pre-assigned to fixed positions. Let us now formally define the *gasoline problem*, which is the second problem we will consider in this paper.

1.3 The Gasoline Problem

As input, we are given the two sets of positive integers $X = \{x_1 \geq x_2 \geq \cdots \geq x_n\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$, where the y_i's are fixed in the given order and $\sum_{i=1}^n x_i = \sum_{i=1}^n y_i$. Our goal is to find a permutation π that minimizes the value of η:

$$\forall [k,\ell] : \quad \left| \sum_{\pi(i) \in [k,\ell]} x_i - \sum_{i \in [k,\ell-1]} y_i \right| \quad \leq \quad \eta. \tag{1}$$

Given a circle with n points labeled 1 through n, the interval $[k,\ell]$ denotes a consecutive subset of integers assigned to points k through ℓ. For example, $[5,8] = \{5,6,7,8\}$, and $[n-1,3] = \{n-1,n,1,2,3\}$. We will often use μ_x to refer to x_1, i.e. the maximum x-value, which is a lower bound on the optimal value of a solution.

Observe that in (1) we consider only intervals that contain one more x-value than y-value. One might argue that, in order to model our problem correctly, one also has to look at intervals that contain one more y-value than x-value. However, let I be such an interval and let $I' = [1,n] \setminus I$. Then the absolute value of the difference of the x-values and the y-values is the same in I and I' (with inverted signs) due to the assumption $\sum_{i=1}^n x_i = \sum_{i=1}^n y_i$.

We can also write the constraint (1) as:

$$\forall k: \sum_{\pi(i)\in[1,k]} x_i - \sum_{i\in[1,k-1]} y_i \quad \leq \quad \beta, \tag{2}$$

$$\forall k: \sum_{\pi(i)\in[1,k]} x_i - \sum_{i\in[1,k]} y_i \quad \geq \quad \alpha, \tag{3}$$

where $\alpha \leq 0$, $\beta \geq 0$ and $\eta = \beta - \alpha$. This version is slightly more general since it encompasses the scenario where we would like to minimize β for some fixed value of α. (With these constraints, it is no longer required that the sum of the x_i's equals the sum of the y_i's.)

What is the approximability of this problem? Getting a constant factor approximation appears to be a challenge since the following example shows that it is no longer the case that 2μ is an upper bound. Despite this, we show in Section 3 that there is in fact a 2-approximation algorithm for the gasoline problem.

Example showing unbounded gap between OPT and μ

Suppose X and Y each have the following n entries:

$$X = \{\underbrace{1, 1, \ldots, 1, 1, 1, \ldots, 1}_{n \text{ entries}}\}, \quad Y = \{\underbrace{2, 2, \ldots, 2}_{\frac{n}{2} \text{ entries}}, \underbrace{0, 0, \ldots, 0}_{\frac{n}{2} \text{ entries}}\}.$$

In the example above, $\mu = 2$. However, the optimal value is $n/2$.

1.4 Generalizations of the Gasoline Problem

The requirement that the x- and y-jobs alternate may seem to be somewhat artificial or restrictive. A natural generalization of the gasoline problem (which we will refer to as the *generalized gasoline problem*) is where the y-jobs are assigned to a set of predetermined positions, which are not necessarily alternating. As in the gasoline problem, our goal is to assign the x-jobs to the remaining slots so as to minimize the difference between the maximum and the minimum prefix. There is a simple reduction from this seemingly more general problem to the gasoline problem. Let $X = \{x_1 \geq x_2 \geq \cdots \geq x_{n_x}\}$ and $Y = \{y_1, y_2, \ldots, y_{n_y}\}$ be the input, where the y-jobs are assigned to n_y (arbitrary) slots. The remaining n_x slots are for the x-jobs. To reduce to an instance of the gasoline problem (with alternation), we do the following. For each set of y-jobs assigned to adjacent slots, we add them up to form a single job in a single slot. For each pair of consecutive x-slots, we place a new y-slot between them where the assigned y-job has value zero. Thus, we obtain an instance of the gasoline problem as originally defined in the beginning of this section.

Our new algorithm, developed in Section 3 to solve the gasoline problem, can also be applied to a natural generalization of the alternating stock size problem, in which we relax the required alternation between the x- and y-jobs and consider a scenario in which each slot is labeled as an x- or a y-slot and can only accomodate a job of the designated type. In other words, in the solution, the x-jobs and y-jobs will follow some specified pattern that is not necessarily alternating. The goal is to find a feasible assignment of x- and y-jobs to x- and y-slots, respectively, that minimizes the difference between the prefixes with highest and lowest values. Since this is simply a generalization of the stock size problem with the additional condition that each slot is *slated* as an x- or a y-slot, we refer to this problem as the *slated stock size problem.*

Formally, we are given two sets of positive integers $X = \{x_1 \geq x_2 \geq \cdots \geq x_{n_x}\}$ and $Y = \{y_1 \geq y_2 \geq \cdots \geq y_{n_y}\}$, and $n = n_x + n_y$ slots, each designated as either an x-slot or a

y-slot. Let I_x and I_y denote the indices of the x- and y-slots, respectively, and let P denote a prefix. Then, the objective is to find a permutation π that minimizes the value of $\beta - \alpha$, where

$$\forall P: \quad \alpha \; \leq \sum_{\pi(i) \in P \cap I_x} x_i - \sum_{\pi(i) \in P \cap I_y} y_i \; \leq \; \beta. \tag{4}$$

For this problem, we obtain an algorithm with approximation guarantee $OPT + \mu_x + \mu_y$. The details for this analysis as well as all proofs not provided in this extended abstract can be found in the full version [16].

1.5 Related Work

The work most related to the alternating stock size problem is contained in the aforementioned paper by Kellerer et al. [11]. Earlier, Abdel-Wahab and Kameda studied a variant of the stock size problem in which the output sequence of the jobs is required to obey a given set of precedence constraints, but the stock size is also allowed to be negative. They gave a polynomial-time algorithm for the case when the precedence constraints are series parallel [1]. The gasoline problem and its generalization are related to those found in a widely-studied research area known as resource constrained scheduling, where the goal is usually to minimize the completion time or to maximize the number of jobs completed in a given timeframe while subject to some limited resources [4, 6]. For example, in addition to time on a machine, a job could require a certain amount of another resource and would be eligible to be scheduled only if the *inventory* for this resource is sufficient.

A general framework for these types of problems is called scheduling with non-renewable resources. Here, *non-renewable* means not abundantly available, but rather replenished according to some rules, such as periodically and in pre-determined increments (as in the gasoline problem), or in specified increments that can be scheduled by the user (as in the alternating stock size problem), or at some arbitrary fixed timepoints. Examples for scheduling problems in this framework are described by Briskorn et al., by Györgyi and Kis, and by Morsy and Pesch [5, 8, 9, 14]. While the admissibility of a schedule is affected by the availability of a resource (e.g. whether or not there is sufficient inventory), minimizing the inventory is not a main objective in these papers.

For example, suppose we are given a set of jobs to be scheduled on a single machine. Each job consumes some resource, and is only allowed to be scheduled at a timepoint if there is sufficient resource available for that job at this timepoint. Jobs may have different resource requirements. Periodically, at timepoints and in increments known in advance, the resource will be replenished. The goal is to minimize the completion time. If at some timepoint, there is insufficient inventory for any job to be scheduled, then no job can be run, leading to gaps in the schedule and ultimately a later completion time. This problem of minimizing the completion time is polynomial time solvable (sort the jobs according to resource requirement), but an optimal schedule may contain idle times.

Suppose that we have some investment amount α that we can add to the inventory in advance to ensure that there is always sufficient inventory to schedule some job, resulting in a schedule with no empty timeslots, i.e. the optimal completion time. There is a natural connection between this scenario and the gasoline problem: Let $|\alpha|$ in Equation (3) denote the available investment. For this investment, suppose we wish to minimize β, which is the maximum inventory, in order to complete the jobs in the optimal completion time. For any feasible α and β, our algorithm in Section 3 produces a schedule with the optimal completion time using inventory size at most $\beta + \mu$.

There are other works that directly address the problem of minimizing the maximum or cumulative inventory. Monma considers a problem in which each job has a specified effect on the inventory level [13]. Neumann and Schwindt consider a scheduling problem in which the inventory is subject to both upper and lower bounds [15]. However, to the best of our knowledge, our work is the first to give approximation algorithms for the problem of minimizing the maximum inventory for non-renewable resource scheduling with fixed replenishments.

The stock size problem is also closely related to the Steinitz problem, which is a well-known problem in discrepancy theory [2]. Given a set of vectors $v_1, v_2, \dots v_n \in \mathbb{R}^d$ where $||v_i|| \leq 1$ for some fixed norm and $\sum_{i=1}^{n} v_i = 0$, the Steinitz problem is to find a permutation of the vectors so that the norm of the sum of each prefix is bounded. There exists a permutation in which the norm of each prefix is at most d [7, 3]. It has been conjectured that this bound can be improved to $O(\sqrt{d})$, but only $O(\sqrt{d}\log^{2.5} n)$ is known [10]. The stock size problem is the one-dimensional analogue of the Steinitz problem. The variants of the stock size problem that we introduce in this paper can be extended to higher dimensions.

2 Algorithms for the Alternating Stock Size Problem

The existence of a feasible solution for the alternating stock size problem follows from the solution for the gasoline puzzle. Furthermore, the upper bound of 2μ is also tight for the alternating problem. If we modify the example given in [11], we have an example for the alternating problem with an optimal stock size of $2p-3$, while $\mu = p$.

$$X = \{\underbrace{p-1, \dots, p-1}_{p \text{ entries}}, 2\}, \qquad Y = \{\underbrace{p, \dots, p}_{p-1 \text{ entries}}, 1, 1\}.$$

In this section, we will present algorithms for the alternating stock size problem. We will use the notion of a (q,T)-pair, which is a special case of a (q,T)-batch introduced and used by [11] for the stock size problem.

▶ **Definition 2.** [11] A pair of jobs $\{x, y\}$, for $x \in X$ and $y \in Y$, is called a (q,T)-pair for positive reals T and $q \leq 1$, if:
(i) $x, y \leq T$, (ii) $|x - y| \leq qT$.

The following lemma is a special case of Lemma 3 in [11], and the proofs are identical.

▶ **Lemma 3.** *For positive T, $q \leq 1$ and a set of jobs partitioned into (q,T)-pairs, we can find an alternating sequence of the jobs with maximum stock size less than $(1+q)T$.*

2.1 The Pairing Algorithm

We now consider the simple algorithm that pairs x- and y-jobs, and then applies Lemma 3 to sequence the pairs. Suppose that there is some specific pairing that matches each x_i to some y_j, and consider the difference $x_i - y_j$ for each pair. Let $\alpha_1 \geq \dots \geq \alpha_{n_1}$ denote the positive differences, and let $\beta_1 \geq \dots \geq \beta_{n_2}$ denote the absolute values of the negative differences, where $n_1 + n_2 = n$.

▶ **Lemma 4.** *The matching $M^\star$ that matches x_i and y_i for all $i \in \{1, \dots, n\}$ minimizes both α_1 and β_1.*

The pairing given by $M^\star$ directly results in a 2-approximation for the alternating stock size problem, by applying Lemma 3. Without loss of generality, let us assume that $\max\{\alpha_1, \beta_1\} =$

α_1, and observe that $\alpha_1 \leq \mu$. Then $M^\star$ partitions the input into $(\alpha_1/\mu, \mu)$-pairs. Applying Lemma 3, we obtain an algorithm that computes a solution with value at most $\mu + \alpha_1 \leq 2\mu$. We note that if $\alpha_1 \leq (1-\epsilon)\mu$, then we have a $(2-\epsilon)$-approximation.

2.2 Lower Bound for the Alternating Stock Size Problem

In order to obtain an approximation ratio better than 2, we need to use a lower bound that is more accurate than μ. We now introduce a lower bound closely related to the one given for the stock size problem in [11] (Lemma 8). We refer to a real number C, which divides the sets X and Y into sets of small jobs and big jobs, as a *barrier*. Let $C \leq \mu$ be a barrier such that:

$$X = \{a_1 \geq a_2 \geq \ldots \geq a_{n_a} \geq \quad C \quad > v_k \geq v_{k-1} \geq \ldots \geq v_1\}, \tag{5}$$

$$Y = \{\, b_1 \geq b_2 \geq \ldots \geq b_{n_b} \geq \quad C \quad > w'_1 \geq w'_2 \geq \ldots w'_{n_a-n_b} \geq w_1 \geq w_2 \geq \ldots \geq w_k\}, \tag{6}$$

where, without loss of generality, $n_a \geq n_b$. (If not then by swapping the x's and the y's we have a symmetric sequencing problem with $n_a \leq n_b$). The elements of (5) are all the x-jobs (partitioned into the sets A and V) and the elements of (6) are all the y-jobs. The jobs in Y that have value at most C are partitioned into W' and W.

Let $A' = \{a_{n_b+1}, \ldots a_{n_a}\} = \{a'_1, \ldots a'_{n_a-n_b}\}$, let V_i denote the i smallest v_j's, i.e. $\{v_1, v_2, \ldots, v_i\}$, and let W_i denote the i largest w_j's in W, i.e. $\{w_1, w_2, \ldots, w_i\}$. (Note that A', V_i, and W_i each depend on C, but in order to avoid cumbersome notation, we do not use superscript C.) Let $s \in \{1, \ldots, n_a - n_b\}$. After fixing a barrier C, let h be the (unique) index such that $w_h > v_h$ and $w_{h+1} \leq v_{h+1}$, and recall that S^* is the value of an optimal ordering. Then we obtain the following lower bound on S^*.

▶ **Lemma 5.** *For $n_a > n_b$, $1 \leq s \leq n_a - n_b$, the following inequality holds:*

$$S^* \geq LB(C) = \frac{1}{n_a - n_b - s + 1}\left(2\sum_{i=s}^{n_a-n_b} a'_i - \sum_{i=s}^{n_a-n_b} w'_i + \sum_{i=1}^{h}(v_i - w_i)\right).$$

2.3 Alternating Batches: Definition

We need a few more tools before we can outline our new algorithm. The notion of *batches* introduced in [11], to which we briefly alluded before Lemma 3, is quite useful for the stock size problem. For $B \subseteq X \cup Y$, let $x(B)$ and $y(B)$ denote the total value of the x-jobs and y-jobs, respectively, in B. In its original form, the batching lemma (Lemma 3, [11]) calls for a partition of the input into groups or batches such that for some fixed positive real numbers T and $q \leq 1$, each group B has the following properties: $x(B), y(B) \leq T$ and $|x(B) - y(B)| \leq qT$. Given such a partition of the input, a sequence with stock size at most $(1+q)T$ can be produced.

This approach is not directly applicable to the alternating stock size problem, because the output is not necessarily an alternating sequence. However, we will now show that the procedure can be modified to yield a valid ordering. With this goal in mind, we define a new type of batch, which we call an *alternating batch*. An alternating batch will either contain two jobs (small) or more than two jobs (large).

The modified procedure to construct an ordering of the jobs first partitions the input into alternating batches, then orders these batches, and finally orders the jobs contained within each batch. In the case of a small alternating batch, the batch will contain both an x-job and a y-job, and the last step simply preserves this order. A large alternating batch will be required to fulfill certain additional properties that allow the elements to be sequenced in a way that is both alternating and feasible, i.e. all prefixes are nonnegative.

Suppose $B = \{(x'_1, y'_1), (x'_2, y'_2), \ldots, (x'_\ell, y'_\ell)\}$, and consider the following four properties:

(i) $\sum_{i=1}^{\ell} x'_i - \sum_{i=1}^{\ell} y'_i \geq 0$,
(ii) $x'_1 - y'_1 \geq 0$,
(iii) $x'_i - y'_i \leq 0$, for $2 \leq i \leq \ell$,
(iv) $y'_1 \geq y'_2 \geq \ldots \geq y'_\ell$.

▶ **Lemma 6.** *If a batch B satisfies properties (i), (ii), (iii) and (iv), then we can sequence the elements in B so that the items alternate, each prefix is non-negative and the maximum height (or prefix sum) of the sequence is x'_1.*

▶ **Definition 7.** We call a set B a $(1-\epsilon)$-*alternating batch* if $B = \{(x'_1, y'_1), (x'_2, y'_2), \ldots, (x'_\ell, y'_\ell)\}$ such that:
(1) $|\sum_{i=1}^{\ell} x'_i - \sum_{i=1}^{\ell} y'_i| \leq (1-\epsilon)\mu$,
(2) if $\ell > 1$, then conditions (i) to (iv) hold.

▶ **Definition 8.** We say that a $(1-\epsilon)$-alternating batch with more than two jobs is a *large* alternating batch. In other words, a large alternating batch obeys conditions (1) and (2) in Definition 7. A *small* alternating batch contains only two jobs and obeys condition (1) in Definition 7.

Note that, by definition, in a large alternating batch B, the sum of the x-jobs in B is at least the sum of the y-jobs in B.

▶ **Lemma 9.** *If the sets X and Y can be partitioned into large and small $(1-\epsilon)$-alternating batches, then we can find an alternating sequence with maximum stock size less than $(2-\epsilon)\mu$.*

2.4 Alternating Batches: Construction

In this section, we present the final tool required for our algorithm. Suppose that for some some $\epsilon : 0 \leq \epsilon \leq 1$, the following conditions hold for an input instance to the alternating stock size problem: $\alpha_1 > (1-\epsilon)\mu$, and $LB(C) < \frac{2}{2-\epsilon}\mu$, for $C = (1-\epsilon)\mu$. Then, we claim, there is some value of ϵ (to be determined later) for which the above two conditions can be used to partition the input into $(1-\epsilon)$-alternating batches, to which we can then apply Lemma 9. In this section, we will heavily rely on the notation introduced in Section 2.2.

The sets $A' = \{a'_1, \ldots, a'_{n_a - n_b}\}$ and $W' = \{w'_1, \ldots, w'_{n_a - n_b}\}$ contain exactly the pairs in $M^\star$ that are split by barrier C. Let s be the smallest index such that $w'_s < \epsilon\mu$. To see that such an s actually exists, we note the following. Let $i^\star$ denote the index such that $x_{i^\star} - y_{i^\star} = \alpha_1$. Then $y_{i^\star} < \epsilon\mu$ and the pair $(x_{i^\star}, y_{i^\star})$ is split by C. Thus, $y_{i^\star}$ corresponds to some $w'_{i'}$, and therefore $s \leq i'$. See Figure 1 for a schematic drawing.

For i in $\{1, \ldots, n_a - n_b\}$, we define $\alpha'_i = a'_i - w'_i$ and for j in $\{1, \ldots, h\}$, $\beta'_j = w_j - v_j$. (Recall that for $j \in \{1, \ldots, h\}$, $w_j - v_j > 0$.) Furthermore, let $\mathcal{A}_i$ denote the pair $\{a'_i, w'_i\}$ and let $\mathcal{B}_j$ denote the pair $\{v_j, w_j\}$. Since $w'_s < \epsilon\mu$, it follows that all w_i's in W also have value less than $\epsilon\mu$. Moreover, $\beta'_j < \epsilon\mu$ for $j \in \{1, \ldots, h\}$.

Our goal is now to construct $(1-\epsilon)$-alternating batches. For each $i \in \{1, \ldots s-1\}$, note that $\alpha'_i \leq (1-\epsilon)\mu$. The set $\mathcal{A}_i$ therefore forms a small $(1-\epsilon)$-alternating batch. For each $\mathcal{A}_i$ where $i \in \{s, \ldots, n_a - n_b\}$, we will find a set of $\mathcal{B}_j$'s that can be grouped with this $\mathcal{A}_i$ to create a large $(1-\epsilon)$-alternating batch. However, to do this, we require that the condition on ϵ found in Claim 10 be satisfied.

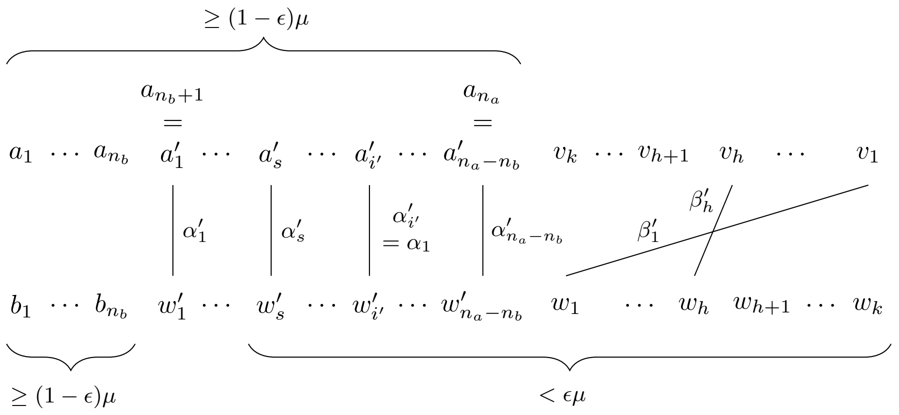


Figure 1 An illustration of the various elements used in the construction of the lower bound.

▶ **Claim 10.** *The condition*

$$2(1-\epsilon) - \frac{2}{2-\epsilon} > 2\epsilon \tag{7}$$

is satisfied when $\epsilon = .21$.

▶ **Lemma 11.** *If* $LB(C) < 2\mu/(2-\epsilon)$, $C = (1-\epsilon)\mu$, *and* $2(1-\epsilon) - \frac{2}{2-\epsilon} > 2\epsilon$, *then* $\sum_{i=1}^{h} \beta'_i + \sum_{i=s}^{n_a-n_b} w'_i > 2\epsilon\mu(n_a - n_b - s + 1)$.

For ease of notation, we set $d = n_a - n_b - s + 1$. In the following lemma, we show that we can also construct a $(1-\epsilon)$-alternating batch for each $\mathcal{A}_i$ for $i \in [s, n_a - n_b]$.

▶ **Lemma 12.** *There exists d disjoint subsets* $S_1, \ldots, S_d$ *of* $\{\mathcal{B}_1, \ldots, \mathcal{B}_h\}$ *such that for all i in* $\{1, \ldots, d\}$, *the set* $S_i \cup \mathcal{A}_{i+s-1}$ *is a* $(1-\epsilon)$*-alternating batch.*

Now we want to complete the construction of the $(1-\epsilon)$-alternating batches, so that we can apply Lemma 9. For the sets $\mathcal{A}_i$, where $i \in \{s, \ldots, n_a - n_b\}$, we construct batches according to Lemma 12. Let $y_{i^*} = w'_s$. For all $i < i^*$, the pair (x_i, y_i) form a small $(1-\epsilon)$-alternating batch. This follows from the fact that for all $i < i^*$, $y_{i^*} \geq \epsilon\mu$, by definition of s. Finally, if there are remaining elements, they are v_i's and w_i's, which can be paired up arbitrarily to construct more small $(1-\epsilon)$-alternating batches, since each remaining v_i has value stictly less than $(1-\epsilon)\mu$ due to our choice of barrier, and each remaining w_i has value at most $\epsilon\mu$. Since the only limits on the value of ϵ are imposed by Lemma 11, we can set $\epsilon = .21$ and partition the input into .79-alternating batches.

2.5 A 1.79-Approximation Algorithm

We are now ready to present an algorithm for the alternating stock size problem with an approximation guarantee of 1.79.

▶ **Theorem 13.** *Algorithm 1 is a* 1.79*-approximation for the alternating stock size problem.*

3 Gasoline Problem

Let the variable z_{ij} be 1 if gas station x_i is placed in position j, and be 0 otherwise. Then we can formulate the gasoline problem as the following integer linear program whose solution matrix Z is a permutation matrix.

Algorithm 1 1.79-approximation

```
1: Input: the sets $X$ and $Y$ of positive numbers sorted in nonincreasing order.
2: Output: a sequence that is a 1.79-approximation.
3: Set $\epsilon = .21, C = (1-\epsilon)\mu$.
4: Match each $x_i$ with $y_i$.
5: if $\alpha_1 \leq (1-\epsilon)\mu$ or if $LB(C) \geq \frac{2}{2-\epsilon}\mu$ then
6:     return solution for the Pairing Algorithm with guarantee of most $\mu + \alpha_1$.
7: else
8:     Partition the input into $(1-\epsilon)$-alternating batches described in Section 2.4.
9:     Run algorithm from Lemma 9 on the $(1-\epsilon)$-alternating batches.
10: end if
```

$$\min \beta - \alpha$$

$$\forall j \in [1,n] : \sum_{i=1}^{n} z_{ij} = 1, \quad \forall i \in [1,n] : \sum_{j=1}^{n} z_{ij} = 1, \quad \forall i,j \in [1,n] : z_{ij} \in \{0,1\},$$

$$\forall k \in \{1,\dots,n\} : \sum_{j=1}^{k}\sum_{i=1}^{n} z_{ij} \cdot x_i - \sum_{j=1}^{k-1} y_j \leq \beta, \tag{8}$$

$$\forall k \in \{1,\dots,n\} : \sum_{j=1}^{k}\sum_{i=1}^{n} z_{ij} \cdot x_i - \sum_{j=1}^{k} y_j \geq \alpha. \tag{9}$$

Observe that (8) and (9) imply that for every interval $I = [k,\ell]$ the sum of the x_i's assigned to I by Z and the sum of the y_i's in I differ by at most $\beta - \alpha$. If we replace $z_{ij} \in \{0,1\}$ with the constraint $z_{ij} \in [0,1]$, then the solution to the linear program, Z, is an $n \times n$ doubly stochastic matrix. Now we have the following rounding problem. We are given an $n \times n$ doubly stochastic matrix $Z = \{z_{ij}\}$ and we define z_j to be the total fractional value of the x_i's that are in position j, i.e., $z_j = \sum_{i=1}^{n} z_{ij} \cdot x_i$. Our goal is to find a permutation of the x_i's such that the x_i assigned to position j is roughly equal to z_j.

A natural approach would be to decompose Z into a convex combination of permutation matrices and see if one of these gives a good permutation of the elements in X. However, consider the following example:

$$X = \{\underbrace{1,1,\dots,1}_{n-k \text{ entries}},\underbrace{B,B,\dots,B}_{k \text{ entries}}\}, \quad \forall i \in [1,n] : \; y_i = \gamma = \frac{k \cdot B + n - k}{n}.$$

In this case, $z_j = \gamma$ for all $j \in [1,n]$. Thus, a possible decomposition into permutation matrices could look like:

$$\{B,B,\dots,B,1,1,\dots,1,1\}$$
$$\{1,B,B,\dots,B,1,1,\dots,1\}$$
$$\dots$$
$$\{1,1,\dots,1,1,B,B,\dots,B\}.$$

Each of these permutations has an interval with very large value, while the optimal permutation of the elements in X is

$$\{1,1,\dots 1,B,1\dots,1,B,1,\dots 1\}.$$

Algorithm 2 SHIFT$(Z, j, i_1, i_2, i_3, \delta)$

```
1: $\forall i \in \{1,\dots,n\} \setminus \{i_1,i_2,i_3\} : a_i = z_{ij}$;
2: $a_{i_2} = z_{i_2 j} + \delta$;
3: if $x_{i_1} = x_{i_3}$ then
4:     $a_{i_1} = z_{i_1 j} - \delta$;    $a_{i_3} = z_{i_3 j}$;
5: else
6:     $a_{i_1} = z_{i_1 j} - \delta \cdot \frac{x_{i_2} - x_{i_3}}{x_{i_1} - x_{i_3}}$;    $a_{i_3} = z_{i_3 j} - \delta \cdot \frac{x_{i_1} - x_{i_2}}{x_{i_1} - x_{i_3}}$;
7: end if
8: return $a$
```

Algorithm 3 TRANSFORM(Z, j, i_1, i_2, i_3)

```
1: The $j^{\text{th}}$ column of $Z'$ equals SHIFT$(Z, j, i_1, i_2, i_3, \delta)$ for $\delta > 0$ to be chosen later.
2: Let $j' > j$ denote the smallest index larger than $j$ with $z_{i_2 j'} > 0$. Such an index
   must exist because row $i_2$ is not finished in $Z$ in column $j$. The $(j')^{\text{th}}$ column of $Z'$
   equals SHIFT$(Z, j', i_1, i_2, i_3, -\delta)$.
3: All columns of $Z$ and $Z'$, except for columns $j$ and $j'$, coincide.
4: The value $\delta$ is chosen as the largest value for which all entries of $Z'$ are in $[0, 1]$. This
   value must be strictly larger than 0 due to our choice of $j$, $j'$, $i_1$, $i_2$, and $i_3$.
5: return $Z'$
```

3.1 Transformation

Given a doubly stochastic matrix $Z = \{z_{ij}\}$, we transform it into a doubly stochastic matrix $T = \{t_{ij}\}$ with special properties. First of all, for each j, $z_j = \sum_{i=1}^{n} t_{ij} \cdot x_i$. This means that if (Z, α, β) is a feasible solution to the linear program then (T, α, β) is also a feasible solution. In particular, if Z is an optimal solution, for which $\beta - \alpha$ is as small as possible, then T is also optimal.

We call a row i in a doubly stochastic matrix $A = \{a_{ij}\}$ *finished* at column ℓ if $\sum_{j=1}^{\ell} a_{ij} = 1$. We say that a matrix T has the *consecutiveness property* if the following holds: for each column j and any rows i_1 and i_3 with $i_1 < i_3$, $t_{i_1 j} > 0$, and $t_{i_3 j} > 0$, each row $i_2 \in \{i_1 + 1, \dots, i_3 - 1\}$ is finished at column j.

Our procedure to transform the matrix Z into a matrix T with the desired property relies on the following transformation rule. Assume that there exist indices j, i_1, i_3, and $i_2 \in \{i_1 + 1, \dots, i_3 - 1\}$ such that $z_{i_1 j} > 0$, $z_{i_3 j} > 0$, and row i_2 is not finished in matrix Z at column j. Then the procedure SHIFT shown as Algorithm 2 computes a column vector $a = (a_1, \dots, a_n)$, which satisfies the following lemma.

▶ **Lemma 14.** *For any $\delta \geq 0$, the vector a returned by* SHIFT$(Z, j, i_1, i_2, i_3, \delta)$ *satisfies* $\sum_{i=1}^{n} a_i \cdot x_i = z_j$.

Let Z' denote the matrix that we obtain from Z if we replace the j^{th} column by the vector a returned by the procedure SHIFT. The previous lemma shows that Z' satisfies (8) and (9) for the same β and α as Z because the value z_j is not changed by the procedure. However, the matrix Z' is not doubly stochastic because the rows i_1, i_2, and i_3 do not add up to one anymore. In order to repair this, we have to apply the SHIFT operation again to another column with $-\delta$. Formally, let us redefine the matrix $Z' = \{z'_{ij}\}$ as the outcome of the operation TRANSFORM shown as Algorithm 3.

Observe that Z' is a doubly stochastic matrix because the rows i_1, i_2, and i_3 sum up to one and all entries are from $[0, 1]$. Applying Lemma 14 twice implies that (Z', β, α) is a feasible solution to the linear program if (Z, β, α) is one.

We will transform Z by a finite number of applications of the operation TRANSFORM. As long as the current matrix T (which is initially chosen as Z) does not have the consecutiveness property, let j be the smallest index for which there exist indices i_1, i_3, and $i_2 \in \{i_1+1,\dots,i_3-1\}$ such that $t_{i_1j} > 0$, $t_{i_3j} > 0$, and row i_2 is not finished in T at column j. Furthermore, let i_1 and i_3 be the smallest and largest index with $t_{i_1j} > 0$ and $t_{i_3j} > 0$, respectively, and let i_2 be the smallest index from $\{i_1+1,\dots,i_3-1\}$ for which row i_2 is not finished at column j. We apply the operation TRANSFORM(T, j, i_1, i_2, i_3) to obtain a new matrix T.

▶ **Lemma 15.** *After at most a polynomial number of* TRANSFORM *operations, no further such operation can be applied. Then T is a doubly stochastic matrix with the consecutiveness property.*

In the remainder, we will not need the matrix Z anymore but only matrix T. For convenience, we will use the notation $t_j = \sum_{i=1}^{n} t_{ij} \cdot x_i$ instead of z_j even though the transformation ensures that t_j and z_j coincide.

We now define a graph whose connected components or *blocks* will correspond to the row indices from columns that overlap. More formally, let $V = \{1,\dots,n\}$ denote a set of vertices and let G_0 be the empty graph on V. Each column j of T defines a set E_j of edges as follows: the set E_j is a clique on the vertices $i \in V$ with $t_{ij} > 0$, i.e., E_j contains an edge between two vertices i and i' if and only if $t_{ij} > 0$ and $t_{i'j} > 0$. We denote by G_j the graph on V with edge set $E_1 \cup \dots \cup E_j$.

▶ **Definition 16.** A *block* in G_j is a set of indices in $[1, n]$ that forms a connected component in G_j. A block in G_j is called finished if all rows in T corresponding to the indices it contains are finished at column j. Similarly, if a block in G_j contains at least one unfinished row at column j, it is called an *unfinished block.*

If $B \subseteq \{1,\dots,n\}$ is a block in G_j with $i \in B$ then we will say that *block B contains row i*. For the following lemma it is convenient to define a matrix $C = \{c_{ij}\}$, which is the cumulative version of T. To be more precise, the j^{th} column of C equals the sum of the first j columns of T.

▶ **Lemma 17.** *The following three properties are satisfied for every j.*

1. *Let B be a block in G_j and let $k = \sum_{i\in B} c_{ij}$ the denote the value of block B at column j. The number of rows in B is k if B is finished and it is $k+1$ if B is an unfinished block.*
2. *The set of blocks in G_j emerges from the set of blocks in G_{j-1} by either merging exactly two unfinished blocks or by making one unfinished block finished.*
3. *Let $B_1,\dots,B_\ell$ denote the unfinished blocks in G_j. Then there exist non-overlapping intervals $I_1,\dots,I_\ell \subseteq [1, n]$ with $B_i \subseteq I_i$ for every i.*

One might ask if the consecutiveness property is satisfied by every optimal extreme point of the linear program. Let us mention that this is not the case. A simple counterexample is provided by the instance $X = \{9, 6, 4, 1\}$ and $Y = \{5, 5, 5, 5\}$. In this instance, an optimal extreme point would be, for example, to take one half of each of the items x_1 and x_4 in steps one and three and to take one half of each of the items x_2 and x_3 in steps two and four. This extreme point does however not satisfy the consecutiveness property. Hence, the transformation described in this section is necessary.

3.2 Rounding

In this section, we use the transformed matrix T to create the solution matrix R, which is a doubly stochastic 0/1 matrix, i.e., a permutation matrix. We apply the following rounding method.

```
1: for $j = 1$ to $n$ do
2:     Let $B$ denote the active block in $G_j$, i.e., the block that contains the rows $i$ with $t_{ij} > 0$.
3:     Let $p$ denote the smallest index in $B$ such that $r_{pi} = 0$ for all $i < j$.
4:     Set $r_{pj} = 1$ and $r_{qj} = 0$ for all $q \neq p$.
5: end for
```

Observe that the first step is well-defined because all non-zero entries in column j belong by definition to the same block of G_j. The resulting matrix R will be doubly stochastic, since each column contains a single one, as does each row. We just need to prove that in Line 3 there always exists a row $p \in B$ that is unfinished in R at column $j-1$. This follows from the first part of the next lemma because, due to Lemma 17, the active block B in G_j emerges from one or two unfinished blocks in G_{j-1} and these blocks each contain a row that is unfinished in R at column $j-1$.

▶ **Lemma 18.** *Let B be a block in G_j for some $j \in \{1,\dots,n\}$.*

1. *If B is an unfinished block in G_j and p is the largest index in B, then $r_{pi} = 0$ for all $i \le j$ and all rows corresponding to $B \setminus \{p\}$ are finished in R at column j.*
2. *If B is a finished block in G_j, then for all $q \in B$, row q is finished in R at column j.*

We define the *value* of a permutation matrix M to be the smallest γ for which there exist α' and β' with $\gamma = \beta' - \alpha'$ such that (M, α', β') is a feasible solution to the linear program.

▶ **Theorem 19.** *Let (T, α, β) be an optimal solution to the linear program. Then $(R, \alpha, \beta + \mu_x)$ is a feasible solution to the linear program. Hence, the value of the matrix R is at most $(\beta - \alpha) + \mu_x \le 2 \cdot OPT$, where OPT denotes the value of the optimal permutation matrix.*

For ease of notation, we define r_j as follows: $r_j = \sum_{i=1}^{n} r_{ij} \cdot x_i$. Note that r_j corresponds to the value of the element from Y that the algorithm places in position j. We will see later that Theorem 19 follows easily from the next lemma.

▶ **Lemma 20.** *For each $k \in \{1,\dots,n\}$,*

$$\sum_{j=1}^{k} (r_j - t_j) \in [0, \mu_x]. \qquad (10)$$

We need the following lemma in the proof of Lemma 20.

▶ **Lemma 21.** *Let b be the largest index in an unfinished block B in G_j. Then,*

$$c_{bj} \quad = \sum_{i \in B \setminus \{b\}} (1 - c_{ij}).$$

Proof. Let the value of the unfinished block B be $k = \sum_{i \in B} c_{ij}$. By property 1 of Lemma 17, block B consists of $k+1$ rows. Thus, we have:

$$c_{bj} = k - \sum_{i \in B \setminus \{b\}} c_{ij} = \sum_{i \in B \setminus \{b\}} (1 - c_{ij}). \qquad ◀$$

Proof of Lemma 20. Let us consider the sets of finished and unfinished blocks in G_k, $\mathcal{B}_F$ and $\mathcal{B}_U$, respectively. For a block $B \in \mathcal{B}_F \cup \mathcal{B}_U$, we denote by

$$\mathrm{er}_k(B) = \sum_{i \in B} \sum_{j=1}^{k} x_i(r_{ij} - t_{ij})$$

its rounding error. Since each row is contained in exactly one block of G_k,

$$\sum_{j=1}^{k}(r_j - t_j) = \sum_{j=1}^{k}\sum_{i=1}^{n} x_i(r_{ij} - t_{ij}) = \sum_{i=1}^{n}\sum_{j=1}^{k} x_i(r_{ij} - t_{ij}) = \sum_{B \in \mathcal{B}_F \cup \mathcal{B}_U} \mathrm{er}_k(B). \tag{11}$$

Hence, in order to prove the lemma, it suffices to bound the rounding errors of the blocks.

If block B is finished in G_k, then all rows that belong to B are finished in T and in R (due to property 2 of Lemma 18) at column k. Hence,

$$\mathrm{er}_k(B) = \sum_{i \in B}\sum_{j=1}^{k} x_i(r_{ij} - t_{ij}) = \sum_{i \in B} x_i \cdot \left(\sum_{j=1}^{k} r_{ij} - \sum_{j=1}^{k} t_{ij}\right) = \sum_{i \in B} x_i \cdot (1-1) = 0. \tag{12}$$

Now consider an unfinished block B in G_k, and let a and b denote the smallest and largest index in B, respectively. By Lemma 18, all rows in the block except for b are finished in R at column k (i.e., $\sum_{j=1}^{k} r_{ij} = 1$ for $i \in B \setminus \{b\}$ and $\sum_{j=1}^{k} r_{bj} = 0$). The rounding error of B can thus be bounded as follows (remember that $c_{ik} = \sum_{j=1}^{k} t_{ij}$):

$$\begin{aligned}
\mathrm{er}_k(B) &= \sum_{i \in B}\sum_{j=1}^{k} x_i(r_{ij} - t_{ij}) = \sum_{i \in B} x_i \sum_{j=1}^{k} r_{ij} - \sum_{i \in B} x_i \sum_{j=1}^{k} t_{ij} \\
&= \sum_{i \in B\setminus\{b\}} x_i - \sum_{i \in B} x_i c_{ik} = \sum_{i \in B\setminus\{b\}} x_i(1 - c_{ik}) - x_b c_{bk} \\
&= \sum_{i \in B\setminus\{b\}} x_i(1 - c_{ik}) - x_b \sum_{i \in B\setminus\{b\}} (1 - c_{ik}) && (13) \\
&= \sum_{i \in B\setminus\{b\}} (x_i - x_b)(1 - c_{ik}) \\
&\le (x_a - x_b) \sum_{i \in B\setminus\{b\}} (1 - c_{ik}) && (14) \\
&= (x_a - x_b) \cdot c_{bj} && (15) \\
&\le x_a - x_b. && (16)
\end{aligned}$$

Equations (13) and (15) follow from Lemma 21. Inequality (16) follows from the fact that $c_{bj} \le 1$. Inequality (14) follows from the facts that $1 - c_{ik} \ge 0$ and $x_i - x_b \ge 0$ for all $i \in B$. These facts also imply that $\mathrm{er}_k(B) \ge 0$. Hence,

$$\mathrm{er}_k(B) \in [0, x_a - x_b]. \tag{17}$$

Together (11) and (12) imply

$$\sum_{j=1}^{k}(r_j - t_j) = \sum_{B \in \mathcal{B}_F \cup \mathcal{B}_U} \mathrm{er}_k(B) = \sum_{B \in \mathcal{B}_F} \mathrm{er}_k(B) + \sum_{B \in \mathcal{B}_U} \mathrm{er}_k(B) = \sum_{B \in \mathcal{B}_U} \mathrm{er}_k(B). \tag{18}$$

Now, let $B_1, \ldots B_h$ denote the unfinished blocks in G_k, and for each block B_f in $\mathcal{B}_U$, let a_f and b_f denote the minimum and maximum indices, respectively, contained in the block.

Property 3 of Lemma 17 implies that the intervals $[a_f, b_f]$ are pairwise disjoint. Hence, (17) implies

$$\sum_{B\in\mathcal{B}_U} \mathrm{er}_k(B) \in \left[0, \sum_{f=1}^{h}(x_{a_f} - x_{b_f})\right] \subseteq \left[0, x_1 - x_n\right] \subseteq [0, \mu_x].$$

Together with (18) this implies the lemma. ◀

Now we are ready to prove Theorem 19.

Proof of Theorem 19. Let (T, α, β) denote an optimal solution to the linear program. By definition, our rounding method produces a permutation matrix R. Lemma 20 implies that $(R, \alpha, \beta + \mu_y)$ is also a feasible solution to the linear program because for each $k \in \{1, \ldots, n\}$,

$$\sum_{j=1}^{k} x_j - \sum_{j=1}^{k-1}\sum_{i=1}^{n} r_{ij}\cdot x_i = \sum_{j=1}^{k} x_j - \sum_{j=1}^{k-1} r_j \le \sum_{j=1}^{k} x_j - \sum_{j=1}^{k-1} t_j + \mu_x \le \beta + \mu_x$$

and

$$\sum_{j=1}^{k} x_j - \sum_{j=1}^{k}\sum_{i=1}^{n} r_{ij}\cdot x_i = \sum_{j=1}^{k} x_j - \sum_{j=1}^{k} r_j \ge \sum_{j=1}^{k} x_j - \sum_{j=1}^{k} t_j \ge \alpha.$$

Now the theorem follows because OPT $\ge \mu_x$ and OPT $\ge \beta - \alpha$. ◀

4 Conclusions

We have introduced two new variants of the stock size problem and have presented non-trivial approximation algorithms for them. The most intriguing question for our variants as well as for the original stock size problem is if the approximation guarantees can be improved. Each of these problems is NP-hard but no APX-hardness is known. So it is conceivable that there exists a PTAS. Closing this gap seems very challenging.

We note that the additive integrality gap of the linear program in Section 3 can be arbitrarily close to μ_y. Consider the following instance:

$$x = \frac{(n-1)+\mu}{n}, \quad X = \{\underbrace{x, \ldots, x}_{n \text{ entries}}\}, \quad Y = \{\mu, \underbrace{1, 1, \ldots, 1}_{n-1 \text{ entries}}\}.$$

Then the value of the linear program is x. However, the optimal value is μ, which can be arbitrarily larger than x.

Acknowledgments. We would like to thank Anupam Gupta for several useful examples and enlightening conversations. We would like to thank Jochen Könemann for pointing out a connection between the alternating stock size problem and the optimization version of the gasoline puzzle. We would also like to thank Tamás Kis for pointing out related papers on scheduling problems.

Part of this work was done during the trimester program on Combinatorial Optimization at the Hausdorff Institute for Mathematics in Bonn and we would like to thank HIM for their organization and hospitality during the program.

References

1 H. M. Abdel-Wahab and T. Kameda. Scheduling to minimize maximum cumulative cost subject to series-parallel precedence constraints. *Operations Research*, 26(1):141–158, 1978.

2 Wojciech Banaszczyk. The steinitz constant of the plane. *Journal für die Reine und Angewandte Mathematik*, 373:218–220, 1987.

3 Imre Bárány. On the power of linear dependencies. In *Building bridges*, pages 31–45. Springer, 2008.

4 Jacek Blazewicz, Jan Karel Lenstra, and AHG Rinnooy Kan. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5(1):11–24, 1983.

5 Dirk Briskorn, Byung-Cheon Choi, Kangbok Lee, Joseph Leung, and Michael Pinedo. Complexity of single machine scheduling subject to nonnegative inventory constraints. *European Journal of Operational Research*, 207(2):605–619, 2010.

6 J Carlier and AHG Rinnooy Kan. Scheduling subject to nonrenewable-resource constraints. *Operations Research Letters*, 1(2):52–55, 1982.

7 V. S. Grinberg and S. V. Sevastyanov. The value of the Steinitz constant. *Functional Analysis and its Applications*, 14:56–57, 1980.

8 Péter Györgyi and Tamás Kis. Approximation schemes for single machine scheduling with non-renewable resource constraints. *Journal of Scheduling*, 17(2):135–144, 2014.

9 Péter Györgyi and Tamás Kis. Approximability of scheduling problems with resource consuming jobs. *Annals of Operations Research*, 235(1):319–336, 2015.

10 Nick Harvey and Samira Samadi. Near-optimal herding. In *COLT*, pages 1165–1182, 2014.

11 Hans Kellerer, Vladimir Kotov, Franz Rendl, and Gerhard J. Woeginger. The stock size problem. *Operations Research*, 46(3):S1–S12, 1998.

12 L. Lovász. *Combinatorial Problems and Exercises*. North-Holland, 1979.

13 Clyde L. Monma. Sequencing to minimize the maximum job cost. *Operations Research*, 28(4):942–951, 1980.

14 Ehab Morsy and Erwin Pesch. Approximation algorithms for inventory constrained scheduling on a single machine. *Journal of Scheduling*, 18(6):645–653, 2015.

15 Klaus Neumann and Christoph Schwindt. Project scheduling with inventory constraints. *Mathematical Methods of Operations Research*, 56(3):513–533, 2003.

16 Alantha Newman, Heiko Röglin, and Johanna Seif. The alternating stock size problem and the gasoline puzzle. *arXiv:1511.09259*, 2015.

New Parameterized Algorithms for APSP in Directed Graphs

Ely Porat[1], Eduard Shahbazian[2], and Roei Tov[3]

1 **Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel**
`porately@gmail.com`

2 **Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel**
`shediyo@gmail.com`

3 **Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel**
`roei81@gmail.com`

Abstract

All Pairs Shortest Path (APSP) is a classic problem in graph theory. While for general weighted graphs there is no algorithm that computes APSP in $O(n^{3-\varepsilon})$ time ($\varepsilon > 0$), by using fast matrix multiplication algorithms, we can compute APSP in $O(n^{\omega} \log n)$ time ($\omega < 2.373$) for undirected unweighted graphs, and in $O(n^{2.5302})$ time for directed unweighted graphs. In the current state of matters, there is a substantial gap between the upper bounds of the problem for undirected and directed graphs, and for a long time, it is remained an important open question whether it is possible to close this gap.

In this paper we introduce a new parameter that measures the *symmetry* of directed graphs (i.e. their closeness to undirected graphs), and obtain a new parameterized APSP algorithm for directed unweighted graphs, that generalizes Seidel's $O(n^{\omega} \log n)$ time algorithm for undirected unweighted graphs. Given a directed unweighted graph G, unless it is highly asymmetric, our algorithms can compute APSP in $o(n^{2.5})$ time for G, providing for such graphs a faster APSP algorithm than the state-of-the-art algorithms for the problem.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Graphs, distances, APSP, fast matrix multiplication

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.72

1 Introduction

All Pairs Shortest Path (APSP) has a long history, emerging from the 1950's to our present days. In APSP our goal is to compute the distances between all pairs of vertices in the graph. For general directed weighted graphs with n vertices, the algorithm of Floyd-Warshall [6] computes APSP in $O(n^3)$ time. For sparse graphs with m edges, we can obtain an improved algorithm that runs in $O(mn + n^2 \log n)$ time, by first finding and eliminating cycles of negative weight, using Johnson's algorithm [15], and then executing Dijkstra algorithm [6] (implemented with Fibonacci heaps [8]) from each vertex in the graph. This classic result was later improved by Pettie [16] to $O(mn + n^2 \log\log n)$, and for undirected graphs with nonnegative edge weights, APSP can be computed in $O(mn)$ time, using Thorup's algorithm for single source shortest path [24].

All the algorithms mentioned above have a running time of $O(n^3)$ for dense graphs. Fredman was the first to break the $O(n^3)$ (cubic) time barrier, by obtaining an algorithm with a running time of $O(n^3/(\log\log n/\log n)^{1/3})$ [7]. Since then, a line of subsequent improvements to Fredman's algorithm followed (e.g. [22, 12, 23, 27, 13, 3], etc.), where the current best result for the problem was recently obtained by Chan and Williams [4], which showed an algorithm that computes APSP in $O(n^3/2^{\Omega(\log n)^{1/2}})$ time. Nevertheless, these algorithms only provides a slightly improvement to the classic cubic-time algorithm of Floyd and Warshall, and it remains an open question whether an $O(n^{3-\varepsilon})$-time algorithm for APSP exists. The hardness of APSP and other fundamental graph and matrix problems (e.g. minimum weight cycle, replacement paths on directed weighted graphs, etc.) that are all known to have cubic time algorithms but no $O(n^{3-\varepsilon})$-time algorithms, might be explained by the result of V. Vassilevska Williams and R. Williams [25], which proved that these problems are subcubic equivalent, meaning, either all of these problems can be solved in $O(n^{3-\varepsilon})$ time or none of them can be.

By using Fast Matrix Multiplication (FMM) algorithms, truly subcubic time (i.e. $O(n^{3-\varepsilon})$) algorithms can be obtained for unweighted graphs and for graphs with small integer edge-weights. The naïve algorithm for multiplying two $n \times n$ matrices runs in $O(n^3)$ time, however, there exist faster algorithms to compute matrix multiplication (e.g. Strassen's algorithm [6], Coppersmith-Winograd [5]). Denote ω to be the exponent of square matrix multiplication, currently $\omega < 2.373$ ([11]) is the smallest known value for ω. Notice, it is straightforward to reduce Boolean Matrix Multiplication (BMM) to FMM in $O(n^2)$ time, hence BMM can be also computed in $O(n^\omega)$ time. Given this fact, many APSP algorithms use the following basic property: let G be an unweighted graph with adjacency matrix A and let $M = A^k$, it follows $M[i,j] = 1$ if and only if there is a shortest path in G from i to j of length at most k. Alon, Galil and Margalit [2] were the first to obtain a truly subcubic algorithm for APSP. They showed an algorithm that computes APSP in $\widetilde{O}(n^{(\omega+3)/2}) = \widetilde{O}(n^{2.69})$ time[1] for directed graphs with edge-weights from $\{-1,0,1\}$. Zwick in [26] improved Alon, Galil, and Margalit's result [2]. He showed that using fast *rectangular* matrix multiplication algorithms (current fastest rectangular matrix multiplication algorithm is due to Le Gall [10]), APSP for directed graphs with edges of weights $\{-M,\dots,M\}$ can be computed in $O(M^{0.68}n^{2.53})$ time. For undirected graphs with edges of small integers weights $\{1,\dots,M\}$, Galil and Margalit [9] showed an algorithm with a running time of $\widetilde{O}(M^{(\omega+1)/2}n^\omega)$. This result was later improved by Shoshan and Zwick [21] to $\widetilde{O}(Mn^\omega)$. Considering the case of undirected unweighted graphs, Seidel [20] obtained an algorithm for APSP that runs in $\widetilde{O}(n^\omega)$ time. The advantage of Seidel's $\widetilde{O}(n^\omega)$ time algorithm is that it is much simpler than that of [9].

Currently, for unweighted graphs, there is a large gap in the upper bounds for directed and undirected graphs. Many believe that $\omega = 2 + o(1)$, and if this is indeed the case, then the $\widetilde{O}(n^\omega)$ algorithms for undirected (e.g. Seidel's algorithm) match, up to logarithmic factors, the natural $O(n^2)$-time lower bound for the problem. On the other hand, considering this case for directed graphs, both the algorithm of Alon, Galil and Margalit, and the algorithm of Zwick run in $\widetilde{O}(n^{2.5})$ time. Moreover, the improved result of Zwick relies on the fact, that currently there exist faster algorithms for rectangular matrix multiplication (e.g. [10]) than square rectangular matrix multiplication. Actually, if only using square matrix multiplication, the over twenty years old result of Alon, Galil and Margalit is still the best we know so far. Also, currently, the only known way to achieve $\widetilde{O}(n^\omega)$-time APSP algorithm for directed graphs is to settle with an *approximation*: Zwick showed in [26] an algorithm that computes a $(1+\varepsilon)$ approximation to APSP in $\widetilde{O}(\frac{1}{\varepsilon}n^\omega \log \frac{1}{\varepsilon})$ time. This raises an interesting open question, whether an $\Omega(n^{2.5})$-time is the lower bound to compute APSP for directed graphs.

So what are the obstacles that prevent us from obtaining an $\widetilde{O}(n^\omega)$-time algorithm for directed unweighted graphs as well? Consider Seidel's algorithm. The idea of this algorithm

[1] The notation $\widetilde{O}$-notation suppress polylogarithmic factors from the O-notation.

is fairly simple: let $G = (V, E)$ be a directed graph with adjacency matrix A, and assume we have computed recursively the distance matrix for the graph G' induced by A^2. Recall that in G', an edge (u, v) exists if and only if there is a path of length at most 2 from u to v. This implies that either $d_G(u, v) = 2d_{G'}(u, v)$, if $d_G(u, v)$ is even, or $d_G(u, v) = 2d_{G'}(u, v) - 1$, otherwise. Since in each recursion the distances in the induced graph is cut by half, the depth of the recursion is $O(\log n)$. The only thing that is left in order to complete the algorithm, is to determine for every $u, v \in V$, if $d_G(u, v)$ is even or odd. Let $N(v)$ be the set of neighbors of v in G. For every $w \in N(v)$, by the triangle inequality for unweighted *undirected* graphs,

$$d(u, v) - 1 \leq d(u, w) \leq d(u, v) + 1$$

Using this property, it is not hard to verify that $\sum_{w\in N(v)} d_{G'}(u, w) \geq |N(v)| \cdot d_{G'}(u, v)$ when $d_G(u, v)$ is even and $\sum_{w\in N(v)} d_{G'}(u, w) < |N(v)| \cdot d_{G'}(u, v)$ otherwise. Since the sums $\sum_{w\in N(v)} d_{G'}(u, w)$, for every $u, v \in V$, can be computed at once using a single matrix multiplication, the total computation time of a recursion stage is $O(n^\omega)$ (for more details, see [20]). The sole obstacle that prevents us from implementing Seidel's algorithm for directed graphs is that the triangle inequality in *directed* unweighted graphs holds only for one side: for an incoming edge (w, v), it only holds that $d(u, v) - 1 \leq d(w, u)$ (similarly, for an outgoing edge (v, w), we have $d(u, w) \leq d(u, v) + 1$). For an incoming edge (w, v), we can get two-sides bounds for $d(u, w)$, if we have a path from v to w. Specifically, if for an incoming edge (w, v), it holds that $d(v, w) \leq d$ (for some d), then $d(u, w) \leq d(u, v) + d$ (similar argument can be applied for an outgoing edge as well). In other words, for any $u, v \in V$, the more an edge (w, v) is "symmetric" (i.e. $d(v, w)$ is close to 1), the better bounds we get from the triangle inequality on $d(u, w)$, using $d(u, v)$. As by definition, undirected graphs are fully-symmetric, we have that $d(v, u) = 1$ for every edge (u, v) in an undirected unweighted graph. On the other hand, in a directed graph, for an edge (w, v), it might be the case that the graph does not have any path at all from v to w, thus we cannot guarantee a two-sides bound by the triangle inequality for $d(u, w)$ as in unweighted graphs.

In this paper we introduce a new parameter for directed strongly-connected graphs that measures the closeness of a directed strongly-connected graph to symmetric graph (i.e. undirected). The symmetry parameter $s(G)$ of a directed strongly-connected graph $G = (V, E)$ is defined to be $\max_{(u,v)\in E}\{d(v, u)\}$. Interestingly, the definition of the symmetry parameter is very similar to the definitions of the *girth* and *diameter*, both natural and well-studied (e.g. [1, 14, 17, 18, 19]) distance-related graphs parameters: the girth of directed graphs is $\min_{(u,v)\in E}\{d(v, u)\}$, and the diameter of the graph is $\max_{(u,v)\in V\times V}\{d(u, v)\}$. As we shall see later, similarly to the girth and the diameter, we can also compute the symmetry parameter of the graph in $\widetilde{O}(n^\omega)$ time.

Our main contribution in this paper is an algorithm (Theorem 8) that provides a non-trivial generalization of Seidel's algorithm, and computes APSP for a directed strongly-connected graph G with symmetry parameter $s = s(G)$ in $O(s \cdot n^\omega \log_{s+1} n)$ time. The rough idea of the algorithm is as follows. As discussed above, using the symmetry parameter, we can obtain generalized triangle inequalities. In our algorithm we use these new inequalities, and adapt Seidel's algorithm to find all the pairs $(u, v) \in V \times V$, such that $d(u, v) \equiv 1 \mod (s + 1)$, and their distances. However, it is unclear how to deduce the distances for all other pairs of vertices, i.e., for all $(u, v) \in V \times V$, such that $d(u, v) \not\equiv 1 \mod (s + 1)$. Our approach to tackle this obstacle is to find, using one BMM, all the pairs $(u, v) \in V$ such that $d(u, v) \equiv i \mod (s + 1)$, based on that we already know all the pairs $(u, v) \in V$, such that $d(u, v) \equiv j \mod (s+1)$, for all $0 < j < i$. Surprisingly, here comes the parameter into action again. Using a designated matrix multiplication, the parameter allows us to find all the relevant pairs and their distances. Continuing this process inductively, we can compute the distances between all pairs of vertices. Our algorithm can also be applied on general directed unweighted graphs, by using an $\widetilde{O}(n^{\omega})$-time reduction from APSP for general directed unweighted graphs to APSP for directed strongly-connected unweighted graphs.

In the definition of the symmetry parameter, we are taking the maximum over all the edges, thus, even a single edge in a graph can cause the symmetry parameter of the graph to be very large. However, this can be relaxed by the concept of *violating-edge*. Let $z < n$ be some threshold value. An edge (u, v) is a z-violating edge if $d(v, u) > z$. By applying Breadth-First Search (BFS) in and out of the endpoints of all z-violating edges in the graph, we can remove these edges, and by that, decrease the symmetry parameter of the new graph to be as small as our threshold value z. Now, the distance from u to v in the original graph is either their distance in the new graph, or otherwise, equals to $d(u, x) + 1 + d(y, v)$, where (x, y) is some z-violating edge. Using this idea, for any graph that has at most $o(n^{0.53})$ violating edges for a $o(n^{0.157})$-threshold, we can compute APSP faster than the state of the art algorithm for the problem. Notice that a larger threshold is allowed as ω decreases. For example, if $\omega = 2 + o(1)$, then our APSP runs in $o(n^{2.5})$ time for any graph that has at most $o(n^{1/2})$ violating edges for any $o(n^{1/2})$-threshold. This may suggest the following strategy to compute APSP for directed unweighted graphs. First compute in $\widetilde{O}(n^{\omega})$ time (Lemma 13) the number of violating-edges for any $o(n^{0.157})$-threshold. If there are at most $o(n^{0.53})$ violating edges, use our algorithm, otherwise use the state of the art algorithm for the problem.

Another interesting property of our algorithm is that it provides an improvement over a basic parameterized-APSP algorithm for directed unweighted graphs, that is also used sometimes as an ingredient in other algorithms for the problem (e.g. in [2]). For a directed unweighted graph G with adjacency matrix A and *diameter* $D \leq n$, we can easily compute APSP in $O(D \cdot n^{\omega})$ by computing $A, A^2, A^3, \ldots, A^D$. For a large diameter, the running time of this algorithm can be as high as $O(n^{1+\omega})$ (which is even worse than Floyd-Warshall's algorithm), however, it can be turned to be useful for graphs that their diameter is less than $n^{2.5-\omega}$, in this case, the running time of this algorithm will be $o(n^{2.5})$. Our algorithms provide improvements over this basic algorithm in two ways. First they provide a weaker constraint on the graph, as $s(G) = \max_{(u,v)\in E}\{d(v, u)\} \leq \max_{(u,v)\in V\times V}\{d(u, v)\} = D$. Also, using the concept of z-violating edges, we can decrease the value of the symmetry parameter, while, to our best knowledge, there is no such equivalent method for the parameterized-APSP algorithm that is based on the diameter.

The paper is organized as follows. In the next section we provide some preliminaries for our algorithms. Section 3 presents our main algorithm. In Section 4 we show a reduction that allows us to compute APSP on general directed graphs using our parameterized algorithm. In Section 5 we give a hybrid algorithm that allows us to reduce the size of the symmetry parameter of the graph in exchange to additional BFS executions on the graph. We also show in this section how to compute in $\widetilde{O}(n^{\omega})$ time the symmetry parameter of a graph and the violating edges in the graph for some specific threshold.

2 Preliminaries

Let $G = (V, E)$ be a directed graph with n vertices and m edges. Let $u, v \in V$ and let $d_G(u, v)$ be the length of the shortest path from u to v in G. If there is no path from u to v in G, then $d_G(u, v) = \infty$. We use a simplified notation $d(u, v)$, when the referred graph is clear from the context. A directed graph is strongly-connected if for all $u, v \in V$, $d(u, v) < \infty$.

Our goal in this paper is to compute efficiently All Pairs Shortest Path (APSP) for directed unweighted graphs. The general form of the problem is defined as follows.

▶ **Definition 1** (The APSP Problem)**.** Let $G = (V, E)$. Compute a matrix M of size $|V| \times |V|$, such that for every $u, v \in V$, $M(u, v) = d(u, v)$.

The diameter of a graph G is $D(G) = \max_{u,v \in V}\{d(u, v)\}$. Notice that a directed graph G is strongly-connected if and only if $D(G) < \infty$. An extended definition of the diameter that includes non-necessarily strongly-connected graphs (with non-empty edges set) is $\max_{u,v \in V}\{d(u, v) \mid d(u, v) < \infty\}$. Let $v \in V$, $N_{in}(v)$ is the set of all vertices that have an outgoing edge to v, namely, $N_{in}(v) = \{u \mid (u, v) \in E\}$.

The *symmetry parameter* introduced here is a new notion that measures the closeness of a directed strongly-connected graph to a symmetric (undirected) graph. We define the symmetry parameter of an edge (u, v) as $d(v, u)$. The symmetry parameter of a graph is the maximum over the symmetry parameters of all the edges in the graph.

▶ **Definition 2** (Directed Graphs, Symmetry Parameter)**.** Let $G = (V, E)$ be a directed strongly-connected graph. $s(G) = \max_{(u,v) \in E}\{d(v, u)\}$.

Notice that in undirected graphs $s(G) = 1$. Since the definition of the symmetry parameter of the graph takes the maximum over all the edges, we may achieve a smaller value for the parameter if we exclude some of the edges in the graph. This idea is encapsulated in the definition of z-violating edges.

▶ **Definition 3** (z-violating Edges)**.** Let $G = (V, E)$ be a directed strongly-connected graph and let $z < s(G)$ be a threshold value. An edge $(u, v) \in E$ is z-violating if $d(v, u) > z$.

We denote by A_G, the adjacency matrix of a graph G. In A_G, it holds for every $u, v \in V$ that:

$$A_G(u, v) = \begin{cases} 1 & (u, v) \in E \vee u = v \\ 0 & (u, v) \notin E \end{cases}$$

Note that for undirected graphs the adjacency matrix is symmetric.

Let A and B be two $n \times n$ matrices with integral values. We denote $A \cdot B$ as the integer multiplication of A and B. We use the same notation, when A and B are Boolean matrices and the operation is Boolean Matrix Multiplication (BMM).We can multiply two integer/Boolean metrics in $O(n^\omega)$ time, where currently $\omega < 2.373$ [10].

We use the notation A^k ($k > 0$) for the multiplication of A by itself k times. For the rest of the paper, when using A^k, we assume the multiplication that is taken is BMM. Naïvely, we can compute A^k in $O(k \cdot n^\omega)$, however this can be done in $O(n^\omega \log n)$ time using "repeated squaring" method (e.g. [14]). The matrix that is obtained from multiplying the adjacency matrix of a graph by itself is meaningful in respect to the distances of the graph. This relation is given in the following proposition.

▶ **Proposition 4** (Distances vs. the Adjacency Matrix [14])**.** *Let* $D_k = (A_G)^k$. $D_k(u, v) = 1$ *if and only if* $d(u, v) \le k$.

Denote G_k to be the graph induced from the adjacency matrix D_k. The graph G_k is the graph G with some additional new edges. Specifically, G_k has a new edge (u, v), if $(u, v) \notin G$ and there exists a shortest path in G from u to v of length at most k. In other words, in G_k we add "shortcuts edges" to paths of length at most k in G. This leads us to the following definition: let $k > 1$ and $u, v \in V$, we denote r_{uv} to be ($d_G(u, v) \mod k$), that is, the residues of $d(u, v)/k$.

3 Parameterized Algorithm for APSP

In this section we show in Theorem 8 how to compute APSP for a directed strongly-connected graph G in $O(s \cdot n^{\omega} \log_{s+1} n)$ time, where $s = s(G)$ is the symmetry parameter of G. Before we turn to prove the theorem, we give some lemmas that are needed for our theorem.

In undirected graphs, for vertices u, v and a neighbor w of v, using the triangle inequality, we have $d(u, v) - 1 \leq d(u, w) \leq d(u, v) + 1$. Considering the same situation in directed graphs, however, we can only guarantee either that $d(u, v) - 1 \leq d(u, w)$, where (w, v) is an incoming edge of v, or that $d(u, w) \leq d(u, v) + 1$, where (v, w) is an outgoing edge of v. With the symmetry parameter we can obtain both the following lower and upper bounds on $d(u, w)$.

▶ **Lemma 5.** *Let $G = (V, E)$ be a directed, strongly-connected, unweighted graph. Then:*

1. *For every $u, v \in V$ and every $w \in N_{in}(v)$, it follows that $d(u, v) - 1 \leq d(u, w) \leq d(u, v) + s(G)$.*
2. *For every $u, v \in V$ $(u \neq v)$ there exists a $w \in N_{in}(v)$ such that $d(u, w) = d(u, v) - 1$.*

Proof. Let $u, v \in V$. If $w \in N_{in}(v)$, then $d(w, v) = 1$, and from the definition of the symmetry parameter $d(v, w) \leq s(G)$. From the triangle inequality on the pair (u, v), we get that $d(u, v) \leq d(u, w) + d(w, v) \leq d(u, w) + 1$, that is, $d(u, v) - 1 \leq d(u, w)$. From the triangle inequality on the pair (u, w), we get that $d(u, w) \leq d(u, v) + d(v, w) \leq d(u, v) + s(G)$. This proves the first part of the lemma.

For the second part, consider a shortest path P from u to v, and let w be the vertex that immediately preceding v on P. Since $u \neq v$, we have that $w \neq v$, and therefore $w \in N_{in}(v)$. Since w is on a shortest path from u to v, $d(u, v) = d(u, w) + 1$, and thus $d(u, w) = d(u, v) - 1$. ◀

The next lemma shows the relation between the distances in the original graph G and the graph that is induced by $A' = (A_G)^x$.

▶ **Lemma 6.** *Let $G = (V, E)$ be a directed graph and let $0 < x \leq n$ be an integer. Let G' be the graph induced from $A' = (A_G)^x$. Let $r_{uv} = d_G(u, v) \mod x$, for every $u, v \in V$ it follows that:*

1. *$d_{G'}(u, v) = \lceil \frac{d_G(u,v)}{x} \rceil$.*
2. *If $r_{uv} = 0$: $d_G(u, v) = x \cdot d_{G'}(u, v)$.*
3. *If $r_{uv} \neq 0$: $d_G(u, v) = x \cdot (d_{G'}(u, v) - 1) + r_{uv}$.*

Proof. Let $u, v \in V$, such that $d = d_G(u, v)$. Notice that $d_G(u, v) = x \cdot \lfloor \frac{d_G(u,v)}{x} \rfloor + r_{uv}$. Proposition 4 implies that $d_G(u, v) \leq x \cdot d_{G'}(u, v)$. Notice first that it cannot be $d_{G'}(u, v) < \lceil \frac{d_G(u,v)}{x} \rceil$, as otherwise we get $d_G(u, v) \leq x \cdot (\lceil \frac{d_G(u,v)}{x} \rceil - 1) < \lfloor \frac{d_G(u,v)}{x} \rfloor + r_{uv}$. Denote $b = \lfloor \frac{d_G(u,v)}{x} \rfloor$. Let $P = \langle u = v_0, \ldots, v_d = v \rangle$ be a shortest path from u to v in G. From Proposition 4 we have for every $0 \leq i \leq b$ the edges $(v_{i \cdot x}, v_{(i+1) \cdot x})$ are in G'. Also, if $r_{uv} \neq 0$, we have the edge $(v_{b \cdot x}, v_d)$, and otherwise (i.e. when $r_{uv} = 0$) we have $v_{b \cdot x} = v_d$. In both cases, we get that there exists a path from u to v in G' of length $\lceil \frac{d_G(u,v)}{x} \rceil$. Claims (2) and (3) of this lemma follow immediately from the construction of the path in G'. ◀

Lemma 7 guarantees that the symmetry parameter of the graph induced by $A' = (A_G)^x$ is not bigger than $s(G)$.

▶ **Lemma 7.** *Let $G = (V, E)$ be a directed strongly-connected graph and let $0 < x \leq n$ be an integer. Let G' be the graph induced from $A' = (A_G)^x$. It follows that G' is also strongly-connected and $s(G') \leq s(G)$.*

Proof. First note that since G is strongly-connected and we have from Lemma 6 that $d_{G'}(u,v) = \lceil \frac{d_G(u,v)}{x} \rceil$ for every $u, v \in V$, it follows that G' is also strongly-connected. Let (u, v) be an edge in G'. From the way we obtained G', $d_G(u,v) \leq x \cdot d_{G'}(u,v) = 1 \cdot x = x$. Let $P = \langle u = v_0, \ldots v_d = v \rangle$ be a shortest path from u to v in G. By the symmetry parameter of G, there exists a path in G of length $s(G)$ at most from every (v_{i+1}, v_i), $0 \leq i \leq d_G(u,v)$. This implies a shortest path in G from v to u of length $x \cdot s(G)$ at most. By Lemma 6, $d_{G'}(v,u) = \lceil \frac{d_G(v,u)}{x} \rceil \leq \lceil \frac{x \cdot s(G)}{x} \rceil = \lceil s(G) \rceil = s(G)$, as required. ◀

We now turn to present our main theorem. Notice that the algorithm receives as an input the parameter $s(G)$ of G. As we shall show later, $s(G)$ can be computed in $O(n^\omega \log n)$ time. A pseudocode of our algorithm is given in Algorithm 2.

▶ **Theorem 8.** *Let $G = (V, E)$ be a directed strongly-connected unweighted graph with a parameter $s = s(G)$. We can compute APSP for G in $O(s \cdot n^\omega \log_{s+1} n)$ time.*

Proof. Notice first that $s = s(G)$ is well defined, since G is strongly-connected. Let $s' = s+1$. Our algorithm will follow a similar approach as done in Seidel's algorithm [20] for undirected graphs.

Let A be the adjacency matrix of the graph G (recall we assume that A always has 1's in its diagonal). If all entries in A are 1's, we return A as the distance matrix for G. If this is not the case, we compute $A' = A^{s'}$ in $O(s' \cdot n^\omega)$ time. Let G' be the graph of A' and let D' be the distance matrix for G', that is, $D'(u,v) = d_{G'}(u,v)$. We obtain D' by invoking our algorithm recursively. Notice that according to Lemma 7 the parameter $s(G)$ does not increase in G' (i.e. $s(G') \leq s(G)$).

Our goal is to compute D, the distance matrix of G, based on D', the distance matrix computed recursively for G'. According to Lemma 6, for every $u, v \in V$ it follows that $d_{G'}(u,v) = \lceil \frac{d_G(u,v)}{s'} \rceil$ and $d_G(u,v) = s' \cdot (d_{G'}(u,v) - 1) + r_{uv}$ (or $d_G(u,v) = s' \cdot d_{G'}(u,v)$, for the case that $r_{uv} = 0$). Since we know $d_{G'}(u,v)$, this implies we only left to compute r_{uv} in order to find the distance from u to v in G.

Our first step is to compute the distances for all $u, v \in V$ such that $d_G(u,v) = d_{G'}(u,v) + 1$ (i.e. $r_{uv} = 1$).

Let $u, v \in V$. Denote $k = d_{G'}(u,v)$. Examine first the case where $r_{uv} = 1$. Let $w \in N_{in}(v)$. It follows from Lemma 5 that there exists a $w' \in N_{in}(v)$ such that $d_G(u,w') = d_G(u,v) - 1$. Since $r_{uv} = 1$, it must be that $d_{G'}(u,w') = k - 1$, as $r_{uw'} = 0$ and $d_{G'}(u,w') = \lceil \frac{d(u,w')}{s'} \rceil$. From Lemma 5 and from the symmetry parameter s of G it follows that for every $w \in N_{in}(v)$, $d_G(u,w) \leq d_G(u,v) + s$. Therefore, $d_{G'}(u,w) \leq k$, and for $r_{uv} = 1$, we have that $\sum_{w \in N_{in}(v)} d_{G'}(u,w) < |N_{in}(v)|k = |N_{in}(v)| d_{G'}(u,v)$.

For the case that $r_{uv} \neq 1$. We know from Lemma 5 that for every $w \in N_{in}(v)$ we have $d_G(u,v) - 1 \leq d_G(u,w) \leq d_G(u,v) + s$. Since for every $w' \in N_{in}(v)$, such that $d_G(u,w) = d_G(u,v) - 1$, it holds that $d_{G'}(u,w') = k$, it implies that $d_{G'}(u,w) \geq k$ for every $w \in N_{in}(v)$ and we have in this case that $\sum_{w \in N_{in}(v)} d_{G'}(u,w) \geq |N_{in}(v)|k = |N_{in}(v)| d_{G'}(u,v)$.

We conclude that for every $u, v \in V$, $r_{uv} = 1$ if and only if $\sum_{w \in N_{in}(v)} d_{G'}(u,w) < |N_{in}(v)| d_{G'}(u,v)$. We can obtain $\sum_{w \in N_{in}(v)} d_{G'}(u,w)$, for every $u, v \in V$, by computing the integer matrix multiplication $D_1 = D' \cdot A$ in $O(n^\omega)$ time. By computing the integer matrix multiplication $1^{n \times n} \cdot A$, we can obtain $|N_{in}(v)|$ for every $v \in V$. Overall we can check for every $u, v \in V$ if $\sum_{w \in N_{in}(v)} d_{G'}(u,w) < |N_{in}(v)| d_{G'}(u,v)$ in $O(1)$ time. At this stage we set $D(u,v) = s'(D'(u,v) - 1) + 1$, for every $u, v \in V$ such that $r_{uv} = 1$.

It is left to compute the distances for every $u, v \in V$ such that $r_{uv} \neq 1$. We show by induction for $1 \leq i \leq s'$, that we can compute the distances for every $u, v \in V$ such that $r_{uv} = i \mod s'$ in $O(n^\omega)$ time. We already showed the base case of the induction (i.e. for

$i = r_{uv} = 1$). By the hypothesis of the induction, we assume that we have the distances for every $u, v \in V$ such that $r_{uv} = j \mod s'$, $1 \leq j \leq i$, and we show that we can compute the distances for $u, v \in V$ such that $r_{uv} = (i+1) \mod s'$.

Let $N^i_{uv} = \{w \in N_{in}(v) \mid r_{uw} = i \mod s'\}$.

▶ **Claim 9.** *Let $u, v \in V$ ($u \neq v$), such that $d_{G'}(u,v) = k$. For every $w \in N^i_{uv}$, $d_{G'}(u,w)$ is either $k-1$, k or $k+1$. Moreover, if $r_{uv} \neq 1$, it cannot be that $d_{G'}(u,w) = k-1$.*

Proof. By Lemma 5, $d_G(u,v) - 1 \leq d_G(u,w)$. Now, $d_{G'}(u,w) = \lceil \frac{d_G(u,w)}{s'} \rceil \geq \lceil \frac{d_G(u,v)-1}{s'} \rceil \geq \lceil \frac{d_G(u,v)}{s'} \rceil - 1 = d_{G'}(u,v) - 1 = k-1$. Notice that the only case that $\lceil \frac{d_G(u,v)-1}{s'} \rceil = \lceil \frac{d_G(u,v)}{s'} \rceil - 1$ is when $r_{uv} = 1$. Therefore, $d_{G'}(u,w) \geq k-1$, and $d_{G'}(u,w) \geq k$ for $r_{uv} \neq 1$.

Assume toward contradiction that there exists $w \in N^i_{uv}$ such that $d_{G'}(u,w) > k+1$. For such a w, we have $d_G(u,w) > s'(k+1)$. Now, since $(w,v) \in E$ and the symmetry parameter of G is s, it follows that $d_G(v,w) \leq s$, but then $d_G(u,v) + d_G(v,w) \leq s'k + s' = s'(k+1) < d_G(u,w)$, a contradiction to the minimality of $d_G(u,w)$. ◀

Using Claim 9 we can now provide a criteria we will use to identify every $u, v \in V$ ($u \neq v$) such that $r_{uv} = (i+1) \mod s'$.

▶ **Claim 10.** *Let $u, v \in V$ ($u \neq v$), such that $r_{uv} = j \mod s'$. Then:*

1. *If $j = i+1 \Rightarrow \sum_{w \in N^i_{uv}} d_{G'}(u,w) < |N^i_{uv}| \cdot (d_{G'}(u,v)+1)$*
2. *If $i+2 \leq j \leq s' \Rightarrow \sum_{w \in N^i_{uv}} d_{G'}(u,w) \geq |N^i_{uv}| \cdot (d_{G'}(u,v)+1)$*

Proof. Denote $d_{G'}(u,v) = k$. Since the claim only considers $j \geq 2$, we have that $r_{uv} \neq 1$. Now, according to Claim 9 it holds that $d_{G'}(u,w) = k$ or $d_{G'}(u,w) = k+1$ for every $w \in N^i_{uv}$. If $j = i+1$ then there exists a $w' \in N^i_{uv}$ with $d_{G'}(u,w') = k$. Therefore, $\sum_{w \in N^i_{uv}} d_{G'}(u,w) < |N^i_{uv}| \cdot (k+1) = |N^i_{uv}| \cdot (d_{G'}(u,v)+1)$.

If $j \geq i+2$ we cannot have any $w \in N^i_{uv}$ such that $d_{G'}(u,w) = k$, since otherwise we get that $d_G(u,w) + d_G(w,v) = (s' \cdot (k-1) + i) + 1 < s' \cdot (k-1) + (i+2) \leq d_G(u,v)$, a contradiction to the minimality of $d_G(u,v)$. Therefore, $\sum_{w \in W^i_{uv}} (d_{G'}(u,v)+1) = |N^i_{uv}| \cdot (k+1) = |N^i_{uv}| \cdot (d_{G'}(u,v)+1)$. ◀

By Claim 10 we can now identify $r_{uv} = j \mod s'$, $j = i+1$ by first checking whether $\sum_{w \in N^i_{uv}} D'_G(u,w) < |N^i_{uv}| \cdot (D'_G(u,v)+1)$ is satisfied or not. Even if the inequality holds, it still may be that $j < i+1$, but since by the induction hypothesis we already computed the distances for $j \leq i$, we can distinguish between $j = i+1$ and $j < i+1$. We compute the exact distances for $r_{uv} = (i+1) \mod s'$ using Lemma 6. We are left to show how to compute $\sum_{w \in N^i_{uv}} D'_G(u,w)$ and $|N^i_{uv}|$. Notice that at this stage we know the exact distance for every $u, v \in V$ such that $r_{uv} = i \mod s'$. We define $D'_i(u,v)$ to be $D'(u,v)$ if $r_{uv} = i \mod s'$ and 0 otherwise. Similarly, we define $A_i(u,v)$ to be 1 if $r_{uv} = i \mod s'$ and 0 otherwise. We compute by integer matrix multiplications $D_i = D'_i \cdot A$ and $N_i = A_i \cdot A$. Notice that $D_i(u,v) = \sum_{w \in N^i_{uv}} D'_G(u,w)$ and that $N_i(u,v) = |N^i_{uv}|$. This concludes the correctness of the algorithm.

By Lemma 12 the symmetry parameter of the graph can be obtained in $O(n^\omega \log n)$ time. The time to compute $A^{s'}$ is $O(s \cdot n^\omega)$. The time to compute a specific $r_{uv} = i \mod s'$ ($1 \leq i \leq s'$) is also $O(n^\omega)$, therefore, the total running time for a recursive invocation is $(s'+1) \cdot O(n^\omega)$. Since G is an unweighted graph, its diameter is at most $n-1$, and hence we have at most $O(\log_{s'} n)$ recursive invocations. The total time of the algorithm, therefore, is $O(s \cdot n^\omega \log_{s'} n)$. ◀

4 Extending to general directed graphs

As noted earlier, Algorithm 2 works only on strongly-connected graphs, where $s(G)$ is well-defined. Nevertheless, as we shall see here, it is possible to reduce any directed unweighted graph to a directed strongly-connected unweighted graph, where our parameterized-APSP can be applied. Let $G = (V, E)$ be our graph. Let $Diam(G) = \max_{u,v \in V} \{d_G(u, v) \mid d_G(u, v) < \infty\}$ be the diameter of the graph G. In the reduced graph G', we will have $s(G') \leq Diam(G) + 1$.

The reduction is done as follows. Let $G = (V, E)$, and let $d = Diam(G)$. First we build a new directed graph $G' = (V', E')$. The graph G' is initially a copy of G. Next, we add to G' new vertices $v_1, v_2, \dots, v_d$, and the edges (v_i, v_{i+1}), where $1 \leq i \leq d - 1$, and the edge (v_d, v_1). We also add for each vertex $v \in V$ the edges (v, v_1) and (v_d, v).

We can apply on G' our parameterized APSP of Theorem 8.

From our construction $s(G') \leq d + 1$. To see that, first notice that the new vertices $v_1, v_2, \dots, v_d$ are connected by a cycle of length $d - 1$, therefore, for the edges (v_i, v_{i+1}), where $1 \leq i \leq d - 1$, and the edge (v_d, v_1) the symmetry parameter is $d - 2$. Let $v \in V$, the edges (v, v_1) and (v_d, v) are contained in the cycle $v \to v_1 \to v_2 \to \cdots \to v_d \to v$, thus the symmetry parameter of the edges (v, v_1) and (v_d, v) for every $v \in V$ is d. Also, for an edge $(u, v) \in E$ we have the path $\langle v, v_1, v_2, \dots, v_d, u \rangle$ of length $d + 1$. The insertion of the new edges to G may shorten the distance between vertices that were originally in G. However, it is easy to see that any path in G' between two vertices from G, that uses at least one of the new edges, must has a length of at least $d + 1$. Since the original diameter of G is d, this implies that for every $u, v \in V$, if $d_{G'}(u, v) \leq d$ then $d_{G'}(u, v) = d_G(u, v)$, otherwise, if $d_{G'}(u, v) > d$ then $d_G(u, v) = \infty$.

The diameter can be computed, using repeated squaring, in $O(n^\omega \log n)$ time. The time to construct G' and to compute the distances of G from the distances of G' is bound by $O(n^2)$. Thus the total time of the reduction is $O(n^\omega \log n)$.

Any algorithm that computes APSP must take $\Omega(n^\omega)$ time (otherwise matrix multiplication can be computed faster), therefore, using the reduction above, we can convert any algorithm that computes APSP for strongly-connected directed graphs in $T(n)$ time into algorithm that computes $T(2n) + O(n^\omega \log(n)) = O(T(n) \log(n))$ time general directed graphs.

5 A hybrid APSP algorithm for directed graphs

As shown, the efficiency of our parameterized-APSP algorithm strongly depends on the symmetry parameter of the input graph. Since by the definition of the parameter, the *maximum* is taken over all the edges of the graph, even a single edge might cause the parameter of the graph to be large.

Recall that for a threshold value $z < s(G)$ of a directed strongly-connected graph G, an edge (u, v) is z-violating edge, if $d(v, u) > z$. We now show that our algorithm can be easily modified such that even if a graph holds to have $o(n^{0.53})$ edges that are $o(n^{0.16})$-violating, the modified algorithm provides a faster algorithm than the state-of-the-art algorithm for the problem [26].

Let $\beta_z(G)$ be the number of z-violating edges in G. The following theorem gives an algorithm to compute APSP for G in a time that depends on z and $\beta_z(G)$. A pseudocode of the algorithm is also given in Algorithm 3.

▶ **Theorem 11.** *Let $G = (V, E)$ be a directed, strongly-connected graph, with a parameter $s(G)$, and let $z < s(G)$ be an integer. We can compute APSP for G in $O(zn^\omega \log n + n^2\beta_z(G))$ time.*

Proof. Let $E_z(G)$ be the set of edges in G that are z-violating. We first compute $E_z(G)$ in $O(n^\omega \log z)$ time, using Lemma 13. Let $(u, v) \in E_z(G)$, we compute in $O(m) = O(n^2)$ time BFS in and out of u and v. Next we removed from G all the edges in $E_z(G)$. Let H be the new graph after the removal of the edges. For the sake of simplicity we assume H is strongly-connected (otherwise, we can run the parameterized APSP algorithm on each of its strongly-connected components). Notice that $s(H) < z$, since we remove all the edges that violate the threshold, and any non-violating edge must participates in a cycle such that all its edges are non-violating edges (thus removing the violating edges would not affect a non-violating edge). Therefore, we compute in $O(zn^\omega \log n)$ time the distances of H. If a shortest path from u to v in G contains an edge (p, q) from $E_z(G)$, we can find the distance in $|E_z(G)| = \beta_z(G)$ time by taking the minimum from $min_{(p,q)\in E_z(G)}\{d(u, p) + d(q, v) + 1\}$ (the distances $d(u, p)$ and $d(q, v)$ are obtained from the in and out BFS computed for all the endpoints in $E_z(G)$). Otherwise, $d(u, v) = d_H(u, v)$. The time for this step is $O(n^2\beta_z(G))$ and the total time for this algorithm is $O(zn^\omega \log n + n^2\beta_z(G))$. ◀

Since the reduction to strongly-connected graph and the computation of the z-violating edges takes $O(n^\omega \log n)$ time, this leads us to the following hybrid approach. Reduce the graph to strongly-connected and compute the violating edges for z such that it gives faster running time than the current known fastest algorithm for the problem. If the number of violating edges does not exceed the threshold, use Algorithm 3, otherwise use the fastest known algorithm for the problem. Unless an $O(n^\omega \log n)$-time algorithm for the problem is found, this hybrid approach may provide a faster running time for some instances of the problem.

5.1 Fast computation of the parameter and the violating edges

Our algorithms need to know the symmetry parameter of the graph. We show in the next lemma how to compute this parameter in $O(n^\omega \log n)$ time.

▶ **Lemma 12.** *Let $G = (V, E)$ be a directed strongly-connected graph with symmetry parameter $s(G)$. We can compute $s(G)$ in $O(n^\omega \log n)$ time.*

Proof. Denote $D_G = (A_G \bigvee I)$. Compute $D_G, D_G^2, D_G^4, \ldots, D_G^{2^{\log n}}$ using repeated squaring. Let k be the first value in $\{0, \ldots, \lceil \log n \rceil\}$ such that $D_G^{2^k}(v, u) = 1$ for every $(u, v) \in E$. If $k = 0$, then $s(G) = 1$. Otherwise, by Proposition 4 it must be that $s(G) \in [2^{k-1}, 2^k]$. By setting $s'(G) = 2^k$, we have a 2-approximation for $s(G)$. We need only $\lceil \log n \rceil$ matrix multiplication to compute $D_G, D_G^2, D_G^4, \ldots, D_G^{2^{\log n}}$, thus the time for this is $O(n^\omega \log n)$.

Now, we show how to compute $s(G)$ exactly. We have $s(G) \in [2^{k-1}, 2^k]$, we perform a binary search for $s(G)$ in the range $[2^{k-1}, 2^k]$. Let $[a, b]$ $(a < b)$ be the current range, we compute D_G^c for $c = \frac{a+b}{2}$. We check if $D_G^c(v, u) = 1$ for every $(u, v) \in E$. If it is true, we continue the search in the range $[a, c]$, otherwise we continue the search in the range $[c, b]$. There are $O(\log n)$ steps in the binary search. We can compute D_G^c by using only 2 matrix multiplications. To see this, notice first, that since we do our binary search on $[2^{k-1}, 2^k]$, it follows, that at any stage of the binary search, $(b - a)/2$ is a power of 2. Since $c = \frac{a+b}{2} = a + \frac{(b-a)}{2}$, we have that $D_G^c = D_G^a \cdot D_G^{2^j}$ (for some $j \leq k$), where D_G^a and $D_G^{2^j}$ were already computed. This concludes that the total time to compute $s(G)$ exactly is $O(n^\omega \log n)$. ◀

Similarly, we find all z-violating edges for a $z < n$ in time $O(n^\omega \log z)$, by computing D^z_G and returning all the edges $(u, v) \in E$ such that $D^z_G(v, u) = 0$. The statement is given in the following lemma.

▶ **Lemma 13.** *Let $G = (V, E)$ be a directed strongly-connected graph, and let $z < n$ be an integral value. We can find all z-violating edges for a $z < n$ in $O(n^\omega \log z)$ time.*

References

1 Donald Aingworth, Chandra Chekuri, Piotr Indyk, and Rajeev Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM J. Comput.*, 28(4):1167–1181, 1999. doi:10.1137/S0097539796303421.

2 Noga Alon, Zvi Galil, and Oded Margalit. On the exponent of the all pairs shortest path problem. In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 569–575, 1991.

3 Timothy M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. *SIAM J. Comput.*, 39(5):2075–2089, 2010. doi:10.1137/08071990X.

4 Timothy M. Chan and Ryan Williams. Deterministic apsp, orthogonal vectors, and more: Quickly derandomizing razborov-smolensky. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1246–1255, 2016. doi:10.1137/1.9781611974331.ch87.

5 Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9(3):251–280, 1990. doi:10.1016/S0747-7171(08)80013-2.

6 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms (3. ed.)*. MIT Press, 2009. URL: http://mitpress.mit.edu/books/introduction-algorithms.

7 Michael L. Fredman. New bounds on the complexity of the shortest path problem. *SIAM J. Comput.*, 5(1):83–89, 1976. doi:10.1137/0205006.

8 Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987. doi:10.1145/28869.28874.

9 Zvi Galil and Oded Margalit. All pairs shortest distances for graphs with small integer length edges. *Inf. Comput.*, 134(2):103–139, 1997. doi:10.1006/inco.1997.2620.

10 François Le Gall. Faster algorithms for rectangular matrix multiplication. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 514–523, 2012. doi:10.1109/FOCS.2012.80.

11 François Le Gall. Powers of tensors and fast matrix multiplication. In *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*, pages 296–303, 2014. doi:10.1145/2608628.2608664.

12 Yijie Han. Improved algorithm for all pairs shortest paths. *Inf. Process. Lett.*, 91(5):245–250, 2004. doi:10.1016/j.ipl.2004.05.006.

13 Yijie Han. An $O\ (\ n\ ^3(\log\log\ n\ /\log\ n\ )^{5/4})$ time algorithm for all pairs shortest path. *Algorithmica*, 51(4):428–434, 2008. doi:10.1007/s00453-007-9063-0.

14 Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. *SIAM J. Comput.*, 7(4):413–423, 1978. doi:10.1137/0207033.

15 Donald B. Johnson. Efficient algorithms for shortest paths in sparse networks. *J. ACM*, 24(1):1–13, 1977. doi:10.1145/321992.321993.

16 Seth Pettie. A new approach to all-pairs shortest paths on real-weighted graphs. *Theor. Comput. Sci.*, 312(1):47–74, 2004. doi:10.1016/S0304-3975(03)00402-X.

17 Liam Roditty and Roei Tov. Approximating the girth. *ACM Transactions on Algorithms*, 9(2):15, 2013. doi:10.1145/2438645.2438647.

18 Liam Roditty and Virginia Vassilevska Williams. Minimum weight cycles and triangles: Equivalences and algorithms. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 180–189, 2011. doi:10.1109/FOCS.2011.27.

19 Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 515–524, 2013. doi:10.1145/2488608.2488673.

20 Raimund Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *J. Comput. Syst. Sci.*, 51(3):400–403, 1995. doi:10.1006/jcss.1995.1078.

21 Avi Shoshan and Uri Zwick. All pairs shortest paths in undirected graphs with integer weights. In *40th Annual Symposium on Foundations of Computer Science, FOCS'99, 17-18 October, 1999, New York, NY, USA*, pages 605–615, 1999. doi:10.1109/SFFCS.1999.814635.

22 Tadao Takaoka. Subcubic cost algorithms for the all pairs shortest path problem. *Algorithmica*, 20(3):309–318, 1998. doi:10.1007/PL00009198.

23 Tadao Takaoka. A faster algorithm for the all-pairs shortest path problem and its application. In *Computing and Combinatorics, 10th Annual International Conference, COCOON 2004, Jeju Island, Korea, August 17-20, 2004, Proceedings*, pages 278–289, 2004. doi:10.1007/978-3-540-27798-9_31.

24 Mikkel Thorup. Floats, integers, and single source shortest paths. *J. Algorithms*, 35(2):189–201, 2000. doi:10.1006/jagm.2000.1080.

25 Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 645–654, 2010. doi:10.1109/FOCS.2010.67.

26 Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, 2002. doi:10.1145/567112.567114.

27 Uri Zwick. A slightly improved sub-cubic algorithm for the all pairs shortest paths problem with real edge lengths. In *Algorithms and Computation, 15th International Symposium, ISAAC 2004, Hong Kong, China, December 20-22, 2004, Proceedings*, pages 921–932, 2004. doi:10.1007/978-3-540-30551-4_78.

A Algorithms

Algorithm 1: Directed-APSP($G = (V, E)$)

```
if $G$ is not strongly-connected then
    Reduce $G$ to a strongly-connected graph $G'$;
    $G \leftarrow G'$;
Compute $s = s(G)$;
return Param-Directed-APSP($A_G, s$);
```

Algorithm 2: Param-Directed-APSP(A, s)

```
$A \leftarrow A \vee I$;
if $A = 1^{n \times n}$ then
    return $A$
$D' \leftarrow$ Param-Directed-APSP($A^{s+1}, s$);
$D_1 \leftarrow D' \cdot A$;
foreach $(u, v) \in V^2$ do  $D(u, v) \leftarrow -\infty$ ;
foreach $(u, v) \in V^2$ do
    if $D_1(u, v) < |N_{in}(v)| \cdot D'(u, v)$ then
        $D(u, v) \leftarrow (s + 1)(D'(u, v) - 1) + 1$;
foreach $i \in \{2, \dots, s + 1\}$ do
    foreach $(u, v) \in V^2$ do
        if $D(u, v) = (s + 1)(D'(u, v) - 1) + i - 1$ then
            $D'_i(u, v) \leftarrow D'(u, v)$; $A_i(u, v) \leftarrow 1$;
        else
            $D'_i(u, v) \leftarrow 0$; $A_i(u, v) \leftarrow 0$;
    $D_i \leftarrow D'_i \cdot A$; $N_i \leftarrow A_i \cdot A$;
    foreach $(u, v) \in V^2$ do
        if $D_i(u, v) < |N_i(u, v)| \cdot D'(u, v) \wedge D(u, v) = -\infty$ then
            $D(u, v) = (s + 1)(D'(u, v) - 1) + i$;
return D
```

Algorithm 3: Directed-APSP-z($G = (V, E)$, z)

```
Compute $E_z(G)$, the $z$-violating edges of $G$;
foreach $(u, v) \in E_z(G)$ do
    Compute in and out BFS for $u$ and $v$, and store the obtained distances;
Let $H = (V, E \setminus E_z(G))$ ;
$D_H \leftarrow$ Param-Directed-APSP($A_H, s(H)$);
foreach $(u, v) \in V^2$ do
    $D(u, v) = min\{D_H(u, v), min_{(p,q) \in E_z(G)}\{d(u, p) + d(q, v) + 1\}\}$;
return $D$;
```

Online Budgeted Maximum Coverage

Dror Rawitz[*1] **and Adi Rosén**[†2]

1 **Faculty of Engineering, Bar-Ilan University, Ramat Gan, Israel**
`dror.rawitz@biu.ac.il`

2 **CNRS and Université Paris Diderot, Paris, France**
`adiro@liafa.univ-paris-diderot.fr`

Abstract

We study the ONLINE BUDGETED MAXIMUM COVERAGE (OBMC) problem. Subsets of a weighted ground set U arrive one by one, where each set has a cost. The online algorithm has to select a collection of sets, under the constraint that their cost is at most a given *budget*. Upon arrival of a set the algorithm must decide whether to accept or to reject the arriving set, and it may also drop previously accepted sets (preemption). Rejecting or dropping a set is irrevocable. The goal is to maximize the total weight of the elements covered by the sets in the chosen collection.

We present a deterministic $\frac{4}{1-r}$-competitive algorithm for OBMC, where r is the maximum ratio between the cost of a set and the total budget. Building on that algorithm, we then present a randomized $O(1)$-competitive algorithm for OBMC. On the other hand, we show that the competitive ratio of any deterministic online algorithm is $\Omega(\frac{1}{\sqrt{1-r}})$.

We also give a deterministic $O(\Delta)$-competitive algorithm, where Δ is the maximum weight of a set (given that the minimum element weight is 1), and if the total weight of all elements, $w(U)$, is known in advance, we show that a slight modification of that algorithm is $O(\min\{\Delta, \sqrt{w(U)}\})$-competitive. A matching lower bound of $\Omega(\min\{\Delta, \sqrt{w(U)}\})$ is also given.

Previous to the present work, only the unit cost version of OBMC was studied under the online setting, giving a 4-competitive algorithm [36]. Finally, our results, including the lower bounds, apply to REMOVABLE ONLINE KNAPSACK which is the preemptive version of the ONLINE KNAPSACK problem.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases budgeted coverage, maximum coverage, online algorithms, competitive analysis, removable online knapsack

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.73

1 Introduction

The BUDGETED MAXIMUM COVERAGE problem (abbreviated BMC) is the dual of the classical SET COVER problem. In SET COVER the input consists of a collection of sets $\mathcal{S} = \{S_1, \ldots, S_m\}$ over the ground set $U = \{u_1, \ldots, u_n\}$ with cost for each set, and the goal is to find a sub-collection of sets of minimal cost, whose union covers all the elements in the ground set. In BMC we are, in addition, given weights for the elements and a budget cap B. The goal is to find a subcollection of the sets that maximizes the total weight of the union

[*] Supported in part by a grant from the Israeli Ministry of Science, Technology, and Space, Israel (French-Israeli project Maimonide 31768XL) and by the Israel Science Foundation (grant No. 497/14).
[†] Research supported in part by ANR project NeTOC, and by a French-Israeli grant PHC Maimonide 31768XL.

of those sets, under the constraint that the total cost of the subcollection is at most B. In other words, the goal is to find a subcollection that maximizes coverage given a knapsack constraint. In fact the KNAPSACK problem can be viewed as the special case of BMC in which the sets are pairwise disjoint.

In the ONLINE BUDGETED MAXIMUM COVERAGE problem (OBMC) sets arrive online and the goal of an online algorithm for this problem is to find (in an online manner) a sub-collection of the sets, that maximizes the weight of the covered items, while adhering to the budget constraint. Preemption of previously used sets is allowed, but a preempted (or rejected at arrival) set cannot be used again later. Preemption is necessary in order to achieve an unbounded competitive ratio, for this problem, by a deterministic or randomized online algorithm. BMC has many applications both in its offline and online versions, such as facility location [32, 8, 7], where the budget is the number of facilities, and web streaming [36], where the budget is the number (or total size) of web objects that can be stored in memory.

The BUDGETED MAXIMUM COVERAGE problem, as well as its dual problem, the SET COVER problem, both in their weighted (general costs) and unweighted (unit costs) versions, are fundamental, widely studied problems (cf. [23, 40]). Even in their unweighted versions they are NP-hard problems [20], while approximation algorithms do exist for their weighted versions (with approximation ratios of $O(\log n)$ for SET COVER [13] and $\frac{e}{e-1}$ for BMC [28]). These approximation ratios are best possible unless P = NP [18, 3]. The online version of the SET COVER problem has been studied in many variants (see, e.g., [15, 2, 11, 29]). As to the OBMC problem, only the unweighted case has been studied in the online setting, where a 4-competitive deterministic algorithm is given [36].

In the present paper we give the first results for the *budgeted* (i.e., when sets have varying costs) online maximum coverage problems. We give both upper and lower bounds on the competitive ratio of deterministic and randomized algorithms for this problem, in terms of a number of parameters of the instance.

1.1 Our Contributions

We present a deterministic $\frac{4}{1-r}$-competitive algorithm, where $r \triangleq \max_i \frac{c(S_i)}{B}$ is the maximum fraction of the budget needed for any single set. Our algorithm is inspired by the online algorithms for the case of unit costs [36, 4]. However, several new ideas are needed to cope with general costs, such as working with a fractional solution in the background and rounding it in an online manner to an integral solution while incurring only a small penalty. Furthermore, the natural algorithm, that results from reducing the weighted (set costs) case to the unit cost case by duplicating the sets, does not necessarily yield fractional solutions that can be readily converted to integral ones (i.e., many sets could be used only fractionally). Instead, we give an online algorithm for the weighted fractional setting that computes a solution which has at most one set used fractionally. Such solution can be converted to an integral one in an online manner while incurring only a small penalty (a $1/(1-r)$ factor).

On the negative side, we show that the competitive ratio of any deterministic online algorithm for OBMC must depend on r, by showing a lower bound of $\Omega(\frac{1}{\sqrt{1-r}})$. Building on our deterministic algorithms we then also give an $O(1)$-competitive randomized algorithm for OBMC.

We further give a deterministic $(\Delta + 2)$-competitive algorithm for OBMC, where $\Delta \triangleq \max_{S\in\mathcal{S}} w(S)$ is the maximum weight of a set (defined under the assumption that all element weights are at least 1). Note that for unit weights we have that Δ is the maximum set size. If $w(U)$, i.e., the total weight of all elements in the ground set, is known in advance, we show that a slight modification of that algorithm is $O(\min\{\Delta, \sqrt{w(U)}\})$-competitive. We give a matching lower bound, namely that the competitive ratio of any deterministic online algorithm for OBMC is $\Omega(\min\{\Delta, \sqrt{w(U)}\})$, even for the special case of unit weights. Note that $w(U) = n$ in the unit weights case.

We note that by applying our deterministic upper bounds to the special case of OBMC in which the sets are pairwise disjoint, we obtain results for the Removable Online Knapsack problem. This problem is the version of Online Knapsack in which preemption is allowed. Furthermore, our deterministic lower bounds apply to this problem since they can be obtained using constructions containing pairwise disjoint sets.

Due to lack of space some of the proofs are omitted from this extended abstract.

1.2 Related Work

In the Maximum Coverage problem the goal is, given an integer parameter k, to cover as many elements as possible, using at most k sets. In this case the natural greedy algorithm computes solutions whose weight is within a factor of $1-(1-\frac{1}{k})^k > 1-\frac{1}{e}$ from the optimum (see [33, 24, 23]). This ratio holds even in the more general case of nonnegative, nondecreasing, submodular set function maximization [34, 19].[1] Khuller, Moss and Naor [28] showed that Maximum Coverage cannot be approximated to within a factor better than $\frac{e}{e-1}$, unless NP $\subseteq$ DTIME$(n^{O(\log\log n)})$. Feige [18] did the same under the weaker assumption of P$\neq$NP. Ageev and Sviridenko [1] presented an approximation algorithm for Maximum Coverage that computes solutions whose weight is within a factor of $1-(1-\frac{1}{\Delta})^{\Delta}$ from the optimum, where Δ is the maximum size of a set. Buchbinder et al. [9] studied submodular maximization with cardinality constraints which contain Maximum Coverage as a special case. Khuller et al. [28] showed that BMC can be approximated to within $\frac{e}{e-1}$. Sviridenko [38] extended this result to maximization of a monotone submodular set function subject to a budget constraint.

Saha and Getoor [36] presented a deterministic 4-competitive algorithm for the Online Maximum Coverage problem. Ausiello et al. [4] analyzed a variant of the above algorithm and showed that its competitive ratio is strictly less than 4, but that it tends to 4 as k increases. They also considered the special case of Online Maximum Coverage in which vertices are used to cover edges (i.e., an element appears in exactly two sets) and provided a simple deterministic 2-competitive algorithm for the latter that simply chooses the k largest sets seen so far. Ausiello et al. [4] also gave lower bounds 2 and $\frac{3}{2}$ for Online Maximum Coverage and for that special case, respectively, on the competitive ratio of any deterministic online algorithm. Ashwinkumar [39] and Chakrabarti and Kale [12] presented streaming 4-approximation algorithms for maximizing a monotone submodular function subject to cardinality constraint. Buchbinder, Feldman, and Schwartz [10] provided constant competitive ratio algorithms for online submodular maximization with preemption and a cardinality constraint. They also gave a deterministic 4-competitive algorithm for the monotone case.

We note that Awerbuch et al. [5] studied a problem they called Online Set Cover. However they actually consider a variant of Online Maximum Coverage in which the elements arrive in an online manner, and the sets are revealed during this process. The goal is to cover as many elements as possible using k sets without preemption, where an element is considered covered only by a set that contains it which is added to the solution after the

[1] A function f is called submodular if $f(T)+f(T') \geq f(T \cup T') + f(T \cap T')$ for every two sets T and T' in the domain of f.

arrival of the element. Awerbuch et al. [5] gave a randomized $O(\log n \log \frac{m}{k})$-competitive algorithm for this problem.

The dual of Maximum Coverage is the classical Set Cover problem. For the unweighted Set Cover problem, Johnson [27] and Lovász [30] showed that the greedy algorithm is an H_Δ-approximation algorithm, where H_n the nth harmonic number. This result was generalize by Chvátal [13] to the weighted case. Feige [18] proved a lower bound of $(1 - o(1)) \ln n$ on the approximability of that problem (unless NP $\subseteq$ DTIME$(n^{O(\log\log n)})$). In [35, 3] it was shown that Set Cover cannot be approximated within a factor of $c \log n$, for some $c > 0$, unless P=NP. Set Cover can also be approximated to within a factor of $\Delta_U \triangleq \max_{u \in U} |\{S : u \in S\}|$ [22, 6]. However, it is NP-hard to approximate it within $\Delta_U - 1 - \varepsilon$, for any $\varepsilon > 0$, assuming $\Delta_U > 2$ [16], or within 1.36 for $\Delta_U = 2$ [17].

A certain online version of Set Cover was studied by Alon et al. [2]. In this problem the sets are known in advance, subsets of the elements arrives in an online manner, and the goal is to cover all seen elements with a sub collection of sets of minimal cardinality. A deterministic $O(\log m \log n)$-competitive algorithm and a nearly matching lower bound are given in [2] (n is the number of elements and m the number of sets).

Knapsack is a special case of BMC in which the sets are pairwise disjoint. Knapsack is known to be NP-hard, but admits an FPTAS [37, 25]. Removable Online Knapsack (ROK) is a special case of OBMC in which the sets are pairwise disjoint. In other words, in ROK items arrive one by one, each with its load and value. An online algorithm is required to accept or to reject an incoming item upon arrival, and it is allowed to drop previously accepted items to make room for a new item. The goal is to maximize the value accrued by the accepted items, under the constraint that their total load is within a given maximum load. Iwama and Taketomi [26] considered the special case of ROK in which the value of and item is equal to its load. They provided a deterministic competitive algorithm whose ratio is $\frac{\sqrt{5}+1}{2} \approx 1.62$, and a matching lower bound. Han, Kawase, and Makino [21] and Cygan, Jeż, and Sgall [14] gave a randomized 2-competitive algorithm and showed that the competitive ratio of any randomized online algorithm is at least $\frac{e+1}{e}$. Both lower bounds apply to OBMC. Non-removable Online Knapsack is the variant in which accepted items cannot be dropped. In this case the deterministic [31] and randomized [41] competitive ratios are known to be unbounded.

1.3 The Model

An instance of the BMC problem is composed of a weighted ground set $U = \{u_1, \ldots, u_n\}$, with each element having a known weight $w(u_i) \geq 1$ (we define all weights to be at least 1 to avoid arbitrary scaling of the weights). The instance is further composed of a collection $\mathcal{S} = \{S_1, \ldots, S_m\}$ of sets, where $S_i \subseteq U$, for every i. The cost of a set S_i is denoted $c(S_i)$. W.l.o.g. we assume that the budget cap is 1, and therefore $0 < c(S_i) \leq 1$, for every i.

In OBMC the sets of $\mathcal{S}$ arrive online, where each set S_i is given by the elements that it contains, as well as its cost, $c(S_i)$. When a set arrives the online algorithm has to decide whether to *accept* it or to *reject* it, under the constraint that the currently accepted sets at any given time should have a cumulative cost not larger than 1. The algorithm can *drop* a previously accepted set, i.e., extract it from the currently accepted sets. However, a rejected or dropped set cannot later be re-accepted. The goal of the online algorithm is to maximize the total weight of the elements covered by the accepted sets.

Given a set $S \subseteq U$, we define its weight to be $w(S) = \sum_{u \in S} w(u)$. Given a sub-collection $\mathcal{C} \subseteq \mathcal{S}$, we define its cost to be $c(\mathcal{C}) = \sum_{S \in \mathcal{C}} c(S)$. Given an instance of the OBMC problem we define $\Delta \triangleq \max_{S \in \mathcal{S}} w(S)$, i.e., the maximum weight of a set. Note that for unit weights we have $\Delta = \max_{S \in \mathcal{S}} |S|$. Further we define $r \triangleq \max_i c(S_i)$ to be the maximum cost of a set.

2 Deterministic Lower Bound

In this section we give lower bounds on the competitive ratio of deterministic online algorithms for OBMC in terms of three parameters: the number of elements n, the weight of the heaviest set, Δ, and the maximum cost of a set, r.

We start with our lower bound in terms of n and Δ.

▶ **Theorem 1.** *The competitive ratio of any deterministic online algorithm for OBMC is* $\Omega(\min\{\sqrt{n}, \Delta\})$.

Proof. Let ALG be any deterministic online algorithm for OBMC. Consider an input sequence containing a collection of subsets of a ground set U that contains (at most) k^2 unit weight elements, for an arbitrary integer k. We define the input sequence given by an adversary. The input sequence starts with a set S_0, where $S_0 = \{u_1, \dots, u_k\}$ and $c(S_0) = 1$. If ALG does not accept S_0, then the sequence terminates. Otherwise, the sequence continues with $S_1, S_2, \dots$, such that $S_i = \{u_i\}$ and $c(S_i) = 1/k^2$, for $i \geq 1$, until either ALG drops S_0 or $i = k^2$.

To analyze the competitive ratio of ALG, note that there are three options as to the actual input sequence given by the adversary:

- If ALG rejects S_0, then OPT covers k elements using S_0 while ALG covers nothing.
- If ALG accepts S_0 and when S_i, for some $i \geq 1$, arrives, ALG drops S_0 and (possibly) accepts S_i, then OPT covers k elements using S_0, while ALG covers at most one element.
- If ALG accepts S_0 and never drops it, then ALG covers k elements, while OPT covers k^2 elements using $S_1, \dots, S_{k^2}$.

Hence the competitive ratio of ALG is at least k, and the theorem follows, since $k = \sqrt{n}$ and $k = \Delta$. ◀

A similar construction works for r as well.

▶ **Theorem 2.** *The competitive ratio of any deterministic online algorithm for OBMC is* $\Omega(1/\sqrt{1-r})$. *In particular, if* $r = 1$ *the competitive ratio of any deterministic algorithm is unbounded.*

Proof. The theorem clearly holds for $r \leq \frac{8}{9}$, so we assume for the rest of the proof that $r > \frac{8}{9}$. If $r < 1$, let k be a positive integer such that $\frac{1}{2} \cdot \frac{1}{\sqrt{1-r}} \leq k \leq \frac{1}{\sqrt{1-r}}$. There exists such a k, since $\frac{1}{2} \cdot \frac{1}{\sqrt{1-r}} \geq 1.5$. Also, note that $r + \frac{1}{k^2} \geq r + (1-r) > 1$ and that $\frac{1}{k^2} \leq 4(1-r) \leq \frac{4}{9} < r$. We use the same adversary that is used in Theorem 1 with k defined as above, with $c(S_0) = r$, and with $c(S_i) = 1/k^2$, for $i \geq 1$. The rest of the proof is similar to the proof of Theorem 1, showing that the competitive ratio is $k = \Theta(1/\sqrt{1-r})$.

If $r = 1$, we can pick an arbitrarily large positive integer k, which then shows that the competitive ratio is unbounded. ◀

We can conclude with the following theorem.

▶ **Theorem 3.** *Let* ALG *be a c-competitive deterministic online algorithm for OBMC. Then* $c = \Omega(\min\{\sqrt{n}, \Delta, \frac{1}{\sqrt{1-r}}\})$.

We note that the above constructions can be slightly modified to consist of pairwise disjoint sets.

▶ **Theorem 4.** *Let* ALG *be a c-competitive deterministic online algorithm for* REMOVABLE ONLINE KNAPSACK. *Then* $c = \Omega(\min\{\sqrt{n}, \Delta, \frac{1}{\sqrt{1-r}}\})$.

3 $O(\frac{1}{1-r})$-competitive Algorithm

In this section we present a deterministic $\frac{4}{1-r}$-competitive algorithm for OBMC. In what follows we assume that $r < 1$. Otherwise, the competitive ratio of any deterministic algorithm is unbounded (see Theorem 2).

Roughly speaking, our algorithm is inspired by the online algorithm for the case of unit costs [36]. We use a similar greedy rule: a set joins the solution if its marginal benefit, with respect to the current solution, is high enough. However, in contrast to the unit cost algorithm, we sort the sets in the current solution by cost effectiveness (to be defined later), and consider only sets that are contained in the prefix of the sets of the current solution which sums to a cost of at most 1. This prefix may be fractional, namely there may be a set that is only partly considered, which then complicates both the algorithm and the analysis. In what follows we define some notations and then present formally the algorithm.

Definitions and notations

Our algorithm is defined based on an imaginary fractional solution to the problem that we maintain throughout receiving the input. In this fractional solution, the algorithm can use only a fraction $x(S) \in [0,1]$ of a given set S, paying only $x(S) \cdot c(S)$, and covering by set S at most an $x(S)$ fraction of every element $v \in S$. This fractional solution can be defined using the following variables. In what follows we refer by *time* i to the time *after* the algorithm has processed the ith input set. A variable with a subscript i refers to the value of the variable at time i.

- $x_i(S) \in [0,1]$, for $S \in \mathcal{S}$, is the fraction of set S used by the algorithm at time i.
- $z_i(v,S) \in [0,1]$, for $S \in \mathcal{S}$ and $v \in U$, is the fraction of v that is covered by the algorithm using set S at time i. We set $z_i(v,S) = 0$ if $v \notin S$.

We further define, given any $\vec{z}_i$, for $v \in U$, $\hat{z}_i(v) \triangleq \sum_{S\in\mathcal{S}} z_i(v,S)$.

The (fractional) optimization problem can now be defined by the following linear program:

$$\begin{array}{lll}
\max & \sum_{v\in U} \hat{z}(v)\cdot w(v) & \\
\text{s.t.} & \sum_{S\in\mathcal{S}} x(S)\cdot c(S) \le 1 & \\
& \hat{z}(v) \le 1 & \forall v \in U \\
& z(v,S) \le x(S) & \forall v \in U, S \in \mathcal{S}, \\
& x(S) \in [0,1] & \forall S \in \mathcal{S} \\
& z(v,S) \ge 0 & \forall v \in U, S \ni v \\
& z(v,S) = 0 & \forall v \in U, S \not\ni v
\end{array}$$

Before presenting the algorithm we need the following further notations.

- $w(\vec{z}) \triangleq \sum_v \hat{z}(v)w(v)$, i.e., the total weight covered in a solution defined by the matrix $\vec{z}$.
- $\rho(\vec{z},\vec{x},S) \triangleq \frac{\sum_v z(v,S)w(v)}{x(S)c(S)}$, if $x(S) > 0$, and $\rho(\vec{z},\vec{x},S) \triangleq 0$, otherwise. That is, $\rho(\vec{z},\vec{x},S)$ stands for the total weight covered by set S in a solution defined by the matrix $\vec{z}$, divided by the "actual cost" paid for S. We call this quantity the *efficiency* of set S with respect to the solution $(\vec{z},\vec{x})$.

3.1 The Algorithm

Our algorithm maintains two variables z and x, corresponding to the variables by the same names described above, and which represent a current fractional (imaginary) solution held by the online algorithm. As the algorithm is an online algorithm, we allow it to increase $z(\cdot,S)$ and $x(S)$ only when set S arrives.

Algorithm 1: insert(S, x, z)

```
1  $x' \leftarrow x$; $z' \leftarrow z$
2  $x'(S) \leftarrow 1$
3  foreach $v \in U$ do $z'(v,S) \leftarrow \begin{cases} 1 - \hat{z}'(v) & v \in S, \\ 0 & v \notin S \end{cases}$
4  $\hat{\mathcal{S}} \leftarrow \{S : x'(S) > 0\}$ ; $\ell \leftarrow |\hat{\mathcal{S}}|$
5  Order the sets in $\hat{\mathcal{S}}$ by non-increasing value of $\rho(\vec{z'}, \vec{x'}, S)$; let $S_{j_1}, S_{j_2}, \ldots, S_{j_\ell}$ be the
     ordering.
6  $k \leftarrow \max\left\{k' \leq \ell : \sum_{i=1}^{k'-1} x'(S_{j_i})c(S_{j_i}) < 1\right\}$
7  $\chi \leftarrow \min\left\{\frac{1-\sum_{i=1}^{k-1} x'(S_{j_i})c(S_{j_i})}{c(S_{i_k})}, x'(S_{i_k})\right\}$
8  foreach $v \in S_{i_k}$ do $z'(v, S_{i_k}) \leftarrow \frac{\chi}{x'(S_{i_k})} \cdot z'(v, S_{i_k})$
9  $x'(S_{i_k}) \leftarrow \chi$
10 for $i = k+1$ to $\ell$ do
11     $x'(S_{j_i}) \leftarrow 0$
12     foreach $v \in U$ do $z'(v, S_{j_i}) \leftarrow 0$
13 return $(x', z')$
```

Algorithm 2: α**-greedy**; operations when set S_i arrives.

```
1  $x'(S_i) \leftarrow 1$
2  $z'(v, S_i) \leftarrow \begin{cases} 1 - \hat{z}(v) & \text{if } v \in S_i \\ 0 & \text{otherwise} \end{cases}$
3  if $\rho(\vec{z'}, \vec{x'}, S_i) > \alpha \cdot w(z)$ then $(x, z) \leftarrow \text{insert}(S_i, x, z)$
```

We first define a procedure **insert** that we use in the algorithm. This procedure takes a set S and inserts it into the current solution represented by the variables z and x. This changes the values of z and x to represent the new solution.

We can now define the online algorithm, that we call α-greedy, for any $\alpha > 1$. The optimal value for α will be defined later in the analysis.

α-greedy

We initialize the two (vector) variables $\vec{z} \leftarrow \vec{0}$, $\vec{x} \leftarrow \vec{0}$. Then, for every set S_i that arrives, we use the operations defined in the pseudocode in Algorithm 2.

At any given time, the solution held by the online (regular, integral) algorithm consists of all the sets S for which $x(S) = 1$. As we later prove, the algorithm has, at any given time, at most one set S "used fractionally", i.e., with $x(S) \in (0, 1)$.

We claim that the algorithm is a well defined online algorithm for our problem. That is, that (1) the algorithm accepts a set only when this set arrives, i.e., a set that is not accepted when it arrives, or accepted but subsequently dropped, cannot later be part of the solution; and (2) the solution held by the algorithm at any given time is feasible, i.e., the total budget used by the algorithm is at most 1 at any given time. To see these two points observe that all the changes in the variables held by the algorithm are done in procedure **insert**. Procedure **insert** assigns a value of 1 to variable $x'(S)$, only for S which is the inserted set: an explicit assignment of 1 is only done in Line 1, and in Line 9 the value of $x'(S_{i_k})$ cannot grow compared to the previous round. This proves point (1). Point (2) requires a bit more of formalism, which is given in the proof of the following lemma.

▶ **Lemma 5.** *For any time i, $\sum_{j\in O_i} c(S_j) \le 1$, where $O_i = \{j : x_i(S_j) = 1\}$.*

Proof. We prove this lemma by induction on i. For the base of the induction, i.e., when $i = 0$, we have that $\vec{x}_0 = \vec{0}$. It follows that $O_0 = \emptyset$ and we are done. For the inductive step, we assume that the claim is true for $i - 1$, for $i \ge 1$ and prove it for i. If **insert** is not called during iteration i, then $x_i = x_{i-1}$ and $O_i = O_{i-1}$, and the claim follows from the inductive hypothesis. If **insert** is activated, then x_i is constructed such that $x_i(S_{j_i}) = 0$ if $j_i > k$ and $\sum_{i=1}^{k} x_i(S_{j_i})c(S_{j_i}) \le 1$. Hence $\sum_{j\in O_i} c(S_j) \le 1$. ◀

We now give two technical claims which we use in the analysis. The first claim is immediate from the code of **insert**.

▶ **Claim 6.** *The efficiency of a given set S remains the same throughout the period when $x(S) > 0$.*

The next claim says that at any time i, there may be at most one set S such that $x(S) \in (0, 1)$, and that if such set exists then the whole budget is used. Furthermore, if such set exists then that set is the one with minimum efficiency among the sets with non-zero x-coefficient. We note that these properties of our algorithm are the properties that allow one to obtain in an online manner an integral solution, and that a simple, natural reduction of the weighted fractional case to the unweighted case does not yield an algorithm with such properties.

▶ **Claim 7.** *Let $\mathcal{S}_i^+ = \{S : x_i(S) > 0\}$. There is at most one set $S \in \mathcal{S}^+$ such that $x(S) < 1$. If such a set S exists then (1) $\sum_{S\in\mathcal{S}_i^+} x_i(S)c(S) = 1$; and (2) $\rho(\vec{z}_i, \vec{x}_i, S) \le \rho(\vec{z}_i, \vec{x}_i, S')$, for any $S' \in \mathcal{S}^+ \setminus \{S\}$.*

Proof. We prove the claim by induction on i. For the basis of the induction, i.e., for $i = 0$, the claim is trivial in the empty sense. We now assume that the claim holds for $i - 1$, for $i \ge 1$, and we prove it for i.

If **insert** is not invoked during iteration i, then the claim clearly holds by the induction hypothesis. If procedure **insert** is invoked we have that (according to Line 3 of α-greedy)

$$\rho(z', x', S_i) > \alpha \cdot w(z_{i-1}) = \alpha \cdot \sum_{S\in\mathcal{S}_{i-1}^+} \rho(\vec{z}_{i-1}, \vec{x}_{i-1}, S) \cdot x_{i-1}(S) \cdot c(S) \ .$$

We now consider two cases. The first case is when a set S with $x_{i-1}(S) \in (0, 1)$ exists. In that case, by the induction hypothesis we have that $\sum_{S\in\mathcal{S}_{i-1}^+} x_{i-i}(S)c(S) = 1$. It follows that $\rho(z', x', S_i) > \min_{S\in\mathcal{S}_{i-1}^+}\{\rho(\vec{z}_{i-1}, \vec{x}_{i-1}, S)\}$ (recall that $\alpha > 1$), and therefore the new set S_i is not the last set in the non-increasing order of efficiency defined in Line 5 of procedure **insert**. Points (1) and (2) for time i therefore follow from the code of **insert** and the induction hypothesis of point (2). The second case is when there is no set S with $x_{i-1}(S) \in (0, 1)$. In that case the code of **insert** directly guarantees both points (1) and (2) for time i. ◀

3.2 Competitive Analysis

We analyze the competitive ratio of the algorithm using a charging scheme argument. We first describe the scheme in general terms – more details are given in the following paragraphs. Let OPT be an optimal solution, i.e., an optimal collection of sets. As the sets of OPT arrive, each new element u that is covered by OPT is allotted $w(u)$ monetary units (or simply money). We show that the allotted money can be moved between elements of U such that all the money is allotted to the items covered by ALG, and such that each element v that is covered by ALG has at most $\frac{4}{1-r} \cdot w(v)$ amount of money. This gives an upper bound of $\frac{4}{1-r}$ on the competitive ratio.

We now formally define the charging scheme. Let $\text{OPT}_k = \text{OPT} \cap \{S_1, \ldots, S_k\}$. If $S_i \in \text{OPT}$, let $F_i \triangleq S_i \setminus \cup_{S \in \text{OPT}_{i-1}} S$, i.e., F_i contains all the elements covered by OPT that, when S_i arrives, are covered for the first time by OPT (according to the order of arrival of the sets). When a set $S_i \in \text{OPT}$ arrives, each element $u \in F_i$ is allotted $w(u)$ money. The money that is allotted this way is always held by items covered by ALG, and may sometimes be moved between items covered by ALG; furthermore this money is partitioned into *blue money* and *red money*. We will denote by $\text{blue}_i(v)$ and by $\text{red}_i(v)$ the amount of blue and red money, respectively, held by $v \in U$ at time i.

Now, the rules that govern the allotment and movement of the money are the following rules, applied when a set S_i arrives.

- **Transfer of money**. This rule is applied if the then part of Line 3 of α-**greedy** is reached, i.e., if procedure **insert** is invoked. In this case elements that lost coverage relinquish part of their money, and this money is transferred to elements that gain coverage. Let $\hat{z}'_i(v) \triangleq \sum_{j=1}^{i-1} z_i(v, S_j)$ and $Z_i \triangleq \{v : \hat{z}'_i(v) < \hat{z}_{i-1}(v)\}$. That is, we look at the coverage of an item v by all but the last arriving set, and observe if this coverage reduced during the course of iteration i.
  1. **Out-transfer of money**. For each $v \in Z_i$ let $\delta_v = (1 - \frac{\hat{z}'_i(v)}{\hat{z}_{i-1}(v)})$. We remove from v an amount of R_v red money which is a δ_v-fraction of its current red money, and an amount of B_v blue money which is a δ_v-fraction of its current blue money.
  2. **In-transfer of money**. These total amounts of removed red and blue money are distributed to the various red and blue money variables of $u \in S_i$ proportionally to $z_i(u, S_i) \cdot w(u)$. That is, each element $u \in S_i$ gets additional $\frac{z_i(u,S_i) \cdot w(u)}{\sum_{v \in S_i} z_i(v,S_i) \cdot w(v)} \cdot \sum_{v \in Z_i} R_v$ red money, and gets additional $\frac{z_i(u,S_i) \cdot w(u)}{\sum_{v \in S_i} z_i(v,S_i) \cdot w(v)} \cdot \sum_{v \in Z_i} B_v$ blue money.
- **Creation of new money**. If $S_i \in \text{OPT}$, $w(v)$ money is distributed for each $v \in F_i$ as follows: (1) v gets $\hat{z}_i(v)w(v)$ newly created blue money; and (2) a total amount of $R_i(v) = (1 - \hat{z}_i(v))w(v)$ red money is created, and distributed among the items u currently covered by the algorithm, i.e., each element $u \in U$ gets additional red money in the amount of $R_i(v)\frac{\hat{z}_i(u)w(u)}{w(\vec{z}_i)}$ (i.e., the red money is distributed proportionally to the contribution of each element to the total weight of the current solution).

We now prove upper bounds on the amount of red and blue money held by any element at any given time. We start with a technical claim. In the next lemma we show that if S_i is accepted, then the amount of coverage provided by set S_i is more than α times the coverage lost due to the sets (or part of sets) pushed out.

▶ **Lemma 8.** *Assume that* $x_i(S_i) > 0$. *Then,*

$$\rho(\vec{z}_i, \vec{x}_i, S_i) > \alpha \cdot \frac{\sum_{j<i} \sum_{u \in U} w(u)[z_{i-1}(u, S_j) - z_i(u, S_j)]}{x_i(S_i)c(S_i)} .$$

Proof. Since $x_i(S) > 0$ we know that $\rho(\vec{z}_i, \vec{x}_i, S_i) = \frac{\sum_{u \in U}(1-\hat{z}_{i-1}(u))w(u)}{c(S_i)} > \alpha \cdot w(z_{i-1})$ and that **insert** was invoked. Consider what happens to the solution $(\vec{z}_{i-1}, \vec{x}_{i-1})$ during the invocation of **insert** at iteration i. Some sets do not lose coverage (i.e., sets S for which $x_i(S) = x_{i-1}(S) = 1$). Other sets may leave the cover and their cover is lost (i.e., $x_{i-1}(S) > x_i(S) = 0$). In addition, by Claim 7, there may be at most one set that partly leaves the cover, and so it both retains and loses coverage (i.e., $x_{i-1}(S) > x_i(S) > 0$). Define $\mathcal{Y}_i \triangleq \{S_j : x_{i-1}(S_j) > x_i(S_j)\}$, namely $\mathcal{Y}_i$ contains the sets that lose coverage due to the invocation of **insert** during the ith iteration.

If no coverage is lost during iteration i, namely if $\mathcal{Y}_i = \emptyset$ (or $Z_i = \emptyset$), then $\sum_{j<i} \sum_{u \in U} w(u)[z_{i-1}(u, S_j) - z_i(u, S_j)] = 0$, and we are done since $x_i(S) > 0$ implies that $\rho(\vec{z}_i, \vec{x}_i, S_i) > 0$.

If $\mathcal{Y}_i \neq \emptyset$, then by Claim 7 and the code of **insert** it follows that $\rho(z_{i-1}, x_{i-1}, S) \leq \rho(z_i, x_i, S')$, for every $S \in \mathcal{Y}_i$ and S' such that $x_i(S') > 0$. In words, the efficiency of the sets that retain coverage is at least as high as the efficiency of the sets that lost coverage. As mentioned above, observe that there may be at most one set S_j, such that $S_j \in \mathcal{Y}_i$ and $x_i(S_j) > 0$. Define $\rho^i_{\min} \triangleq \min\{\rho(\vec{z}_{i-1}, \vec{x}_{i-1}, S_j) : x_i(S_j) > 0, j < i\}$ and $\rho(\mathcal{Y}_i) \triangleq \max\{\rho(\vec{z}_{i-1}, \vec{x}_{i-1}, S_j) : S_j \in \mathcal{Y}_i\}$, and we have that $\rho(\mathcal{Y}_i) \leq \rho^i_{\min}$.

Define W^L_i to be the total weight of lost coverage during the ith iteration and define W^R_i to be the total weight of retained coverage at the ith iterations. That is, define: $W^L_i \triangleq \sum_{j<i} \sum_{u \in U} w(u)[z_{i-1}(u, S_j) - z_i(u, S_j)]$ and $W^R_i \triangleq \sum_{j<i} \sum_{u \in U} w(u) z_i(u, S_j)$. Observe that $w(\vec{z}_{i-1}) = W^L_i + W^R_i$. We have that $W^R_i = \sum_{j<i} \rho(\vec{z}_{i-1}, \vec{x}_{i-1}, S_j) x_i(S_j) c(S_j) \geq \rho^i_{\min} \sum_{j<i} x_i(S_j) c(S_j)$, and

$$\begin{aligned}
W^L_i &= \sum_{j<i} \rho(\vec{z}_{i-1}, \vec{x}_{i-1}, S_j)[x_{i-1}(S_j)c(S_j) - x_i(S_j)c(S_j)] \\
&= \sum_{S_j \in \mathcal{Y}_i} \rho(\vec{z}_{i-1}, \vec{x}_{i-1}, S_j)[x_{i-1}(S_j)c(S_j) - x_i(S_j)c(S_j)] \\
&\leq \rho(\mathcal{Y}_i) \sum_{S_j \in \mathcal{Y}_i} [x_{i-1}(S_j)c(S_j) - x_i(S_j)c(S_j)] \\
&= \rho(\mathcal{Y}_i) \sum_{j<i} [x_{i-1}(S_j)c(S_j) - x_i(S_j)c(S_j)] \\
&\leq \rho^i_{\min} \left[1 - \sum_{j<i} x_i(S_j)c(S_j)\right] \\
&\leq W^R_i \cdot \frac{1 - \sum_{j<i} x_i(S_j)c(S_j)}{\sum_{j<i} x_i(S_j)c(S_j)} .
\end{aligned}$$

It follows that

$$w(\vec{z}_{i-1}) = W^L_i + W^R_i \geq W^L_i + W^L_i \cdot \frac{\sum_{j<i} x_i(S_j)c(S_j)}{1 - \sum_{j<i} x_i(S_j)c(S_j)} = \frac{W^L_i}{1 - \sum_{j<i} x_i(S_j)c(S_j)} .$$

Now since $\mathcal{Y}_i \neq \emptyset$, we have that **insert** decreased coverage of at least one set and by the code of **insert** this implies that $\sum_S x_i(S)c(S) = 1$. Hence we have that $w(\vec{z}_{i-1}) \geq \frac{W^L_i}{x_i(S_i)c(S_i)}$, and the lemma follows. ◀

We are now ready to give in the next lemma an upper bound on the amount of blue money that is allotted to any element $u \in U$ at any given time.

▶ **Lemma 9.** *At any given time i and for every $u \in U$, blue$_i(u) \leq w(u)\hat{z}_i(u) \cdot \frac{\alpha}{\alpha-1}$.*

Proof. In this proof we consider separately new blue money, that was not transferred yet, and old blue money, that was transferred at least once. Observe that for each element $u \in U$

there is at most one index j such that $u \in F_j$. We denote this index by $f(u)$. We prove by induction on i that at any given time i and for every $u \in U$, we have that

1. $\text{blue-old}_i(u) \le w(u)\hat{z}_i(u) \cdot \frac{1}{\alpha-1}$, and
2. $\text{blue-new}_i(u) \le \begin{cases} 0 & i < f(u), \\ w(u)\hat{z}_i(u) & i \ge f(u). \end{cases}$

For $i = 0$, i.e., before the first input set arrives, the claims hold as there is no (blue) money. We now prove the claims for time $i \ge 1$, assuming that the claims hold for time $i-1$. We analyze the changes in $\text{blue-old}_i(\cdot)$ and $\text{blue-new}_i(\cdot)$ taking into account one by one, in their order, the operations of the charging scheme defined above. Recall that $\hat{z}'_i(v) \triangleq \sum_{j=1}^{i-1} z_i(v, S_j)$ and that $Z_i \triangleq \{v : \hat{z}'_i(v) < \hat{z}_{i-1}(v)\}$, and observe that the out-transfer and in-transfer phases are performed only if procedure **insert** is invoked.

- **Out-transfer of money**. Blue money (old or new) is removed from elements $u \in Z_i$. For such an element u we have at the end of the out-transfer phase that $\text{blue}_i(u) = \text{blue}_{i-1}(u) \cdot \frac{\hat{z}'_i(u)}{\hat{z}_{i-1}(u)}$. By the induction hypothesis this is at most

$$\text{blue-old}_i(u) \le w(u)\hat{z}_{i-1}(u) \cdot \frac{1}{\alpha-1} \cdot \frac{\hat{z}'_i(u)}{\hat{z}_{i-1}(u)} = w(u)\hat{z}'_i(u) \cdot \frac{1}{\alpha-1}$$

$$\text{blue-new}_i(u) \le w(u)\hat{z}_{i-1}(u) \cdot \frac{\hat{z}'_i(u)}{\hat{z}_{i-1}(u)} = \begin{cases} 0 & i-1 < f(u), \\ w(u)\hat{z}'_i(u) & i-1 \ge f(u). \end{cases}$$

  For $u \in U \setminus Z_i$, $\text{blue-old}_i(u) = \text{blue-old}_{i-1}(u)$ and $\text{blue-old}_i(u) = \text{blue-old}_{i-1}(u)$. Using the induction hypothesis, at the end of the out-transfer phase, the same bound holds for $u \in U \setminus Z_i$ as well.

- **In-transfer of money**. First notice that there is no in-transfer of new blue money. Each element $u \in S_i$ gets old blue money in the amount of

$$\frac{z_i(u, S_i) \cdot w(u)}{\sum_{v \in S_i} z_i(v, S_i) \cdot w(v)} \cdot \sum_{v \in Z_i} \left(1 - \frac{\hat{z}'_i(v)}{\hat{z}_{i-1}(v)}\right) \text{blue}_{i-1}(v) \ .$$

  By the inductive hypothesis this is at most

$$\frac{z_i(u, S_i) \cdot w(u)}{\sum_{v \in S_i} z_i(v, S_i) \cdot w(v)} \cdot \sum_{v \in Z_i} (\hat{z}_{i-1}(v) - \hat{z}'_i(v)) \cdot w(v) \cdot \frac{\alpha}{\alpha-1} \ ,$$

  and by Lemma 8 it follows that this is at most

$$\frac{z_i(u, S_i) \cdot w(u)}{\sum_{v \in S_i} z_i(v, S_i) \cdot w(v)} \cdot \frac{1}{\alpha-1} \cdot \rho(\vec{z}_i, \vec{x}_i, S_i)x_i(S_i)c(S_i) \ \le \ z_i(u, S_i) \cdot w(u) \cdot \frac{1}{\alpha-1} \ .$$

  Hence, using the upper bound on $\text{blue-old}_i(u)$ at the end of the out-transfer phase, we have that at the end of in-transfer phase:

$$\text{blue-old}_i(u) \le w(u)\hat{z}'_i(u) \cdot \frac{1}{\alpha-1} + w(u)z_i(u, S_i) \cdot \frac{1}{\alpha-1} = w(u)\hat{z}_i(u) \cdot \frac{1}{\alpha-1} \ .$$

  Since no transfer of new blue money occurs by the scheme, it holds at the end of the out transfer phase, like at the end of the in-transfer phase that:

$$\text{blue-new}_i(u) \le \begin{cases} 0 & i-1 < f(u), \\ w(u)\hat{z}'_i(u) & i-1 \ge f(u). \end{cases}$$

- **Creation of new money**. if $S_i \in \textsc{opt}$, then any $u \in F_i$ receives $\hat{z}_i(u)w(u)$ new blue money. Observe that for such u, $f(u) = i$. Hence, using our claims as to the end of the in-transfer phase, the inductive claim holds at the end of the creation-of-new-money phase. ◀

We now give an upper bound on the amount of red money an element may have. The proof of the following lemma is omitted from this extended abstract.

▶ **Lemma 10.** *At any given time i and for every $u \in U$, $red_i(u) \leq w(u)\hat{z}_i(u) \cdot \alpha \cdot c(\textsc{opt}_i)$.*

Using Lemmas 9 and 10 we now give a lower bound on the weight of the fractional solution $(\vec{z}, \vec{x})$.

▶ **Lemma 11.** $w(\vec{z}) \geq \frac{1}{4}w(\cup_{S\in\textsc{opt}}S)$ *at termination.*

Proof. First observe that the total amount of blue and red money created during the course of the run is equal to the weight of the elements covered by $\textsc{opt}$, and that all created money remains in the system, held by the various elements $u \in U$, until the end of the run. We therefore compare the total amount of money held at the end by the elements $u \in U$, to the weight of the elements covered by the online algorithm.

Let n be the number of sets in the input sequence. Since $c(\textsc{opt}) = c(\textsc{opt}_n) \leq 1$, we have, for $\alpha = 2$ and using Lemma 9 and Lemma 10, that for each $u \in U$, $\text{blue}_n(u) + \text{red}_n(u) \leq w(u)\hat{z}_n(u) \cdot (\alpha + \frac{\alpha}{\alpha-1}) = w(u)\hat{z}_n(u) \cdot 4$. ◀

We now give a lower bound on the weight of the (integral) solution returned by the online algorithm, in terms of the value of the fractional solution $(\vec{z}, \vec{x})$.

▶ **Lemma 12.** *Let $\textsc{alg}_i$ be the integral solution returned by the online algorithm after processing set S_i. Then, $w(\cup_{S\in\textsc{alg}_i}) \geq (1-r)\cdot w(\vec{z}_i)$.*

Proof. By definition we have that $\textsc{alg}_i = \{S : x_i(S) = 1\}$. Each such set (fully) covers all its elements, hence $w(\cup_{S\in\textsc{alg}_i}) \geq \sum_u \sum_{S\in\textsc{alg}_i} w(u)z_i(u,S)$. By Claim 7 there may be at most one set S' such that $x_i(S') \in (0,1)$. If such a set does not exists, then $w(\hat{z}_i) = \sum_u \sum_{S\in\textsc{alg}_i} w(u)z_i(u,S)$, and we are done. If there exists a set S' such that $x_i(S') \in (0,1)$, then by Claim 7 we have that $\sum_{S\in\textsc{alg}_i} c(S) = 1 - x_i(S')c(S') > 1 - c(S')$. Also, due to Claim 7, $\rho(\vec{z}_i, \vec{x}_i, S') \leq \rho(\vec{z}_i, \vec{x}_i, S)$, for any $S \in \textsc{alg}_i$. Therefore

$$\begin{aligned} w(\cup_{S\in\textsc{alg}_i}) &\geq \sum_{S\in\textsc{alg}_i}\sum_u w(u)z_i(u,S) \\ &\geq \frac{1-c(S')}{c(S')}\cdot\sum_u w(u)z_i(u,S') \\ &\geq \frac{1-r}{r}\cdot\sum_u w(u)z_i(u,S')\ . \end{aligned}$$

(Recall that r is the highest set-cost appearing in the input sequence). It follows that $w(\cup_{S\in\textsc{alg}_i}) \geq (1-r)\cdot\sum_{S\in\textsc{alg}_i\cup\{S'\}}\sum_u w(u)z_i(u,S) = (1-r)\cdot w(\vec{z}_i)$. ◀

It remains to give an upper bound on the competitive ratio of the algorithm.

▶ **Theorem 13.** *Algorithm 2-greedy is $\frac{4}{1-r}$-competitive.*

Proof. By Lemma 11 we have that $w(\vec{z}) \geq \frac{1}{4}w(\cup_{S\in\textsc{opt}}S)$ at termination. By Lemma 12 the total weight of the elements covered by the online algorithm is at least $(1-r)\cdot w(\vec{z})$. ◀

4 $O(1)$-competitive Randomized Algorithm

In this section we give a randomized online algorithm with an $O(1)$ competitive ratio. The algorithm is based on the deterministic $\frac{4}{1-r}$-competitive algorithm from Section 3.

The algorithm is a barely random algorithm that chooses to run one of the following two algorithms with probability 1/2 each:

1. Always keep a single set S_j, which is the set with the highest weight seen so far.
2. Run algorithm α-**greedy** of Section 3, with $\alpha = 2$, only on sets with cost at most $\frac{1}{3}$.

▶ **Theorem 14.** *There is a* 16*-competitive randomized online algorithm for OBMC.*

Proof. Let OPT be an optimal solution, and define $\text{OPT}_{>} = \{S \in \text{OPT} : c(S) > \frac{1}{3}\}$ and $\text{OPT}_{\leq} = \{S \in \text{OPT} : c(S) \leq \frac{1}{3}\}$. Observe that $w(\cup_{S\in\text{OPT}_{>}}S) + w(\cup_{S\in\text{OPT}_{\leq}}S) \geq w(\cup_{S\in\text{OPT}}S)$ and that $|\text{OPT}_{>}| \leq 2$. We have that at least one of the two following options must occur: (i) $w(\cup_{S\in\text{OPT}_{>}}S) \geq \frac{1}{4}w(\cup_{S\in\text{OPT}}S)$, or (ii) $w(\cup_{S\in\text{OPT}_{\leq}}S) \geq \frac{3}{4}w(\cup_{S\in\text{OPT}}S)$.

If $w(\cup_{S\in\text{OPT}_{>}}S) \geq \frac{1}{4}w(\cup_{S\in\text{OPT}}S)$, then there exists a single set in $\text{OPT}_{>}$ whose weight is at least $\frac{1}{8}w(\cup_{S\in\text{OPT}}S)$. In this case, the algorithm that keeps a single set with the maximum-weight coverage seen so far obtains a solution with weight at least $\frac{1}{8}w(\cup_{S\in\text{OPT}}S)$.

If $w(\cup_{S\in\text{OPT}_{\leq}}S) \geq \frac{3}{4}w(\cup_{S\in\text{OPT}}S)$, then α-greedy, for $\alpha = 2$, which runs only on the sets with cost at most $\frac{1}{3}$ is 6-competitive, and therefore it computes a solution whose weight is at least $\frac{1}{6}w(\cup_{S\in\text{OPT}_{\leq}}S) \geq \frac{1}{6}\cdot\frac{3}{4}w(\cup_{S\in\text{OPT}}S) = \frac{1}{8}w(\cup_{S\in\text{OPT}}S)$.

Since the algorithm chooses with equal probability $\frac{1}{2}$ each one of the two algorithms, we have that $\mathbb{E}[\text{ALG}] \geq \frac{1}{2}\cdot\frac{1}{8}\cdot w(\cup_{S\in\text{OPT}}S) = \frac{1}{16}w(\cup_{S\in\text{OPT}}S)$. ◀

5 $O(\Delta)$-competitive Algorithm

In this section we present an $O(\Delta)$-competitive algorithm for OBMC. Given an OBMC instance we define a bipartite graph $G = (\mathcal{S}, U, E)$, where $(S, u) \in E$ if and only if $u \in S$. Given a collection of sets $\mathcal{S}'$, let $G[\mathcal{S}']$ be the subgraph of G that is induced by $\mathcal{S}'$ and U. The algorithm is based on computing maximum cardinality matchings between sets ($\mathcal{S}'$, the left side of G) and elements (U, the right side of G). Let **MaxMatch** be an algorithm that solves the MAXIMUM CARDINALITY MATCHING problem in bipartite graphs.

The OBMC algorithm works as follows. Upon arrival of a set S_i, $i \geq 1$, the algorithm constructs a solution $\mathcal{S}_i$ using the previous solution $\mathcal{S}_{i-1}$ (we initialize $\mathcal{S}_0 = \emptyset$). The algorithm looks for an element to be matched to S_i, where S_i takes precedence over sets that cost more than $c(S_i)$. This is done as follows. First a maximum matching is computed for the collection of sets that includes those sets in the already-computed solution that have cost at most $c(S_i)$, and S_i. If it is impossible to match all these sets, or their total cost exceeds 1, S_i is *rejected*, and the current solution remains unchanged. Otherwise S_i is *accepted*, and all the sets in the already-computed solution that have cost at most $c(S_i)$ are not dropped. In the latter case the algorithm then tries to extend the matching by assigning an element to those sets in $\mathcal{S}_{i-1}$ that cost more than $c(S_i)$. Such a set S_j is *dropped* if a matching cannot be obtained or if the total cost exceeds 1. Algorithm 3 shows a formal pseudo-code of this algorithm.

In what follows ALG denotes the collection of sets that is output by the algorithm. We first prove that the solution is feasible.

▶ **Lemma 15.** $c(\text{ALG}) \leq 1$.

Proof. We prove by induction on i that after each iteration we have $c(\mathcal{S}_i) \leq 1$. The base case is trivial, since $\mathcal{S}_0 = \emptyset$. For the inductive step, assume that $c(\mathcal{S}_{i-1}) \leq 1$. $\mathcal{S}_i$ is initially

Algorithm 3: Match; operations when set S_i arrives.

```
1  $\mathcal{S}_i \leftarrow \{S \in \mathcal{S}_{i-1} : c(S) \leq c(S_i)\}$
2  $M \leftarrow \text{MaxMatch}(G[\mathcal{S}_i \cup \{S_i\}])$
3  if $|M| = |\mathcal{S}_i \cup \{S_i\}|$ then
4      if $c(\mathcal{S}_i \cup \{S_i\}) \leq 1$ then
5          $\mathcal{S}_i \leftarrow \mathcal{S}_i \cup \{S_i\}$; $M_i \leftarrow M$
6          $\mathcal{S}'_i \leftarrow \mathcal{S}_{i-1} \setminus \mathcal{S}_i$
7          while $\mathcal{S}'_i \neq \emptyset$ do
8              $j \leftarrow \text{argmin}_j \{c(S_j) : S_j \in \mathcal{S}'_i\}$
9              $\mathcal{S}'_i \leftarrow \mathcal{S}'_i \setminus \{S_j\}$
10             $M \leftarrow \text{MaxMatch}(G[\mathcal{S}_i \cup \{S_j\}])$
11             if $|M| = |\mathcal{S}_i \cup \{S_j\}|$ then
12                 if $c(\mathcal{S}_i \cup \{S_j\}) \leq 1$ then
13                     $\mathcal{S}_i \leftarrow \mathcal{S}_i \cup \{S_j\}$; $M_i \leftarrow M$
14     else
15         $\mathcal{S}_i \leftarrow \mathcal{S}_{i-1}$; $M_i \leftarrow M_{i-1}$
16 else
17     $\mathcal{S}_i \leftarrow \mathcal{S}_{i-1}$; $M_i \leftarrow M_{i-1}$
```

a subset of $\mathcal{S}_{i-1}$ (Line 1) and therefore it is feasible at this stage due to the inductive hypothesis. Furthermore, we add sets to $\mathcal{S}_i$ only if the budget constraint is not violated (Lines 4 and 12). ◀

We now show that the number of sets in the solution never decreases, and may increase by at most one set in any step (proof omitted from this extended abstract).

▶ **Lemma 16.** $|\mathcal{S}_{i-1}| \leq |\mathcal{S}_i| \leq |\mathcal{S}_{i-1}| + 1$, *for every* $i > 0$.

Since $|\text{ALG}| = |M_m|$ and $|M_m| \leq |\cup_{S \in \text{ALG}} S|$ we have the following observation.

▶ **Observation 17.** $|\cup_{S \in \text{ALG}} S| \geq |\text{ALG}|$.

Let OPT denote an optimal collection of sets. We partition OPT $\setminus$ ALG into two collections. Let τ be the cost of the cheapest set that was either rejected upon arrival due to the budget constraint, or dropped later due to the budget constraint. More formally, define

$$\tau_1 = \min\{\{c(S_i) : S_i \text{ was rejected by Line 4 at the } i\text{th iteration}\} \cup \{\infty\}\} ,$$
$$\tau_2 = \min\{\{c(S_i) : S_i \text{ was dropped by Line 12 at the } j\text{th iteration}\} \cup \{\infty\}\} ,$$
$$\tau = \min\{\tau_1, \tau_2\} ,$$

and define

$$\text{OPT}' = \{S \in \text{OPT} \setminus \text{ALG} : c(S) < \tau\} \quad \text{and} \quad \text{OPT}'' = \{S \in \text{OPT} \setminus \text{ALG} : c(S) \geq \tau\} .$$

The next two lemmas essentially give the upper bound on the competitive ratio of the algorithm. Their proofs are omitted from this extended abstract. The first lemma shows that the elements that are covered by the sets in OPT′ are covered by the sets in ALG. The second one shows that the size of OPT″ is bounded from above by the size of ALG.

▶ **Lemma 18.** $\cup_{S\in\text{OPT}'} S \subseteq \cup_{S\in\text{ALG}} S$.

▶ **Lemma 19.** $|\text{OPT}''| \le |\text{ALG}|$.

It remains to give an upper bound on the competitive ratio of Algorithm **Match**.

▶ **Theorem 20.** *Algorithm* ***Match*** *is* $(\Delta + 2)$*-competitive.*

Proof. We have that

$$\begin{aligned} w(\cup_{S\in\text{OPT}} S) &\le w(\cup_{S\in\text{OPT}\cap\text{ALG}} S) + w(\cup_{S\in\text{OPT}'} S) + w(\cup_{S\in\text{OPT}''} S) \\ &\le 2\cdot w(\cup_{S\in\text{ALG}} S) + w(\cup_{S\in\text{OPT}''} S) && (1) \\ &\le 2\cdot w(\cup_{S\in\text{ALG}} S) + |\text{OPT}''|\cdot\Delta && (2) \\ &\le 2\cdot w(\cup_{S\in\text{ALG}} S) + |\text{ALG}|\cdot\Delta && (3) \\ &\le (\Delta+2)\cdot w(\cup_{S\in\text{ALG}} S)\ , && (4) \end{aligned}$$

where (1) follows from Lemma 18, (2) follows from the assumption that the weight of an element is at least 1, (3) is due to Lemma 19, and (4) is due to Observation 17 and the above assumption. ◀

5.1 $w(U)$ is known in advance

We obtain a deterministic algorithm by running algorithm **Match** with the additional rule that if a set S_i with $w(S_i) \ge \sqrt{w(U)}$ arrives, then we drop all currently taken sets, take set S_i, and never change further the solution (i.e., the final output solution is $\{S_i\}$).

Observe that if there exists a set $S_i \in \mathcal{S}$ with $w(S_i) \ge \sqrt{w(U)}$, then the total weight of the elements in the sets of ALG is at least $\sqrt{w(U)}$, while the total weight of the elements in the sets of OPT is at most $w(U)$. If no such set exists then ALG is equivalent to **Match** which is $(\Delta + 2) < (\sqrt{w(U)} + 2)$-competitive. This leads to the following result.

▶ **Theorem 21.** *There exists a deterministic online algorithm whose competitive ratio is* $O(\min\{\Delta, \sqrt{w(U)}\}$*, provided that* $w(U)$ *is known in advance.*

Note that for the unit-weight case $w(U)$ equals n, and we thus get an $O(\min\{\Delta, \sqrt{n}\})$-competitive deterministic algorithm for this case.

6 Conclusions

We have studied the Online Budgeted Maximum Coverage problem. We presented a deterministic online algorithms in terms of three parameters of the given instance, and we gave deterministic lower bounds bases on these parameters. We also provided a randomized $O(1)$-competitive algorithm. Finally, both our upper and lower bounds on the deterministic competitive ratio apply to Removable Online Knapsack which is the preemptive version of the Online Knapsack problem.

We briefly mention some possible future research directions. Obvious open problems are closing the gap between the deterministic upper and lower bounds ($O(\frac{1}{1-r})$ vs. $\Omega(\frac{1}{\sqrt{1-r}})$), and decreasing the (constant) randomized competitive ratio. Other interesting goals would be to design an $O(\sqrt{w(U)})$-competitive deterministic algorithm that does not require advance knowledge of $w(U)$, and devising a single deterministic algorithm that obtains as a competitive ratio the minimum of all deterministic competitive ratios shown in this paper.

References

1 Alexander A. Ageev and Maxim Sviridenko. Pipage rounding: A new method of constructing algorithms with proven performance guarantee. *Journal of Combinatorial Optimization*, 8(3):307–328, 2004.

2 Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Naor. The online set cover problem. *SIAM Journal on Computing*, 39(2):361–370, 2009.

3 Noga Alon, Dana Moshkovitz, and Shmuel Safra. Algorithmic construction of sets for k-restrictions. *ACM Transactions on Algorithms*, 2(2):153–177, 2006.

4 Giorgio Ausiello, Nicolas Boria, Aristotelis Giannakos, Giorgio Lucarelli, and Vangelis Th. Paschos. Online maximum k-coverage. *Discrete Applied Mathematics*, 160(13-14):1901–1913, 2012.

5 Baruch Awerbuch, Yossi Azar, Amos Fiat, and Frank Thomson Leighton. Making commitments in the face of uncertainty: How to pick a winner almost every time. In *28th Annual ACM Symposium on the Theory of Computing*, pages 519–530, 1996.

6 R. Bar-Yehuda and S. Even. A linear time approximation algorithm for the weighted vertex cover problem. *Journal of Algorithms*, 2:198–203, 1981.

7 Oded Berman, Dimitris Bertsimas, and Richard C. Larson. Locating discretionary service facilities, ii: Maximizing market size, minimizing inconvenience. *Operations Research*, 43(4):623–632, 1995.

8 Oded Berman, Richard C. Larson, and Nikoletta Fouska. Optimal location of discretionary service facilities. *Transportation Science*, 26(4):201–611, 1992.

9 Niv Buchbinder, Moran Feldman, Joseph Naor, and Roy Schwartz. Submodular maximization with cardinality constraints. In *25th Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 1433–1452, 2014.

10 Niv Buchbinder, Moran Feldman, and Roy Schwartz. Online submodular maximization with preemption. In *26th Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 1202–1216, 2015.

11 Niv Buchbinder and Joseph Naor. Online primal-dual algorithms for covering and packing. *Math. Oper. Res.*, 34(2):270–286, 2009.

12 Amit Chakrabarti and Sagar Kale. Submodular maximization meets streaming: Matchings, matroids, and more. In *17th Intl. Conference on Integer Programming and Combinatorial Optimization*, volume 8494 of *LNCS*, pages 210–221, 2014.

13 V. Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

14 Marek Cygan, Lukasz Jeż, and Jiri Sgall. Online knapsack revisited. *Theory of Computing Systems*, 2016. To appear.

15 Marc Demange and Vangelis Th. Paschos. On-line vertex-covering. *Theoretical Computer Science*, 332(1–3):83–108, 2005.

16 I. Dinur, V. Guruswami, S. Khot, and O. Regev. A new multilayered PCP and the hardness of hypergraph vertex cover. *SIAM Journal on Computing*, 34(5):1129–1146, 2005.

17 Irit Dinur and Shmuel Safra. The importance of being biased. In *34th Annual ACM Symposium on the Theory of Computing*, pages 33–42, 2002.

18 Uriel Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.

19 M. L. Fisher, George L. Nemhauser, and Laurence A. Wolsey. Analysis of approximation algorithms for maximizing submodular set function II. *Mathematical Programming Study*, 8:73–87, 1978.

20 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, San Francisco, 1979.

21 Xin Han, Yasushi Kawase, and Kazuhisa Makino. Randomized algorithms for online knapsack problems. *Theoretical Computer Science*, 562:395–405, 2015.

22 Dorit S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal on Computing*, 11(3):555–556, 1982.

23 Dorit S. Hochbaum, editor. *Approximation Algorithms for NP-Hard Problem*. PWS Publishing Company, 1997.

24 Dorit S. Hochbaum and Anu Pathria. Analysis of the greedy approach in problems of maximum k-coverage. *Naval Research Logistics*, 45(6):615–627, 1998.

25 Oscar H. Ibarra and Chul E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM*, 22(4):463–468, 1975.

26 Kazuo Iwama and Shiro Taketomi. Removable online knapsack problems. In *29th Annual Intl. Colloquium on Automata, Languages and Programming*, volume 2380 of *LNCS*, pages 293–305, 2002.

27 D. S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.

28 Samir Khuller, Anna Moss, and Joseph Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70(1):39–45, 1999.

29 Dennis Komm, Richard Královic, and Tobias Mömke. On the advice complexity of the set cover problem. In *7th International Computer Science Symposium in Russia*, pages 241–252, 2012.

30 L. Lovász. On the ratio of optimal integeral and fractional solutions. *Discrete Mathematics*, 13:383–390, 1975.

31 Alberto Marchetti-Spaccamela and Carlo Vercellis. Stochastic on-line knapsack problems. *Math. Program.*, 68:73–104, 1995.

32 N. Megiddo, E. Zemel, and S. L. Hakimi. The maximum coverage location problem. *SIAM Journal on Algebraic and Discrete Methods*, 4(2):253–261, 1983.

33 George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, Inc., 1988.

34 George L. Nemhauser, Laurence A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions I. *Mathematical Programming*, 14(1):265–294, 1978.

35 Ran Raz and Shmuel Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *29th Annual ACM Symposium on the Theory of Computing*, pages 475–484, 1997.

36 Barna Saha and Lise Getoor. On maximum coverage in the streaming model & application to multi-topic blog-watch. In *SIAM International Conference on Data Mining*, pages 697–708, 2009.

37 Sartaj Sahni. Approximate algorithms for the 0/1 knapsack problem. *Journal of the ACM*, 22(1):115–124, 1975.

38 Maxim Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Operations Research Letters*, 32(1):41–43, 2004.

39 Ashwinkumar Badanidiyuru Varadaraja. Buyback problem - approximate matroid intersection with cancellation costs. In *38th Annual Intl. Colloquium on Automata, Languages and Programming*, volume 6755 of *LNCS*, pages 379–390, 2011.

40 Vijay V. Vazirani. *Approximation Algorithms*. Springer-Verlag, Berlin Heidelberg New York, 2001.

41 Yunhong Zhou, Deeparnab Chakrabarty, and Rajan M. Lukose. Budget constrained bidding in keyword auctions and online knapsack problems. In *4th international Workshop on Internet and Network Economics*, volume 5385 of *LNCS*, pages 566–576, 2008.

Min-Sum Scheduling Under Precedence Constraints[*]

Andreas S. Schulz[1] and José Verschae[2]

1 Departments of Mathematics and Economics, TU Munich, Munich, Germany
andreas.s.schulz@tum.de

2 Department of Mathematics & School of Engineering, Pontifical Catholic University of Chile, Santiago, Chile
jverschae@uc.cl

Abstract

In many scheduling situations, it is important to consider non-linear functions of job completions times in the objective. This was already recognized by Smith (1956). Recently, the theory community has begun a thorough study of the resulting problems, mostly on single-machine instances for which all permutations of jobs are feasible. However, a typical feature of many scheduling problems is that some jobs can only be processed after others. In this paper, we give the first approximation algorithms for min-sum scheduling with (nonnegative, non-decreasing) non-linear functions and general precedence constraints. In particular, for $1|\text{prec}|\sum w_j f(C_j)$, we propose a polynomial-time universal algorithm that performs well for all functions f simultaneously. Its approximation guarantee is 2 for all concave functions, at worst. We also provide a (non-universal) polynomial-time algorithm for the more general case $1|\text{prec}|\sum f_j(C_j)$. The performance guarantee is no worse than $2+\epsilon$ for all concave functions. Our results match the best bounds known for the case of linear functions, a widely studied problem, and considerably extend the results for minimizing $\sum w_j f(C_j)$ without precedence constraints.

1998 ACM Subject Classification F.2.2 Sequencing and scheduling

Keywords and phrases scheduling; approximation algorithms; linear programming relaxations; precedence constraints

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.74

1 Introduction

We consider a single-machine scheduling problem with non-linear objective function under precedence constraints. Let $f : \mathbb{R}_+ \to \mathbb{R}_+$ be a non-decreasing cost function. Given a set J of n jobs, where each job $j \in J$ is characterized by a weight $w_j \geq 0$ and an integral processing time $p_j \geq 0$, our goal is to find a sequence of the jobs that minimizes $\sum_j w_j f(C_j)$. Here, C_j denotes the completion time of job j in the corresponding nonpreemptive schedule. Additionally, we consider a set of precedence constraints $P \subseteq J \times J$. Now, $(i,j) \in P$ implies that job i must be completed before j can start. Keeping to the standard three-field notation [11], this problem can be denoted by $1|\text{prec}|\sum w_j f(C_j)$.

Our main result is that there exists a *universal* schedule, i.e., a feasible sequence that depends only on (p_j), (w_j) and P (but not on f), which performs well for all f together.

[*] This work was partially supported by FONDECYT project 3130407, by Núcleo Milenio Información y Coordinación en Redes ICM/FIC RC130003, and by the Alexander von Humboldt-Foundation.

This sequence can be computed in polynomial time, and its performance guarantee is

$$\sup_{C\geq 0} \frac{C \cdot f(C)}{\int_0^C f(t)dt} \ .$$

For concave f, the approximation guarantee can be bounded by 2 with a simple geometric argument. It is worth mentioning that for a vast class of concave functions the approximation guarantee is strictly better than 2, which improves upon the best possible[1] 2-approximation for the linear case.

One somewhat surprising lesson of our study is that an important concept from classic work on the strongly NP-hard problem $1|\text{prec}|\sum w_j C_j$ still holds considerable value for the more general problem $1|\text{prec}|\sum w_j f(C_j)$. In his PhD thesis, Sidney introduced a (polynomial-time computable) decomposition of J into mutually disjoint subsets $J_1, J_2, \ldots, J_k$ and showed that there always exists an optimal solution that follows this order [21]. Chekuri and Motwani [4] as well as Margot, Queyranne and Wang [16] later realized that any feasible sequence that is consistent with Sidney's decomposition is a 2-approximation. This observation was preceded by a number of linear programming based 2-approximation algorithms [19, 12, 6]. Correa and Schulz subsequently proved that virtually all known 2-approximation algorithms are of the Chekuri and Motwani/Margot, Queyranne and Wang type; in fact, several common linear programming relaxations follow Sidney's decomposition [7].

We analyze the same sequence as Chekuri and Motwani/Margot, Queyranne and Wang, but for $1|\text{prec}|\sum w_j f(C_j)$. In contrast to the case of linear f, it is not true that one of the sequences following Sidney's decomposition is optimal, which requires us to devise a very different, more complicated analysis. Still, the algorithm is the same, and our analysis implies that its performance depends on a simple geometric ratio defined by the shape of f. For all concave f, this ratio is at most 2 – the same bound that was earlier observed for linear functions f.

Our main technique relies on analyzing a time-indexed LP relaxation that is intimately related to the *partially ordered knapsack* problem (POK) and its fractional relaxation. In POK we are given a set of items J with weights and values, and a knapsack with a given capacity t. The items have precedence constraints $P \subseteq J \times J$, such that if $(i, j) \in P$ and we pack j into the knapsack, also i must be packed. The total weight of the packing cannot exceed t, and the objective is to maximize the total value. In its fractional version, we are allowed to take fractions of the jobs. If a fraction $x_j \in [0, 1]$ is packed into the knapsack and $(i, j) \in P$, then a fraction of i at least as large as x_j has to be packed. POK and its relaxation was previously studied by Kolliopoulos and Steiner [15], who derived an FPTAS for 2-dimensional precedence constraints and characterized cases where the natural LP relaxation has bounded integrality gap. Our main technical contribution is to show that Sidney's decomposition implies an optimal solution for fractional POK for each t. This implies that the solution is optimal for the time-indexed relaxation, *independently* of the cost function f, which in turn allow us to derive the approximation ratio.

Sidney's decomposition and corresponding algorithm can be viewed as an extension of Smith's optimal WSPT rule for $1||\sum w_j C_j$, which sequences jobs in order of non-increasing ratios of weight to processing time, to $1|\text{prec}|\sum w_j C_j$. In this sense, our work also generalizes earlier contributions by Stiller and Wiese [23] and Höhn and Jacobs [13] for $1||\sum w_j f(C_j)$, to instances with precedence constraints. For arbitrary concave f, Stiller and Wiese showed that Smith's rule guarantees an approximation factor of $(\sqrt{3} + 1)/2 \approx 1.366$. Höhn and

[1] Assuming a stronger version of the Unique Games Conjecture [1].

Jacobs built on their analysis to obtain refined and tight bounds of Smith's rule for any specific concave or convex function f. Unlike our LP-based analysis, their techniques rely on identifying a worst-case instance, which can be shown to only contain jobs with the same weight to processing time ratio. The argument then exploits the fact that for concave or convex functions it is easy to identify the worst WSPT solution and the optimal value. For general functions f, Im, Moseley and Pruhs [14] proved that Smith's rule is a $(2+\epsilon)$-speed O(1)-approximate algorithm. Epstein et al. [8] also gave a universal algorithm (quite different from Smith's rule, based on earlier work by Hall et al. [12]), which has performance guarantee $4+\epsilon$, for any cost function f. Additionally, they derived a randomized version of their algorithm with performance guarantee $e+\epsilon$, also for any f. As for non-universal algorithms, Megow and Verschae designed a PTAS for $1| \,|\sum w_j f(C_j)$ [17] for any given function f.

The entire area of min-sum scheduling with non-linear functions of the completion times was arguably revived by Bansal and Pruhs ([2], see also [3]). They considered the more general objective $\sum_j f_j(F_j)$, where f_j is a (nonnegative, non-decreasing) job-dependent cost function, $F_j - r_j$ is the flow time of job j, and r_j its release date. In case $r_j = 0$ for all jobs j, i.e., the setting considered here, their algorithm has performance guarantee 16. Subsequently, a better primal-dual algorithm was given with approximation ratio $4+\epsilon$ [5, 18].

Here, we also give the first polynomial-time approximation algorithm for $1|\text{prec}|\sum f_j(C_j)$, i.e., the case of job-dependent non-linear functions and general precedence constraints; see Section 3. Its approximation guarantee is at most $2+\epsilon$ for concave functions. The algorithm relies on solving a time-indexed LP relaxation and then rounding the fractional solution using randomized α-points. This type of rounding has been extensively used for the sum of weighted completion times objective (see, e.g., [22] for a summary); to the best of our knowledge this is the first time it is applied to a non-linear objective function.

2 A universal algorithm

In this section, we show that the 2-approximation algorithm for the linear case by Margot et al. [16] and Chekuri and Motwani [4] yields the following theorem.

▶ **Theorem 1.** *For any non-decreasing function $f : \mathbb{R}_+ \to \mathbb{R}_+$, the problem $1|prec|\sum w_j f(C_j)$ admits a polynomial-time, purely combinatorial algorithm with approximation guarantee*

$$\Gamma_f := \sup_{C\geq 0} \frac{C \cdot f(C)}{\int_0^C f(t)dt} \ .$$

Moreover, the solution is universal, *that is, it is independent of the cost function f.*

We say that a set of jobs $I \subset J$ is an *ideal* or *initial set* if it is a feasible prefix of a schedule; that is, for any $j \in I$, if $(i,j) \in P$, then $i \in I$. Also, if an ideal I maximizes $w(I)/p(I)$ over all ideals, we say that I is a *density maximal* ideal. Here, $w(I) = \sum_{j\in I} w_j$, and $p(I)$ is defined similarly. We assume that precedence constraints are given by a precedence digraph $G = (J,P)$, with jobs as nodes and arcs given by P. For a given set of jobs K, we denote by G_K the graph induced by set K. A Sidney decomposition is a collection of sets $J_1, J_2, \ldots, J_\ell$ such that J_i is a density maximal ideal in $G_{(J\setminus\cup_{k<i}J_k)}$. The problem of computing a Sidney decomposition can be seen as a parametric network flow problem, and thus it can be computed efficiently [16].

The algorithm computes a Sidney decomposition $J_1, J_2, \ldots, J_\ell$ of set J, and creates a schedule that processes all jobs in the order given by the decomposition; that is, if $j \in J_i, k \in J_s$ and $i < s$, then j is processed before k. Within each set J_i we pick an arbitrary linear ordering of the jobs that is consistent with the precedence constraints. To analyze this algorithm we use a time-indexed linear programming relaxation and find an explicit optimal solution that follows a Sidney decomposition. This is the core of our analysis.

A preemptive relaxation. Unlike in the classic setting, i.e., $1|\text{prec}|\sum w_j C_j$, for non-linear cost functions we cannot necessarily find an optimal solution that follows a Sidney decomposition. Indeed, in the absence of precedence constraints a Sidney decomposition corresponds to a solution given by Smith's rule, and such a solution is not necessarily optimal for non-linear f [13]. However, we will show that for a relaxed version of our problem a Sidney decomposition indeed gives an optimal solution. The relaxation considers preemptive solutions. To model precedence constraints in this relaxation we use the concept of *fractional precedence constraints*; that is, for $(i, j) \in P$ we impose that for any time t the fraction of job i processed in $[0, t]$ is as least as large as that of j.

Let $T = \sum_j p_j$ be the time horizon (assuming, w.l.o.g., that there is no idle-time). In order to avoid discretizing time, or making unnecessary assumptions on the structure of optimal solutions, we describe an arbitrary preemptive solution as a family of non-decreasing piecewise linear functions $x_j : [0, T] \to [0, 1]$ for each $j \in J$, where $x_j(t)$ represents the fraction of job j that is processed up to time t. In our relaxation, a job can be split in small pieces, each with the same density $\rho_j = w_j/p_j$ as the original job j. If a small piece of (infinitesimal) processing time δ finishes at time t, in our relaxation this incurs a cost of $\delta\rho_j f(t)$. Equivalently, function x_j increases by $x_j(t + \delta) - x_j(t) = \delta/p_j$ and we incur a cost of $w_j \cdot f(t) \cdot (x_j(t + \delta) - x_j(t))$. With this in mind, we define the cost of a preemptive solution as

$$\sum_{j\in J} w_j \cdot \int_0^T f(t)x_j'(t)dt,$$

where x_j' is the derivative of x_j, which is defined almost everywhere, except for the breakpoints of the function. More formally, our relaxation is given by the following optimization problem.

$$[\text{Rel}] \quad \textsc{Opt}_f = \inf \sum_{j\in J} w_j \cdot \int_0^T f(t)x_j'(t)dt$$

$$\text{s.t. } x_j(T) = 1 \qquad \text{for all } j \in J, \quad (1)$$

$$\sum_{j\in J} x_j(t)p_j \le t \qquad \text{for all } t \in [0, T], \quad (2)$$

$$x_j(t) \ge x_k(t) \qquad \text{for all } (j, k) \in P, t \in [0, T] \quad (3)$$

$$x_j : [0, T] \to \mathbb{R}_+ \qquad \text{non-decreas., piecew. lin. for all } j \in J. \quad (4)$$

We call any solution $(x_j)_{j\in J}$ that satisfies all conditions of this problem a *preemptive* schedule.

Let $\textsc{Opt}$ be the optimal value of our original, non-preemptive, problem. It is not hard to see that the optimal value of this program is a lower bound on $\textsc{Opt}$. Indeed, let $\mathcal{S}$ be an arbitrary non-preemptive schedule, and let S_j and $C_j = S_j + p_j$ be the starting time and completion time of job j in that solution, respectively. Naturally, we define x_j as a piecewise linear function

$$x_j(t) = \begin{cases} 0 & \text{if } t \le S_j, \\ \frac{t-S_j}{p_j} & \text{if } S_j < t < C_j, \\ 1 & \text{if } t \ge C_j. \end{cases} \quad (5)$$

It is easy to see that this definition yields a feasible solution to [Rel]. Moreover, the objective function can be upper bounded as follows,

$$\sum_{j\in J} w_j \cdot \int_0^T f(t)x_j'(t)dt = \sum_{j\in J} w_j \cdot \int_{S_j}^{C_j} \frac{1}{p_j} f(t)dt \leq \sum_{j\in J} w_j \cdot f(C_j), \tag{6}$$

where the equality holds since x_j has non-zero slope only in (S_j, C_j), where its derivative equals $1/p_j$. The last inequality follows since f is non-decreasing. We conclude that $\text{OPT}_f \leq \text{OPT}$.

An explicit optimal fractional solution. In what follows we explicitly find an optimal solution to [Rel]. Indeed, let $J_1, \dots, J_\ell$ be a Sidney decomposition of the instance. For any set J_i we define its *starting time* by $S(J_i) = \sum_{\ell<i} p(J_\ell)$ and its *completion time* as $C(J_i) = S(J_i) + p(J_i)$. For any given $j \in J_i$, we define

$$x_j^*(t) = \begin{cases} 0 & \text{if } t \leq S(J_i), \\ \frac{t-S(J_i)}{p(J_i)} & \text{if } S(J_i) < t < C(J_i), \\ 1 & \text{if } t \geq C(J_i). \end{cases}$$

We will show that this solution is optimal for [Rel]. Notice that x^* does not depend on f, which is why we can derive a universal schedule. Our solution can be interpreted as a schedule in which we first process all jobs in J_1, and within J_1 we process all jobs in a round robin fashion: we first schedule an infinitesimal fraction δ of each job in J_1, the same fraction for each job, ordering the fractions within J_1 according to any feasible linear ordering. With this we are able to process a set of jobs with density $w(J_1)/p(J_1)$, which gives us the most bang for the buck since J_1 is a density maximal ideal. This operation is repeated until J_1 is completely processed. Then we continue for each J_i for $i = 2, \dots, k$ iteratively with the same strategy. It is straightforward to check that the constructed solution is feasible to [Rel].

To establish that x^* is optimal, we show that it simultaneously solves a family of *fractional partially ordered knapsack problems*. For a given $t \in [0, T]$ we consider the linear program,

$$\begin{aligned} [\text{FK}(t)] \qquad & \max \quad \sum_{j\in J} x_j w_j && && (7)\\ & \text{s.t.} \quad \sum_{j\in J} x_j p_j \leq t && && (8)\\ & \qquad x_j \geq x_k && \text{for all } (j,k) \in P, && (9)\\ & \qquad 0 \leq x_j \leq 1 && \text{for all } j \in J. && (10) \end{aligned}$$

We show that, for a fixed t, the vector $(x_j^*(t))_{j\in J}$ yields an optimal solution to [FK(t)]. To do so we first start by characterizing the structure of feasible solutions to this LP. For a given set $S \subseteq J$ we denote by χ^S the indicator vector of set S, that is, $\chi_j^S = 1$ if $j \in S$ and $\chi_j^S = 0$ otherwise.

▶ **Lemma 2** (Fractional Decomposition). *Let z be any feasible solution to [FK(t)]. Then there exist r sets $A_1, \dots, A_r$ and numbers $1 \geq \gamma_1 > \dots > \gamma_r > 0$ such that*
1. $z = \sum_{i=1}^r \gamma_i \cdot \chi^{A_i}$,
2. *for all $s \in \{1, \dots, r\}$ the set $\cup_{i\leq s} A_i$ is an ideal,*
3. $\sum_{i=1}^r \gamma_i p(A_i) \leq t$, *and*
4. $A_i \cap A_\ell = \emptyset$ *for all $i, \ell \in J$ with $i \neq \ell$.*

Proof. Consider the set $Z = \{z_j \neq 0 : j \in J\}$ and assume that $Z = \{\gamma_1, \ldots, \gamma_r\}$ with $\gamma_1 > \gamma_2 > \ldots > \gamma_r > 0$. Define $A_i = \{j : z_j = \gamma_i\}$. Property 4 is obvious. Property 1 follows by 4 and the definition of A_i. Property 3 follows from Property 1 and Inequality (8) of [FK(t)]. To show Property 2, notice that $\cup_{i \leq s} A_i = \{j \in J : z_j \geq \gamma_s\}$. Consider an arbitrary job $k \in \cup_{i \leq s} A_i$, so that $z_k \geq \gamma_s$. We need to show that for any $j \in J$ with $(j, k) \in P$, it holds that $z_j \geq \gamma_s$. This follows because by Inequality (9) we have that $z_j \geq z_k \geq \gamma_s$. ◀

For a given feasible solution z, we call the sets $A_1, \ldots, A_r$ and numbers $\gamma_1, \ldots, \gamma_r$ given by the lemma a *fractional decomposition* of z.

▶ **Lemma 3** (Extremal Fractional Decomposition). *For any feasible solution z to [FK(t)] there exists another feasible solution $z' = \chi^{A_1} + \gamma \cdot \chi^{A_2}$ such that*
1. $\sum_j z_j w_j \leq \sum_j z'_j w_j$*,*
2. A_1 *and* $A_1 \cup A_2$ *are ideals,*
3. $p(A_1) + \gamma \cdot p(A_2) \leq t$*,*
4. $A_1 \cap A_2 = \emptyset$*.*

Proof. For a given feasible solution z, consider its fractional decomposition given by sets $A'_1, \ldots, A'_r$ and numbers $\gamma_1 > \ldots > \gamma_r$. To show the lemma we write an LP to optimize over the weights γ. We use β_i for the variables representing the weights γ_i.

$$\begin{aligned} \max \quad & \sum_{i=1}^{r} \beta_i w(A'_i) \\ \text{s.t.} \quad & \sum_{i=1}^{r} \beta_i p(A'_i) \leq t, \\ & 1 \geq \beta_1 \geq \ldots \geq \beta_r \geq 0. \end{aligned}$$

Let β^* be an extreme point optimal solution to this LP. Since there are $r + 2$ inequalities and r variables, at most 2 inequalities in $1 \geq \beta_1 \geq \ldots \geq \beta_r \geq 0$ are not satisfied with equality. Let us first assume that β^* is not an integral solution, and let ℓ be the smallest index such that $\beta^*_\ell \in (0, 1)$. This already induces one strict inequality $\beta^*_\ell < \beta^*_{\ell-1} = 1$. Let $\gamma = \beta^*_\ell$. There must exists an index $u \in \{\ell, \ldots, r\}$ such that $\beta^*_i = \gamma$ for all $i \in \{\ell, \ldots, u\}$ and $\beta^*_i = 0$ for all $i > u$, because otherwise there would be in total 3 strict inequalities. We get the following property: there exist a number $\gamma \in [0, 1]$ and two indices $1 \leq \ell \leq u \leq r$ such that $\beta^*_i = 1$ if $i < \ell$, $\beta^*_i = \gamma$ if $i \in \{\ell, \ldots, u\}$, and $\beta^*_i = 0$ if $i > u$. This property holds also if β^* is integral by taking $\gamma = 0$.

Now we can define $A_1 = \cup_{i=1}^{\ell-1} A'_i$, $A_2 = \cup_{i=\ell}^{u} A'_i$ and $\gamma = \beta^*_\ell$. Property 1 follows since β^* is optimal for the LP and setting $\beta_i = \gamma_i$ for all i yields a feasible solution. Properties 2 and 4 hold because of the Fractional Decomposition Lemma. Finally, Property 3 is implied by the first inequality of the LP and since the sets A'_i are pairwise disjoint. ◀

The next lemma shows that, if we are given a density maximal ideal $I \subset J$ such that $t \leq p(I)$, then an optimal solution of [FK(t)] can be constructed by taking each job in I with a fraction of $\frac{t}{p(I)}$. A similar statement was given by Kolliopoulos and Steiner [15], although they used a different proof technique.

▶ **Lemma 4.** *Given any density maximal ideal $I \subseteq J$, then for any $t \leq p(I)$ the vector $x = \frac{t}{p(I)} \chi^I$ is an optimal solution to [FK(t)].*

Proof. Let x be as defined in the statement of the lemma. It is clear that x is a feasible solution to $[FK(t)]$. Consider any optimal solution to [FK(t)] which, by the Extremal

Fractional Decompostion Lemma, can be taken as $\chi^{A_1} + \gamma \cdot \chi^{A_2}$. Recalling that A_1 and $A_1 \cup A_2$ are ideals, and that $A_1 \cap A_2 = \emptyset$, the difference in objective values of the two solutions is given by

$$\begin{aligned}\frac{t}{p(I)}w(I) - (w(A_1) + \gamma w(A_2)) &= \frac{t}{p(I)}w(I) - (\gamma w(A_1 \cup A_2) + (1-\gamma)w(A_1))\\ &= \gamma\left(\frac{t}{p(I)}w(I) - w(A_1 \cup A_2)\right) + (1-\gamma)\left(\frac{t}{p(I)}w(I) - w(A_1)\right)\\ &= \gamma\left(\frac{w(I)}{p(I)}t - \frac{w(A_1 \cup A_2)}{p(A_1 \cup A_2)}p(A_1 \cup A_2)\right) + (1-\gamma)\left(\frac{w(I)}{p(I)}t - \frac{w(A_1)}{p(A_1)}p(A_1)\right)\\ &\geq \frac{w(I)}{p(I)}\left\{\gamma \cdot (t - p(A_1 \cup A_2)) + (1-\gamma)\cdot(t - p(A_1))\right\}\\ &= \frac{w(I)}{p(I)}\left\{t - p(A_1) - \gamma p(A_2)\right\} \geq 0,\end{aligned}$$

where the first inequality follows since I is density maximal, and the last one because $\chi^{A_1} + \gamma \cdot \chi^{A_2}$ is feasible for [FK(t)]. ◀

▶ **Lemma 5.** *For any $t \in [0, T]$, solution $(x_j^*(t))_{j \in J}$ is an optimal solution to [FK(t)].*

Proof. By Lemma 3, there exists an optimal solution z to [FK(t)] such that $z = \chi^{A_1} + \gamma\chi^{A_2}$ where sets A_1 and A_2 satisfy properties 2, 3, and 4 of Lemma 3. Without loss of generality, we assume that $\sum_j z_j p_j = p(A_1) + \gamma p(A_2) = t$, since otherwise we can add to A_2 a dummy job (which does not participate in any precedence constraint) of zero weight and processing time $\frac{1}{\gamma}(t - p(A_1)) - p(A_2)$. We split the proof in two cases.

Case 1: $p(J_1) \geq t$. In this case $x^*(t) = \frac{t}{p(J_1)}\chi^{J_1}$ and thus it is optimal by Lemma 4.

Case 2: $p(J_1) < t$. We modify z to obtain a solution that takes set J_1 integrally. For some number $\gamma' \in [0, 1]$, we will subtract from z the following vector in order to leave space for J_1

$$y = \chi^{A_1 \cap J_1} + \gamma\chi^{A_2 \cap J_1} + \gamma'\chi^{A_1 \setminus J_1} + \gamma'\gamma\chi^{A_2 \setminus J_1}.$$

Notice that if we choose $\gamma' = 0$ then $\sum_{j \in J} p_j y_j = p(A_1 \cap J_1) + \gamma p(A_2 \cap J_1) \leq p(J_1)$, and setting $\gamma' = 1$ we obtain that $\sum_{j \in J} p_j y_j = p(A_1) + \gamma p(A_2) = t > p(J_1)$. Therefore, by continuity we can choose $\gamma' \in [0, 1]$ so that $\sum_{j \in J} p_j y_j = p(J_1)$, that is,

$$\gamma' = \frac{p(J_1 \setminus A_1) - \gamma p(A_2 \cap J_1)}{p(A_1 \setminus J_1) + \gamma p(A_2 \setminus J_1)} \in [0, 1].$$

We first notice that the total weight of y is larger than the weight of χ^{J_1}. To this end, we first show that y is a feasible solution to [FK($p(J_1)$)]. This suffices to conclude that $w(J_1) \geq \sum_{j \in J} w_j y_j$, since χ^{J_1} is optimal to [FK($p(J_1)$)] by Lemma 4. To show that y is feasible to [FK($p(J_1)$)], notice that Inequality (8) is satisfied by our choice of γ'. Let us check (9) for some $(j, k) \in P$. Notice that if $k \notin A_1 \cup A_2$ then $z_k = 0$ and thus the corresponding inequality in (9) is satisfied immediately. Similarly, if $j \notin A_1 \cup A_2$ then, since $A_1 \cup A_2$ is an ideal, $k \notin A_1 \cup A_2$ and thus (9) holds. Thus it suffices to consider $j, k \in A_1 \cup A_2$. Since J_1, A_1, and A_2 are ideals, the only precedence constraints that we need to check are: $(j, k) \in (A_1 \cap J_1) \times (A_2 \cup (A_1 \setminus J_1))$, $(j, k) \in (A_1 \setminus J_1) \times (A_2 \setminus J_1)$, $(j, k) \in (A_2 \cup J_1) \times (A_2 \setminus J_1)$.

For each of these cases, it holds that $y_j \geq y_k$ since $1 \geq \gamma \geq \gamma\gamma'$ and $1 \geq \gamma' \geq \gamma\gamma'$. We conclude that y is feasible for [FK($p(J_1)$)] and thus $w(J_1) \geq \sum_{j\in J} w_j y_j$.

We are now ready to construct a new solution for [FK(t)],

$$z' = z - y + \chi^{J_1} = (1-\gamma')\left(\chi^{A_1\setminus J_1} + \gamma\chi^{A_2\setminus J_1}\right) + \chi^{J_1},$$

has a weight that is less or equal to z. Moreover, we can check that this solution is also feasible for [FK(t)]. Indeed, by construction it holds that $\sum_j p_j z'_j = \sum_j p_j z_j \leq t$. Similarly as before, the only precedence constraints $(j,k) \in P$ that we need to check are when $(j,k) \in J_1 \times ((A_1 \setminus J_1) \cup (A_2 \setminus J_1))$, and $(j,k) \in (A_1 \setminus J_1) \times (A_2 \setminus J_1)$. All of these constraints hold since $1 \geq \gamma \geq \gamma\gamma'$.

We conclude that there exists an optimal solution z' to [FK(t)] that assigns integrally J_1, that is, $z'_j = 1$ for all $j \in J_1$. As well we have that $x^*_j(t) = 1$ for all $j \in J_1$. We can then remove J_1 from our instance and consider a residual problem with precedence graph $G_{J\setminus J_1}$ for a remaining fractional knapsack problem with capacity $t - p(J_1)$. The lemma follows by recursing on this argument. ◀

▶ **Lemma 6.** *The solution $(x^*_j)_{j\in J}$ is an optimal solution for [Rel].*

Proof. Integrating by parts, the objective function of [Rel] can be rewritten as[2]

$$\sum_{j\in J} w_j \cdot \int_0^T f(t)x'_j(t)dt = w(J)\cdot f(T) - \int_0^T \sum_{j\in J} w_j x_j(t)df(t),$$

and hence we can obtain a problem equivalent to [Rel] if we change the objective to maximize $\int_0^T \sum_{j\in J} w_j x_j(t)df(t)$. Since any solution $(x_j(\cdot))_{j\in J}$ for [Rel] defines a feasible solution $(x_j(t))_{j\in J}$ to [FK(t)], and $(x^*_j(t))_{j\in J}$ optimizes [FK(t)], we obtain that $\sum_{j\in J} w_j x_j(t) \leq \sum_{j\in J} w_j x^*_j(t)$. Because f is non-decreasing, then

$$\int_0^T \sum_{j\in J} w_j x_j(t)df \leq \int_0^T \sum_{j\in J} w_j x^*_j(t)df,$$

which helps us to conclude that $(x^*_j(\cdot))_{j\in J}$ is an optimal solution to [Rel]. ◀

We are finally ready to prove Theorem 1.

Proof of Theorem 1. Consider a feasible schedule $\mathcal{S}$ that follows the Sidney decomposition $J_1,\dots,J_k$. Since for any $j \in J_i$ we have that $C_j \leq C(J_i)$, the cost of that solution can be bounded from above by $\sum_{i=1}^k w(J_i)f(C(J_i))$. On the other hand, the optimal fractional value, attained at solution x^*, can be rewritten as

$$\sum_{j=1}^n w_j \int_0^T f(t)\frac{dx^*_j(t)}{dt}dt = \sum_{i=1}^k w(J_i)\int_{S(J_i)}^{C(J_i)} \frac{f(t)}{p(J_i)}dt,$$

[2] Here the integral taken is the Riemann-Stieltjes integral. This is well defined since f is of bounded variation (since it is non-decreasing) and x_j is continuous. Integration by parts is then valid [10, Theorem 12.14]. Notice that if f were differentiable we could simply write $\int_0^T \sum_{j\in J} w_j x_j(t)f'(t)dt$.

and thus the approximation ratio is at most

$$\frac{\sum_{i=1}^{k} w(J_i) f(C(J_i))}{\sum_{i=1}^{k} w(J_i) \int_{S(J_i)}^{C(J_i)} \frac{1}{p(J_i)} f(t)dt} \leq \max_i \frac{f(C(J_i))}{\int_{S(J_i)}^{C(J_i)} \frac{1}{p(J_i)} f(t)dt} \leq \sup_{0\leq S<C} \frac{f(C)}{\frac{1}{C-S}\int_S^C f(t)dt}$$

$$\leq \sup_{0\leq C} \frac{C \cdot f(C)}{\int_0^C f(t)dt},$$

where the last inequality follows since $\frac{1}{C-S}\int_S^C f(t)dt \geq \frac{1}{C}\int_0^C f(t)dt$, because f is non-decreasing. This shows the theorem. ◀

▶ **Corollary 7.** *There exists a universal solution that, for any concave function f, achieves an approximation guarantee of* 2*. This solution can be computed in polynomial time.*

Proof. It is enough to notice that if f is concave and non-negative, then $f(t) \geq tf(C)/C$ for any $t \in [0, C]$. Hence, $\int_0^C f(t)dt \geq Cf(C)/2$ and thus $\Gamma_f \leq 2$. ◀

It is worth noticing that the result of Theorem 1 is tight, in the sense that the integrality gap of [Rel] is exactly Γ_f. Indeed, note that for any $C \geq 0$ we can take an instance with one job of processing time C and weight 1. Then the optimal LP solution has a cost of $(1/C)\int_0^C f(t)dt$, whereas the optimal schedule has a cost of $f(C)$.

3 A Rounding Procedure for Job-Dependent Cost Functions

The more general case with objective function $\sum_j f_j(C_j)$ for f_j non-decreasing can also be tackled based on [Rel]. For this we simple generalize the objective function to $\sum_j \int_0^T f_j x'_j$. We call the new relaxation [G-Rel]. In this case we are not able to give an analytic optimal solution for the relaxation. Instead, we discretize the time in the relaxation in order to compute $(1+\epsilon)$-approximate solutions. Afterwards, we round this solution to obtain a non-preemptive schedule. Notice that this does not yield a universal solution. The approximation guarantee is not the same as in Theorem 1, however it also yields a guarantee of $2+\epsilon$ for concave functions.

We first show the rounding procedure, which is based on the concept of α-points. Consider a feasible solution $(x_j)_j$ for [G-Rel]. For a given number $\alpha \in [0, 1]$, we define the α-point of job j as $C_j^{\text{LP}}(\alpha) := \min\{t \geq 0 : x_j(t) \geq \alpha\}$, that is, the first point in time in which an α fraction of j is processed. We schedule the jobs in the order of α-points, for some (random) value of α. For simplicity, relabel the jobs so that $C_1^{\text{LP}}(\alpha) \leq \ldots \leq C_n^{\text{LP}}(\alpha)$, and thus the completion time of job j in the algorithm is $C_j^{\text{ALG}} = \sum_{k\leq j} p_j$. The next lemma relates the α-point of a job to its actual completion time. The one thereafter relates the function $C_j^{\text{LP}}(\cdot)$ with the objective function of [G-Rel]. The exposition here takes cues from that in [20].

▶ **Lemma 8** (Goemans [9])**.** *For any $\alpha \in [0, 1]$ and $j \in J$ it holds that*

$$C_j^{\text{ALG}} \leq \frac{1}{\alpha} C_j^{\text{LP}}(\alpha).$$

Proof. Notice that for all $k \leq j$ it holds that $x_k\left(C_j^{\text{LP}}(\alpha)\right) \geq x_k\left(C_k^{\text{LP}}(\alpha)\right) = \alpha$ (since x_k is non-decreasing), and thus

$$C_j^{\text{ALG}} = \sum_{k\leq j} p_k \leq \frac{1}{\alpha}\sum_{k\leq j} p_k x_k(C_j^{\text{LP}}(\alpha)) \leq \frac{1}{\alpha}\sum_{k\in J} p_k x_k(C_j^{\text{LP}}(\alpha)) \leq \frac{1}{\alpha} C_j^{\text{LP}}(\alpha),$$

where the last inequality follows from (2). ◀

▶ **Lemma 9.** *For any j it holds that*

$$\int_0^1 f_j(C_j^{\mathrm{LP}}(\alpha))d\alpha = \int_0^T f_j(t)x'_j(t)dt.$$

Proof. Consider a fixed job j and let $0 = s_1 < s_2 < \ldots < s_\ell = T$ be the breakpoints of the piecewise linear function x_j. Let us also denote by δ_k the derivative of x'_j in (s_k, s_{k+1}), and denote by $\alpha_k = x_j(s_k)$ the fraction of job j processed up to time s_k. Notice that $0 = \alpha_1 \le \alpha_2 \le \ldots \le \alpha_\ell = 1$. Then

$$\int_0^1 f_j(C_j^{\mathrm{LP}}(\alpha))d\alpha = \sum_{k=1}^{\ell-1} \int_{\alpha_k}^{\alpha_{k+1}} f_j(C_j^{\mathrm{LP}}(\alpha))d\alpha$$

Notice that if $\alpha_k < \alpha_{k+1}$, within the interval $\alpha \in (\alpha_k, \alpha_{k+1})$ the function $C_j^{\mathrm{LP}}(\alpha)$ is linear and has a slope of $1/\delta_k$ (observe that $\alpha_k < \alpha_{k+1}$ iff $\delta_k \neq 0$). Hence, using the change of variable $t = C_j^{\mathrm{LP}}(\alpha)$ we obtain that

$$\begin{aligned}\sum_{k=1}^{\ell-1} \int_{\alpha_k}^{\alpha_{k+1}} f_j(C_j^{\mathrm{LP}}(\alpha))d\alpha &= \sum_{k:\alpha_k<\alpha_{k+1}} \int_{\alpha_k}^{\alpha_{k+1}} f_j(C_j^{\mathrm{LP}}(\alpha))d\alpha = \sum_{k:\alpha_k<\alpha_{k+1}} \int_{s_k}^{s_{k+1}} f_j(t)\delta_k dt \\ &= \sum_{k=1}^{\ell-1} \int_{s_k}^{s_{k+1}} f_j(t)\delta_k dt = \int_0^T f_j(t)x'_j(t)dt.\end{aligned}$$ ◀

In our algorithm we take α randomly in $[0,1]$ with density 2α.

▶ **Lemma 10.** *Let $\Gamma'_f = \sup_{0\le t\le\tau} \frac{tf(\tau)}{\tau f(t)}$. Taking $\alpha \in [0,1]$ randomly with density 2α yields a solution such that*

$$\mathbb{E}\left(\sum_{j\in J} f_j(C_j^{\mathrm{ALG}})\right) \le 2(\max_{k\in J}\Gamma'_{f_k}) \cdot \sum_{j\in J}\int_0^T f_j x'_j(t)dt.$$

Proof. Due to the Lemma 8,

$$\mathbb{E}(f_j(C_j^{\mathrm{ALG}})) \le \int_0^1 f_j\left(\frac{1}{\alpha}C_j^{\mathrm{LP}}(\alpha)\right)2\alpha d\alpha.$$

Since for any $0 \le t \le \tau$ it holds that $t \cdot f_j(\tau) \le \Gamma'_{f_j} \cdot \tau \cdot f_j(t)$, we can take $\tau = \frac{1}{\alpha}C_j^{\mathrm{LP}}(\alpha)$ and $t = C_j^{\mathrm{LP}}(\alpha) \le \tau$ and thus $f_j(\frac{1}{\alpha}C_j^{\mathrm{LP}}(\alpha)) \le \Gamma'_{f_j} \cdot \frac{1}{\alpha} \cdot f_j(C_j^{\mathrm{LP}}(\alpha))$ we obtain that

$$\mathbb{E}(f_j(C_j^{\mathrm{ALG}})) \le 2\Gamma'_{f_j}\int_0^1 \frac{1}{\alpha} f_j(C_j^{\mathrm{LP}}(\alpha))\alpha d\alpha = 2(\Gamma'_{f_j}) \cdot \int_0^T f_j(t)x'_j(t)dt.$$

The lemma then follows by summing over j and using linearity of expectations. ◀

Notice that if f_j is concave then $\Gamma'_{f_j} \le 1$, and hence taking x to be optimal for [G-Rel] we would obtain a 2-approximation algorithm.

Moreover generally, if we can compute in polynomial time a solution x that is a $(1+\epsilon)$-approximate solution to [G-Rel], then this lemma shows that the problem admits a $(2(\max_{k\in J}\Gamma'_{f_k})(1+\epsilon))$-approximation algorithm. In what follows we show how to compute such solution x. The proof relies in the classic technique of discretizing the time axis in polynomially many intervals. Similar techniques were used by Bansal and Pruhs [2] and Cheung and Shmoys [5].

In what follows we must assume that we are given an oracle that allows us to query the values $f_j(t)$ for any $t \in \mathbb{N}$ (recall that we assume the processing times to be integral, and thus we only need to query f_j at integral points). Recall also that $T = \sum_j p_j$ is our time horizon. We assume that $f_{\max} := \max_j f_j(T)$ and $f_{\min} := \min\{f_j(t) : j \in J, t \in \{1, \ldots, T\}, f_j(t) > 0\}$ are part of the input, and thus we can manipulate these values in polynomial time. Notice that in any feasible (non-preemptive) schedule the functions f_j get evaluated only at integral points. Hence, without loss of generality, we assume that $f_j(t) = f_j(\lceil t \rceil)$ for all $t \in [0, T]$.

Let us fix a job j now. We partition the time horizon $\{0, 1, 2, \ldots, T\}$ in consecutive sets. The first set is defined as $I_j^0 = \{t \in \mathbb{N} : f_j(t) = 0\}$, and for any integer $\ell \geq 1$ we define

$$I_j^\ell = \{t \in \mathbb{N} : f_{\min} \cdot (1+\epsilon)^{\ell-1} < f_j(t) \leq f_{\min} \cdot (1+\epsilon)^\ell\}.$$

Notice that the intervals I_j^ℓ for $\ell \in \{0, 1, \ldots, \nu\}$ completely cover $\{0, 1, \ldots, T\}$ if $\nu = \lceil \log_{1+\epsilon}(f_{\max}/f_{\min}) \rceil$, and thus a polynomial number of interval suffices. Let t_j^ℓ be the largest number in I_j^ℓ and consider the set $\mathcal{T} = \{0, T\} \cup \{t_j^\ell : \text{for all } j \in J, \ell \in \{0, \ldots, \nu\}, t_j^\ell \leq T\}$. Let us relabel the elements in $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_h\}$ such that $0 = \tau_1 < \tau_2 < \ldots < \tau_h$. We remark that $|\mathcal{T}| = h \leq (\nu + 1)n + 2$.

With this we define a rounded version of the cost function $\tilde{f}_j$. For any $t \in [0, T]$ the value $\tilde{f}_j(t)$ is defined as $f(\tau_{\ell(j,t)})$, where $\ell(j, t)$ is defined such that $\tau_{\ell(j,t)-1} < t \leq \tau_{\ell(j,t)}$. We obtain that $f_j(t) \leq \tilde{f}_j(t) = f_j(\tau_{\ell(j,t)}) \leq (1+\epsilon) f_j(\lceil t \rceil) = (1+\epsilon) f_j(t)$ for all $t \in [0, T]$. Hence, obtaining an optimal solution [G-Rel] with cost functions $\tilde{f}_j$ yields a $(1+\epsilon)$-approximate solutions for the original cost functions.

▶ **Lemma 11.** *We can compute in polynomial time an optimal solution to [G-Rel] with cost functions $\tilde{f}_j$.*

Proof. Consider any solution x for [G-Rel] with cost functions $\tilde{f}_j$. Consider a new solution $\tilde{x}$ obtained by interpolating the values at $\mathcal{T}$, that is,

$$\begin{aligned} &\tilde{x}_j(0) = x_j(0) && \text{for all } j \in J, \\ &\tilde{x}_j(t) = \frac{\tau_\ell - t}{\tau_\ell - \tau_{\ell-1}} x_j(\tau_{\ell-1}) + \frac{t - \tau_{\ell-1}}{\tau_\ell - \tau_{\ell-1}} x_j(\tau_\ell) && \text{for all } \ell, t \in (\tau_{\ell-1}, \tau_\ell], j \in J. \end{aligned} \tag{11}$$

Since the functions $\tilde{f}_j$ are constant within an interval $(\tau_{\ell-1}, \tau_\ell]$, it is easy to see that the new solution achieves the same objective function, and it is also feasible. Hence, we can restrict [G-Rel] to have solutions as in (11) for all $j \in J$ and $t \in [0, T]$. We can regard this LP as having variables $x_j(\tau_\ell)$ for all $j \in J, \tau_\ell \in \mathcal{T}$. This yields a problem with a polynomial number of variables:

$$|J| \cdot |\mathcal{T}| = nh \leq n((\nu+1)n + 2) = O(n^2 \log_{1+\epsilon}(f_{\max}/f_{\min})) = O((n^2/\epsilon) \log(f_{\max}/f_{\min})).$$

Hence, it suffices to argue that we can impose a polynomial number of inequalities that imply the restrictions of the LP. Notice that any solution given by (11) is piecewise linear. The fact that x_j is non-decreasing is equivalent to $x_j(\tau_{\ell-1}) \leq x_j(\tau_\ell)$ for all j, ℓ. Since the value $x_j(t)$ for $t \in (\tau_\ell, \tau_{\ell+1}]$ is a convex combination of $x_j(\tau_\ell)$ and $x_j(\tau_{\ell+1})$, restriction (3) is implied by $x_j(\tau_\ell) \geq x_k(\tau_\ell)$ for all $(j, k) \in P$ and $\ell \in \{1, \ldots, h\}$. For the same reason (2) is implied by

$$\sum_{j \in J} x_j(\tau_\ell) p_j \leq \tau_\ell \qquad \text{for all } \ell \in \{1, \ldots, h\}.$$

Combining all these inequalities yields an equivalent problem of polynomial size. ◀

Collecting our results, we obtain the following theorem.

▶ **Theorem 12.** *For any $\epsilon > 0$, the problem $1|prec|\sum_j f_j(C_j)$ admits a polynomial-time approximation algorithm with approximation factor $(1+\epsilon)\cdot 2\cdot \max_{j\in J} \sup_{t,\tau} \left\{ \frac{t f_j(\tau)}{\tau f_j(t)} \,:\, 0 \le t \le \tau \right\}$. This implies the existence of a $(2+\epsilon)$-approximation algorithm if f_j is concave for all $j \in J$.*

References

1 N. Bansal and S. Khot. Optimal long code test with one free bit. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2009)*, pages 453–462. IEEE, 2009.

2 N. Bansal and K. Pruhs. The geometry of scheduling. In *Proceedings of the 51th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2010)*, pages 407–414. IEEE, 2010.

3 N. Bansal and K. Pruhs. The geometry of scheduling. *SIAM Journal on Computing*, 43:1684–1698, 2014.

4 C. Chekuri and R. Motwani. Precedence constrained scheduling to minimize sum of weighted completion times on a single machine. *Discrete Applied Mathematics*, 98:29–38, 1999.

5 M. Cheung and D. Shmoys. A primal-dual approximation algorithm for min-sum single-machine scheduling problems. In *Proceedings of the 14th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX 2011)*, volume 6845 of *Lecture Notes in Computer Science*, pages 135–146. Springer, 2011.

6 F. A. Chudak and D. S. Hochbaum. A half-integral linear programming relaxation for scheduling precedence-constrained jobs on a single machine. *Operations Research Letters*, 25:199–204, 1999.

7 J. R. Correa and A. S. Schulz. Single-machine scheduling with precedence constraints. *Mathematics of Operations Research*, 30:1005–1021, 2005.

8 L. Epstein, A. Levin, A. Marchetti-Spaccamela, N. Megow, J. Mestre, M. Skutella, and L. Stougie. Universal sequencing on a single machine. *SIAM Journal on Computing*, 41:565–586, 2012.

9 M. X. Goemans. Improved approximation algorithms for scheduling with release dates. In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1997)*, pages 591–598, 1997.

10 R. A. Gordon. *The Integrals of Lebesgue, Denjoy, Perron, and Henstock.* American Mathematical Society, 1994.

11 R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

12 L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Mathematics of Operations Research*, 22:513–544, 1997.

13 W. Höhn and T. Jacobs. On the performance of Smith's rule in single-machine scheduling with nonlinear cost. *ACM Transactions on Algorithms*, 11, 2015.

14 S. Im, B. Moseley, and K. Pruhs. Online scheduling with general cost function. In *Proceedings of the 23rd Annual Symposium on Discrete Algorithms (SODA 2012)*, pages 1254–1265, 2012.

15 S. G. Kolliopoulos and G. Steiner. Partially ordered knapsack and applications to scheduling. *Discrete Applied Mathematics*, 155:889–897, 2007.

16 F. Margot, M. Queyranne, and Y. Wang. Decompositions, network flows, and a precedence constrained single-machine scheduling problem. *Operations Research*, 51:981–992, 2003.

17 N. Megow and J. Verschae. Dual techniques for scheduling on a machine with varying speed. In *Automata, Languages, and Programming (ICALP 2013)*, volume 7965 of *Lecture Notes in Computer Science*, pages 745–756, 2013.

18 J. Mestre and J. Verschae. A 4-approximation for scheduling on a single machine with general cost function. *arXiv:1403.0298*, 2014.

19 A. S. Schulz. Scheduling to minimize total weighted completion time: Performance guarantees of LP-based heuristics and lower bounds. In *Integer Programming and Combinatorial Optimization (proceedings of IPCO V)*, volume 1084 of *Lecture Notes in Computer Science*, pages 301–315, 1996.

20 A. S. Schulz and M. Skutella. Random-based scheduling: New approximations and LP lower bounds. In *Randomization and Approximation Techniques in Computer Science*, volume 1269 of *Lecture Notes in Computer Science*, pages 119–133, 1997.

21 J. B. Sidney. Decomposition algorithms for single machine scheduling with precedence relations and deferral costs. *Operations Research*, 23:283–298, 1975.

22 M. Skutella. List scheduling in order of α-points on a single machine. In E. Bampis, K. Jansen, and C. Kenyon, editors, *Efficient Approximation and Online Algorithms: Recent Progress on Classical Combinatorial Optimization Problems and New Applications*, volume 3484 of *Lecture Notes in Computer Science*, pages 250–291. Springer, 2006.

23 S. Stiller and A. Wiese. Increasing speed scheduling and flow scheduling. In *Proceedings of the 21st Symposium on Algorithms and Computation (ISAAC 2010)*, volume 6507 of *Lecture Notes in Computer Science*, pages 279–290, 2010.

The Power of Migration for Online Slack Scheduling

Chris Schwiegelshohn[*1] **and Uwe Schwiegelshohn**[2]

1 **Computer Science Department, TU Dortmund, Dortmund, Germany**
chris.schwiegelshohn@tu-dortmund.de

2 **Department of Electrical Engineering and Information Technology, TU Dortmund, Dortmund, Germany**
uwe.schwiegelshohn@tu-dortmund.de

Abstract

We investigate the power of migration in online scheduling for parallel identical machines. Our objective is to maximize the total processing time of accepted jobs. Once we decide to accept a job, we have to complete it before its deadline d that satisfies $d \geq (1+\varepsilon)\cdot p + r$, where p is the processing time, r the submission time and the slack $\varepsilon > 0$ a system parameter. Typically, the hard case arises for small slack $\varepsilon \ll 1$, i.e. for near-tight deadlines. Without migration, a greedy acceptance policy is known to be an optimal deterministic online algorithm with a competitive factor of $\frac{1+\varepsilon}{\varepsilon}$ (DasGupta and Palis, APPROX 2000). Our first contribution is to show that migrations do not improve the competitive ratio of the greedy acceptance policy, i.e. the competitive ratio remains $\frac{1+\varepsilon}{\varepsilon}$ for any number of machines.

Our main contribution is a deterministic online algorithm with almost tight competitive ratio on any number of machines. For a single machine, the competitive factor matches the optimal bound of $\frac{1+\varepsilon}{\varepsilon}$ of the greedy acceptance policy. The competitive ratio improves with an increasing number of machines. It approaches $(1+\varepsilon)\cdot\ln\frac{1+\varepsilon}{\varepsilon}$ as the number of machines converges to infinity. This is an exponential improvement over the greedy acceptance policy for small ε. Moreover, we show a matching lower bound on the competitive ratio for deterministic algorithms on any number of machines.

1998 ACM Subject Classification F.1.2 Online Computation

Keywords and phrases Online scheduling, deadlines, preemption with migration, competitive analysis

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.75

1 Introduction

We address a basic scheduling problem on parallel identical machines. Given a sequence of jobs, we are required to use the available resources as best as possible such that no accepted job exceeds its deadline. The deadline of a job is at least $(1+\varepsilon)\cdot p_j$ time units after its submission time, where p_j is the processing time of the job and $\varepsilon > 0$ is the (typically small) slack parameter. We model resource utilization by maximizing the total processing time. In addition to the slack, we require the system to support preemption and migration.

From a system's perspective, migration is an important topic. Not only is it technically feasible, at least to some degree, in some cases it is even required [2, 8, 26]. For instance, when

[*] The first author wants to acknowledge the German Research Council (DFG) for its support of this research within the Collaborative Research Center SFB 876, project A2.

outsourcing jobs to an external provider, these jobs are often executed via virtual machines instead of given access to a real server. Since the underlying systems are highly adaptive environments dealing with machine failure and maintenance, migration is a necessary part of such virtual infrastructure management [25].

From a theoretical perspective, assuming migration is a convenient simplification when designing algorithms. Additionally, understanding when and how much migration improves performance can ease the decision whether implementing migration in a system is worthwhile. For a few examples where migration improves the performance of online algorithms, we refer to [1, 6, 9, 22, 24].

1.1 Our Contribution

In this paper, we consider a system of m parallel identical machines supporting preemption with migration and process jobs in an online fashion. A job can be interrupted at any given time and resumed at a later date on a possibly different machine without overhead. At any given time, a job can only be executed on one machine and each machine can process at most one job. The slack $\varepsilon > 0$ is known to the algorithm a-priori and upon submission of a job J_j at time r_j, the processing time p_j and deadline $d_j \geq (1+\varepsilon)\cdot p_j + r_j$ are revealed to the algorithm. We must immediately decide whether or not to accept J_j, denoted by the indicator variable $U_j = 1$ if rejected and $U_j = 0$ if accepted and receive a payoff proportionate to p_j, if J_j is accepted. Once a job is accepted, we must complete it on time, i.e. we cannot postpone completion of a job beyond its deadline in favor of another job. Using the three-field notation, the problem is $P_m|\varepsilon, \text{online}, \text{pmtn}|\sum p_j \cdot (1-U_j)$. We measure the performance of online algorithms via the competitive ratio. Since we have a maximization problem, the competitive ratio is the maximum value of $\frac{\text{OPT}(J)}{\text{Alg}(J)}$ over all input sequences J, where $\text{OPT}(J)$ is the objective value of an optimal offline algorithm and $\text{Alg}(J)$ is the objective value achieved by our online algorithm.

To accurately present our results, we require the definition of the function $g(x) = \varepsilon\cdot\left(\frac{1+\varepsilon}{\varepsilon}\right)^x$. Our main contribution is the design of a deterministic algorithm with an almost tight competitive factor.

▶ **Theorem 1.** *The $P_m|\varepsilon$, online, pmtn$|\sum p_j \cdot (1-U_j)$ problem admits a deterministic online algorithm with competitive ratio at most* $\max\{\frac{m\cdot(1+\varepsilon)}{\sum_{i=0}^{m-1} g\left(\frac{i}{m}\right)}, \frac{4}{3}\}$.

We are especially interested in the behavior of the function $\frac{m\cdot(1+\varepsilon)}{\sum_{i=0}^{m-1} g\left(\frac{i}{m}\right)}$ for small ε and large m, though we note that for any value of $0 < \varepsilon \leq 1$ and $m > 1$, the competitive ratio of our algorithm is an improvement over the greedy algorithm. For a single machine, there is no migration and the expression reduces to $\frac{1+\varepsilon}{\varepsilon}$. As m tends to infinity, the expression approaches $(1+\varepsilon)\ln\frac{1+\varepsilon}{\varepsilon}$. For small ε, this is an exponential improvement over greedy. We also note that the competitive ratio improves quickly, i.e. even for a comparatively small number of machines a parallel system drastically outperforms a single machine environment. Generally, for m machines, the competitive factor of our algorithm is $O(\sqrt[m]{1/\varepsilon})$. For an arguably more immediate feel for the competitive ratios of various choices of ε and m, we refer to Figure 1.

We further prove the following lower bound.

▶ **Theorem 2.** *Any deterministic online algorithm for the $P_m|\varepsilon$, online, pmtn$|\sum p_j \cdot (1-U_j)$ problem has a competitive ratio of at least* $\frac{\lfloor m\cdot(1+\varepsilon)\rfloor}{\sum_{i=0}^{m-1} g\left(\frac{i}{m}\right)}\cdot(1-\delta)$ *for any $\delta > 0$.*

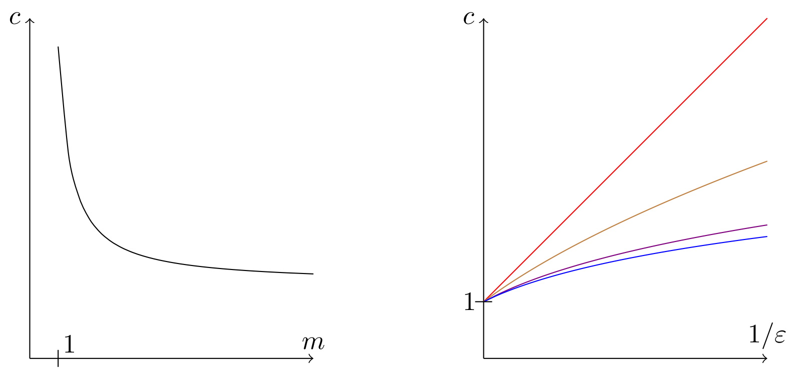


Figure 1 Performance of the online algorithm for increasing number of machines. The plots contain parameterizations of the function $\frac{m\cdot(1+\varepsilon)}{\sum_{i=0}^{m-1}\varepsilon\cdot(\frac{1+\varepsilon}{\varepsilon})^{i/m}}$, which is the competitive ratio of our algorithm for $0<\varepsilon\le 1$. We have $\varepsilon = 0.1$ in the left figure and the x-axis and the y-axis denote the number of machines and the competitive ratio, respectively. In the right figure, the x-axis and y-axis denote the dependency on $1/\varepsilon$ (i.e. $1/\varepsilon$ is large when the slack ε is small) and the competitive ratio, respectively. The red line is the competitive ratio of the greedy algorithm for increasingly smaller slack; the brown, violet and blue lines are the competitive ratios of our algorithm for 2, 10 and an infinite number of machines, respectively. Note that the competitive ratio is not equivalent to the plotted function for large ε (i.e. $1/\varepsilon$ close to 0).

When $m\cdot(1+\varepsilon)$ is integral and $c_A \ge \frac{4}{3}$, our algorithm achieves a tight competitive ratio.

The greedy acceptance policy accepts any job that can be completed on time provided no other accepted job is delayed beyond its deadline. For the same problem without migrations, this algorithm is known to guarantee an optimal competitive ratio of $\frac{1+\varepsilon}{\varepsilon}$ [7]. We show that the greedy acceptance policy cannot benefit from migrations:

▶ **Theorem 3.** *The greedy acceptance policy for the $P_m|\varepsilon, online, pmtn|\sum p_j\cdot(1-U_j)$ problem has a competitive ratio of at least $\frac{1+\varepsilon}{\varepsilon}-\delta$ for any $\delta>0$.*

Since the non-migratory and the migratory models are identical for a single machine, the greedy algorithm is optimal in this case. But its performance is exponentially worse than the performance of our algorithm as $\varepsilon\to 0$ and $m\to\infty$.

1.2 Related Work

The offline problems $1|r_j,\text{pmtn}|\sum w_j\cdot(1-U_j)$ and $P_2|r_j,\text{pmtn}|\sum U_j$ are NP hard [5]. Lawler [18] proposed a dynamic programming algorithm over the range of weights for the former problem, i.e. $1|r_j,\text{pmtn}|\sum(1-U_j)$ is solvable in polynomial time. Kalyanasundaram and Pruhs [13] showed that any schedule for the $P_m|r_j,pmtn|\sum w_j\cdot(1-U_j)$ problem can be transformed into a non-migratory schedule with $O(1)$ additional machines.

Early preemptive online algorithms assumed that every job can be accepted, but possibly delayed arbitrarily. A payoff w_j is only received if the job is completed prior to deadline. In this case there exists a tight 4-competitive deterministic algorithm on a single machine for a class of well-behaved payoff functions including $w_j = p_j$ [3, 17, 27]. For arbitrary payoff functions and without further restrictions on the input, the competitive factor is unbounded [27]. Kalyanasundaram and Pruhs [14] gave a $O(1)$ competitive randomized algorithm for maximizing the number of completed jobs and also showed that no constant competitive deterministic algorithm exists.

When further requiring that any accepted job must be completed before its deadline, there exists no constant competitive algorithm in general. To obtain more meaningful results, research incorporated the slack $\varepsilon > 0$[1]. For $1|\varepsilon, \text{online}, \text{pmtn}|\sum p_j \cdot (1 - U_j)$, Baruah and Haritsa [4] showed that a greedy acceptance policy achieves an optimal $\frac{1+\varepsilon}{\varepsilon}$ competitive ratio. This result was extended to parallel machines supporting preemption but not migration by DasGupta and Palis [7]

A lot of work has also been done for the non-preemptive variants of the problem. In this case, the difficulty of obtaining a meaningful competitive ratio was first addressed by bounding the aspect ratio $\Delta = \frac{\max p}{\min p}$. Lipton and Tomkins [20] gave a lower bound of $\Omega(\log \Delta)$ for the competitive factor of any randomized online algorithm on a single machine and additionally assuming $d_j = p_j + r_j$ for all jobs, obtained an upper bound of $O(\log^{1+\delta} \Delta)$ for any $\delta > 0$. For arbitrary deadlines, Goldman, Parwatikar, and Suri [10] were able to obtain a $6(\lceil \log \Delta \rceil + 1)$ competitive algorithm. They further studied the problem with slack ε and achieved a competitive factor of $1 + \frac{\lceil \varepsilon \rceil}{\varepsilon}$ if there are only two different job processing times. Goldwasser [11] distinguished between exactly two distinct processing times or arbitrary processing times and gave tight deterministic competitive factors of $1 + \max\left\{\frac{\lceil \varepsilon \rceil + 1}{\lceil \varepsilon \rceil}, \frac{\lfloor \varepsilon \rfloor + 1}{\varepsilon}\right\}$ and $2 + \frac{1}{\varepsilon}$, respectively. Kim and Chwa [16] later proved the same competitive factor for a greedy acceptance policy on multiple machines. Lee [19] studied small slack factors $\varepsilon \ll 1$ by way of parallel systems. For $P_m|\text{online}|\sum p_j(1 - U_j)$, he gave a deterministic $m + 1 + m \cdot \varepsilon^{1/m}$ competitive algorithm. Hence, if $m = \log 1/\varepsilon$, the algorithm is $O(\log 1/\varepsilon)$ competitive. By simulating the parallel algorithm and picking one machine uniformly at random, Lee further obtained an expected competitive factor of $1 + 3 \log 1/\varepsilon$.

1.3 Outline

In Section 3, we describe properties of schedules if all jobs are completed prior to their deadlines. In a first order, it allows us to check whether a new job can be added to the schedule without determining the whole schedule. Furthermore, using a series of transformations and employing the slack factor property for deadlines, we are able to derive a canonical schedule without improving the competitive factor. The canonical schedule contains jobs with exponentially increasing deadlines. This exponential sequence represents a tradeoff between already accepted total processing time and the potential to accept more processing time in the future. Having determined the existence of such a schedule, the competitive ratio of our acceptance procedure follows almost immediately if all jobs have the same release date, see Section 4. When jobs arrive over time, the acceptance condition is insufficient due to the existence of jobs with large processing times. However, by introducing a careful charging scheme, we can leverage processing time of accepted jobs against processing time of rejected jobs without impacting the competitive factor, provided ε is small enough. Due to space restrictions, we omit a detailed calculus. We conclude by proving our lower bounds in Section 5.

2 Notations

Formally, we have a system of m parallel identical machines to schedule the jobs of a job sequence $\mathcal{J}$. The system allows preemption with migration, that is, it can interrupt the execution of any job and immediately or later resume the execution on a possibly different

[1] In literature the stretch $f = 1 + \varepsilon$ is often used instead. The two notions are equivalent.

machine without any preemption penalty (increase in processing time or forced additional idle time on the machine). A job $J_j \in \mathcal{J}$ has processing time p_j, release date r_j and deadline $d_j \geq (1+\varepsilon) \cdot p_j + r_j$ with $\varepsilon > 0$ being the slack factor of the system. We say that a schedule for a job system is *legal* if it completes all accepted jobs before or at their respective deadlines and each machine executes at most one job at any moment. The system receives the jobs one by one in sequence order and must irrevocably and without knowledge of any later jobs decide whether it accepts the actual job or not. However, it can only accept a job if there is a legal schedule for this job and all previously accepted jobs. We use the binary variable U_j to express the decision for job J_j: $U_j = 0$ denotes acceptance of job J_j while the rejection of J_j produces $U_j = 1$. It is our goal to maximize the total processing time of all accepted jobs $(\sum p_j \cdot (1 - U_j))$. Therefore, an (online) algorithm A determines $U_j(A)$ for all jobs in $\mathcal{J}$ and produces utilization $\sum p_j \cdot (1 - U_j(A))$.

We evaluate our algorithm by determining bounds for the competitive factor, that is the largest ratio between the optimal objective value and the objective value produced by online algorithm A for all possible job sequences $\mathcal{J}$: A has a competitive factor c_A if $c_A \geq \frac{p^*(\mathcal{J})}{\sum_{J_j \in \mathcal{J}}(1-U_j(A))\cdot p_j}$ holds for any job sequence $\mathcal{J}$ with $p^*(\mathcal{J}) = \max\{\sum_{J_j \in \mathcal{J}}(1-U_j)\cdot p_j\}$ being the maximum total processing time of any legal schedule for $\mathcal{J}$. Finally, c^* is a lower bound of the competitive factor if $c^* \leq c_A$ holds for any online algorithm A.

Next we introduce function $g(x) = \varepsilon \cdot \left(\frac{1+\varepsilon}{\varepsilon}\right)^x$. Straightforward calculus yields the identity:

$$\sum_{j=i}^{m+i-1} g\left(\frac{j}{m}\right) + \left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{i}{m}} = \sum_{j=i+1}^{m+i} g\left(\frac{j}{m}\right) \tag{1}$$

We further use g to define the following threshold expression:

$$f(m,\varepsilon) = \frac{1}{\varepsilon} \cdot \sum_{i=1}^{m} g\left(\frac{i}{m}\right) = \frac{1}{1+\varepsilon} \cdot \frac{\left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{1}{m}}}{\left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{1}{m}} - 1}. \tag{2}$$

3 Legal Schedules

$\max\left\{\max_{J_j}\{p_j\}, \frac{1}{m}\sum_{J_j} p_j\right\}$ is the optimal makespan for the problem $P_m|\text{pmtn}|C_{\max}$, see, for instance, Pinedo [21]. Therefore, there is a legal schedule on m parallel identical machines for a set of jobs $\mathcal{J}$ with a common deadline d if and only if $p_j \leq d$ for all $J_j \in \mathcal{J}$ and $\sum_{J_j \in \mathcal{J}} p_j \leq d \cdot m$ hold. We use this result and a function $V_{\min}(t)$ to derive a similar necessary and sufficient condition for the extended problem with different deadlines. For a set of jobs $\mathcal{J}$ with possibly different deadlines, $V_{\min}(t)$ is the minimum total processing time that we must execute in interval $[0, t)$ of a legal schedule for time $t > 0$:

$$V_{\min}(t) = \sum_{J_j \in \mathcal{J}} \begin{cases} 0 & \text{for} \quad d_j - p_j \geq t \\ p_j & \text{for} \quad t \geq d_j \\ p_j - d_j + t & \text{else} \end{cases} \tag{3}$$

We sort all ν different deadlines in increasing order with d_ν being the largest deadline and set $d_0 = 0$.

▶ **Lemma 4.** *There is a legal preemptive schedule for a set of jobs $\mathcal{J}$ with deadlines on m parallel identical machines if and only if*

$$p_j \leq d_j \text{ for all } J_j \in \mathcal{J} \text{ and} \tag{4}$$

$$V_{min}(t) \leq t \cdot m \text{ for all } t > 0. \tag{5}$$

Proof. Due to Equation (3), $V_{\min}(t)$ is monotonically increasing and continuous. $\frac{d}{dt}V_{\min}(t)$ is piecewise constant and can only decrease if t is a deadline. Therefore, Inequalities (5) are valid if they hold for every deadline.

only if: Clearly if we have either $V_{\min}(d_i) > m \cdot d_i$ for at least one deadline d_i or least one job $J_j \in \mathcal{J}$ with $p_j > d_j$ then there is no legal schedule for $\mathcal{J}$.

if: We assume that Inequalities (4) and (5) hold. Then we generate a schedule for each interval $[d_{i-1}, d_i)$ in a backward order starting with $[d_{\nu-1}, d_\nu)$.

To this end, we generate an LRPT (Least Remaining Processing Time first) schedule for all jobs with deadline d_i. Since LRPT generates an optimal schedule for $P|\text{pmtn}|C_{\max}$, it produces a legal schedule within interval $[0, d_i)$ for these jobs due to the validity of Inequalities (4) for all jobs with deadline d_i and Inequality (5) for deadline d_i. We denote $\Delta = d_i - d_{i-1}$ and use interval $[0, \Delta)$ of the LRPT schedule as interval $[d_{i-1}, d_i)$ of our schedule. Then we reduce the processing time of each job by its total amount of processing within this interval and assign the new deadline d_{i-1} to all jobs with deadline d_i.

Now we must show that Inequalities (4) and (5) hold for the new largest deadline d_{i-1}. Since the LRPT schedule is legal, the total processing of a job in interval $[\Delta, d_i)$ of the LRPT schedule cannot exceed d_{i-1}. Therefore, Inequalities (4) hold for all these jobs after the modifications while Inequalities (4) remain valid for all unmodified jobs.

If there is some idleness in interval $[0, \Delta)$ of the LRPT schedule then we process $\min\{\Delta, p_j\}$ of each job J_j with deadline d_i within this interval, that is, the total processing in this interval is identical to $V_{\min}(d_i) - V_{\min}(d_{i-1})$. Therefore, $V_{\min}(d_{i-1})$ remains unchanged.

If no machine is idle in interval $[0, \Delta)$ of the LRPT schedule then $V_{\min}(d_{i-1})$ may increase and we have

$$V_{\min}(d_{i-1}) \quad = \quad V_{\min}(d_i) - m \cdot \Delta \leq m \cdot d_i - m \cdot \Delta = m \cdot d_i - m \cdot (d_i - d_{i-1}) = m \cdot d_{i-1}.$$

In both cases Inequality (5) is valid for deadline d_{i-1}. ◀

Since Lemma 4 does not consider the slack we introduce another function that leads to a sufficient but not necessary condition. First we say that a job J_j is large if $p_j > \frac{d_j}{1+\varepsilon}$ holds. Although there is no submission of large jobs, the progression of time may turn jobs into large jobs. Therefore, we must consider large jobs as well and define for every time $t > 0$:

$$V_{\text{accept}}(t) \quad = \quad \sum_{J_j \in \mathcal{J}} \begin{cases} p_j & \text{for } d_j \leq t \\ \max\{p_j - \frac{d_j}{1+\varepsilon}, 0\} & \text{for } d_j > t > d_j - \frac{d_j}{1+\varepsilon} \\ p_j - d_j + t & \text{for } d_j - \frac{d_j}{1+\varepsilon} \geq t > d_j - p_j \\ 0 & \text{for } d_j - p_j \geq t \end{cases} \tag{6}$$

Informally, we use $V_{\text{accept}}(t)$ to define a reduced threshold for accepting a new job J_j, that is, we reject a new job if for at least one time $t \geq d_j$, the new jobs leads to $V_{\text{accept}}(t) > t \cdot f(m, \varepsilon)$ - see Equation (2) - even if the acceptance of the job may produce a legal schedule.

Before formally describing this algorithm we must show that the validity of

$$V_{\text{accept}}(t) \quad \leq \quad t \cdot f(m, \varepsilon) \text{ for all } t > 0 \tag{7}$$

always guarantees a legal schedule. Therefore, the next lemma is the key lemma for our algorithm. Its proof is based on a reduction of the instance space with the help of several transformations until we obtain an instance space that we can analyze with the help of a few equations. Schwiegelshohn [23] used a similar approach to generate an alternative proof for

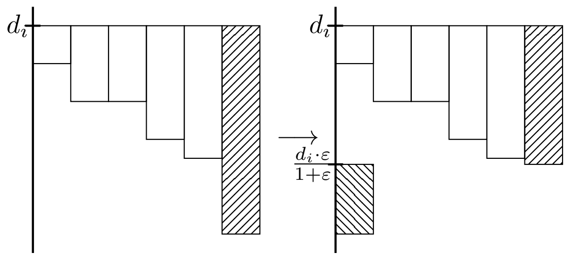


Figure 2 Large Job Splitting. The large job is marked as hatched area.

approximation factor of the lrf (largest ratio first) algorithm for the $P||\sum w_j C_j$ problem, see Kawaguchi and Kyan [15].

▶ **Lemma 5.** *There is a legal preemptive schedule for a set of jobs $\mathcal{J}$ with possibly different deadlines on m parallel identical machines if Inequalities (4) and (7) hold.*

Proof. As with function $V_{\text{min}}(t)$ we need not examine all t to check the validity of Inequalities (7). Due to Equation (6), $V_{\text{accept}}(t)$ has the following properties:

- It is monotonically increasing.
- It is continuous unless t is a deadline.
- $\frac{d}{dt}V_{\text{accept}}(t)$ is piecewise constant.
- $\frac{d}{dt}V_{\text{accept}}(t)$ can only decrease if $t = \frac{d_j}{1+\varepsilon}$ for a large job J_j.

Therefore, Inequalities (7) are valid if they hold for every deadline d_i and the corresponding time instance $\frac{d_i}{1+\varepsilon}$ if there is a large job with deadline d_i.

We use contradiction to prove this lemma and assume that Inequalities (4) and (7) hold while there is no legal schedule for $\mathcal{J}$, that is, there is at least one deadline d_h with $V_{\text{min}}(d_h) > d_h \cdot m$, see Lemma 4. Let d_j be the largest deadline with $V_{\text{min}}(d_j) > d_j \cdot m$. We apply several transformations that do not decrease $V_{\text{min}}(d_j)$ while the validity of Inequalities (4) and (7) is maintained. We use the notation $d_{>i}$ and $d_{i>}$ to describe the next larger and the next smaller deadline of deadline d_i, respectively.

First, we introduce and discuss our transformations.

Large job splitting. We split a large job J_i into one job J_{i_1} with deadline $d_{i_1} = d_i$ and processing time $p_{i_1} = \frac{d_i}{1+\varepsilon}$, and one job J_{i_2} with deadline $d_{i_2} = d_i \cdot \frac{\varepsilon}{1+\varepsilon}$ and processing time $p_{i_2} = p_i - \frac{d_i}{1+\varepsilon} \le d_i - \frac{d_i}{f} = d_{i_2}$, see Fig. 2.
This transformation neither changes $V_{\text{min}}(t)$ for any t nor $V_{\text{accept}}(t)$ for any $t \ge d_{i_2}$. Further, it may decrease but cannot increase $V_{\text{accept}}(t)$ for any $t < d_{i_2}$. Therefore, we assume that our job sequence does not contain any large job.

Job removal. We remove any job J_i with $d_i - p_i \ge d_j$. This transformation does not change $V_{\text{min}}(t)$ for $t \le d_j$ and cannot increase $V_{\text{accept}}(t)$ for any $t > 0$. Since other transformations may generate jobs with $d_i - p_i \ge d_j$ we must apply this transformation repeatedly.

Spread generation. Assume two jobs J_{i_1} and J_{i_2} with

$$d_{i_1} \quad = \quad d_{i_2} > d_j > d_{i_1} - p_{i_2} \ge d_{i_1} - p_{i_1} \ge d_{i_1} - \frac{d_{i_1}}{f}.$$

For job J_{i_2} we reduce its processing time p_{i_2} and its deadline d_{i_2} by $\delta < d_{i_1} - d_{i_1>}$, see Fig. 3. Due to $d_j - d_{i_1} + p_{i_2} = d_j - (d_{i_1} - \delta) + (p_{i_2} - \delta)$ this transformation does not change $V_{\text{min}}(d_j)$ while it does not increase $V_{\text{accept}}(t)$ for $t \ne d_{i_1} - \delta$. Clearly, the

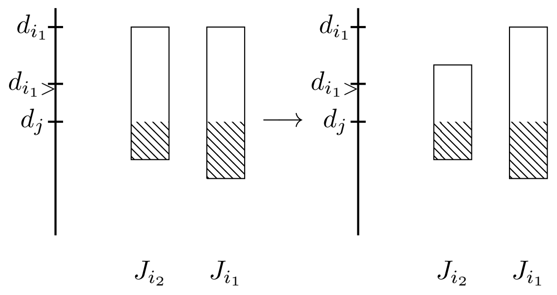


Figure 3 Spread Generation. Hatched areas contribute to $V_{\min}(d_j)$.

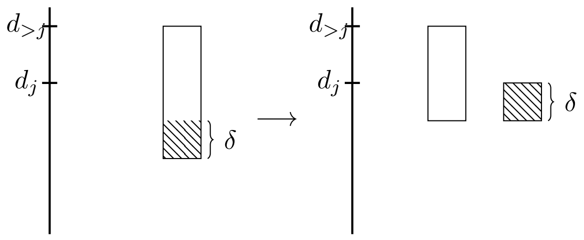


Figure 4 V_0-transformation. The hatched area becomes a new job and continues to contribute to $V_{\min}(d_j)$.

modified job J_{i_2} is not large, and we have

$$\begin{aligned} V_{\text{accept}}(d_{i_1}-\delta) &= V_{\text{accept}}(d_{i_1}) - p_{i_1} - \delta \\ &\leq d_{i_1}\cdot f(m,\varepsilon) - p_{i_1} - \delta \leq (d_{i_1}-\delta)\cdot f(m,\varepsilon) \end{aligned}$$

for $\delta \leq \frac{p_{i_1}}{f(m,\varepsilon)-1}$. Therefore, we can use any δ with $0<\delta<\min\{d_{i_1}-d_{i_1>}, \frac{p_{i_1}}{f(m,\varepsilon)-1}\}$.

V$_0$-transformation. Assume $V_{\text{accept}}(d_j) = d_j\cdot f(m,\varepsilon) - \delta$ with $\delta > 0$ and a job J_i with $d_i = d_{>j}$ and $d_j > d_i - p_i$. We introduce a new job with deadline d_j and processing time $p' = \min\{\delta, d_j - d_i + p_i\}$. Note that the new job is not large since job J_i is not large. Then we reduce processing time p_i by p', see Fig. 4. This transformation does not change $V_{\min}(d_j)$ and $V_{\text{accept}}(t)$ for $t<d_j$ and $t\geq d_{>j}$. For $d_{>j} > t \geq d_j$ we increase $V_{\text{accept}}(t)$ to $d_j\cdot f(m,\varepsilon) - \delta + p' \leq d_j\cdot f(m,\varepsilon)$.

V-transformation. Assume a job J_i with $V_{\text{accept}}(d_i) = d_i\cdot f(m,\varepsilon) - \delta_V < d_i\cdot f(m,\varepsilon)$ and $V_{\text{accept}}(d_{i>}) = d_{i>}\cdot f(m,\varepsilon)$ for $d_{i>}\geq d_j$: We reduce processing time p_i of job J_i by $p' = \frac{\delta_V}{(1+\varepsilon)\cdot f(m,\varepsilon)-1}$ to p_i' and its deadline d_i by $(1+\varepsilon)\cdot p'$ to d_i', respectively, see Fig. 5. This transformation produces $p_i' = p_i - p' \leq \frac{d_i}{1+\varepsilon} - \frac{d_i - d_i'}{1+\varepsilon} = \frac{d_i'}{1+\varepsilon}$ and

$$\begin{aligned} V_{\text{accept}}(t) &= V_{\text{accept}}(d_i) - p' = d_i\cdot f(m,\varepsilon) - \delta_V - \frac{\delta_V}{(1+\varepsilon)\cdot f(m,\varepsilon)-1} \\ &= d_i\cdot f(m,\varepsilon) - \frac{(1+\varepsilon)\cdot\delta_V\cdot f(m,\varepsilon)}{(1+\varepsilon)\cdot f(m,\varepsilon)-1} = (d_i - (1+\varepsilon)\cdot p')\cdot f(m,\varepsilon) \end{aligned}$$

for $d_i > t \geq d_i'$. It increases $V_{\min}(d_j)$ by $\varepsilon\cdot p'$ and decreases $V_{\text{accept}}(t)$ by p' for $t\geq d_i$. $V_{\text{accept}}(t)$ remains unchanged for $t<d_i'$. We have $d_i' > d_{i>}$ since $V_{\text{accept}}(d_i) - p_i \geq V_{\text{accept}}(d_{i>})$ holds. We apply this transformation in the order of increasing deadlines.

p-transformation. Assume two jobs J_{i_1} and J_{i_2} with $d_{i_1} > d_{i_2} > d_j > d_{i_1} - p_{i_1}$, $p_{i_1} < \frac{d_{i_1}}{1+\varepsilon}$, and $p_{i_2} < \frac{d_{i_2}}{1+\varepsilon}$. We reduce p_{i_2} by $p' = \min\{d_j - d_{i_2} + p_{i_2}, \frac{d_{i_1}}{1+\varepsilon} - p_{i_1}\}$ and increase p_{i_1} by p', see Fig. 6. This transformation does change $V_{\min}(d_j)$ and does not increase $V_{\text{accept}}(t)$ for any t.

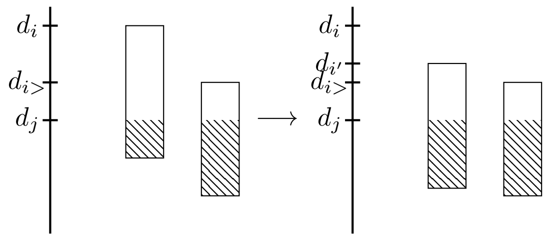


Figure 5 V-transformation. Hatched areas contribute to $V_{\text{min}}(d_j)$.

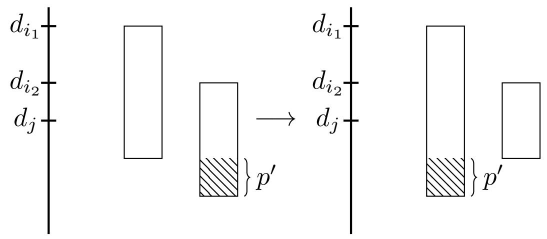


Figure 6 p-transformation. The hatched area moves to another job and continues to contribute to $V_{\text{min}}(d_j)$.

After repeatedly applying these transformations we obtain a job sequence with the following properties:

1. $V_{\text{accept}}(d_i) = d_i \cdot f(m, \varepsilon)$ for $d_i \geq d_j$
2. $p_i < \frac{d_i}{1+\varepsilon}$ for at most one deadline $d_i > d_j$
3. $p_i = \frac{d_i}{1+\varepsilon}$ for every other deadline $d_i > d_j$
4. There are no jobs with $d_i - p_i \geq d_j$.
5. There are no two jobs with the same deadline $d_i > d_j$.

We determine the value of $V_{\text{accept}}(d_i)$ for a job J_i with $d_i > d_j$:

$$V_{\text{accept}}(d_i) \quad = \quad d_i \cdot f(m, \varepsilon) = d_i \cdot \frac{\varepsilon}{1+\varepsilon} \cdot \sum_{z=1}^{m} \left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{z}{m}} = d_i + d_i \cdot \frac{\varepsilon}{1+\varepsilon} \cdot \sum_{z=1}^{m-1} \left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{z}{m}}$$

For $p_i = \frac{d_i}{1+\varepsilon}$ we have

$$V_{\text{accept}}(d_i) \quad = \quad V_{\text{accept}}(d_{i>}) + p_i = d_{i>} \cdot \frac{\varepsilon}{1+\varepsilon} \cdot \sum_{z=1}^{m} \left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{z}{m}} + \frac{d_i}{1+\varepsilon}.$$

Combining these two equations yields $d_i = d_{i>} \cdot \left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{1}{m}}$.

First assume that $p_i = \frac{d_i}{1+\varepsilon}$ holds for every deadline $d_i > d_j$. Then we have a geometric sequence of deadlines $d_j, d_j \cdot \left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{1}{m}}, d_j \cdot \left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{2}{m}}, \ldots$. In this sequence, the contribution $v_{\text{min}}(h)$ of the job with deadline $d_j \cdot \left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{h}{m}}$ to $V_{\text{min}}(d_j)$ is:

$$v_{\text{min}}(h) \quad = \quad d_j - d_j \cdot \left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{h}{m}} \cdot \left(1 - \frac{1}{1+\varepsilon}\right) = d_j \cdot \left(1 - \frac{\varepsilon}{1+\varepsilon} \cdot \left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{h}{m}}\right)$$

Therefore, our sequence ends with $d_j \cdot \left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{m-1}{m}}$ since no job with deadline $d_j \cdot \frac{1+\varepsilon}{\varepsilon}$ or larger can influence $V_{\text{min}}(d_j)$ unless the job is large.

Alternatively we assume a sequence that contains a job J_i with deadline $d_i = d_j \cdot \left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{s-1}{m}} \cdot z$ for $1 \le s \le m$ and $1 < z < \left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{1}{m}}$ and calculate p_i using Equation (2):

$$\begin{aligned} d_i \cdot f(m,\varepsilon) &= d_j \cdot \left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{s-1}{m}} \cdot z \cdot f(m,\varepsilon) = d_j \cdot \left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{s-1}{m}} \cdot f(m,\varepsilon) + p_i \\ \Leftrightarrow p_i &= d_j \cdot \left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{s}{m}} \cdot \frac{1}{1+\varepsilon} \cdot \frac{z-1}{\left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{1}{m}} - 1}. \end{aligned}$$

Then we determine the difference between the contribution to $V_{\text{min}}(d_j)$ by the new sequence and the original geometric sequence. We calculate this difference for the deadlines at position s

$$\begin{aligned} \Delta(s) &= d_j - d_j \cdot \left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{s}{m}} \cdot \left(1 - \frac{1}{1+\varepsilon}\right) - \left(d_j - \left(d_j \cdot \left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{s-1}{m}} \cdot z - p_i\right)\right) \\ &= d_j \cdot \frac{z - \left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{1}{m}}}{\left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{1}{m}} - 1} \cdot \left(\left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{s-m}{m}} - \left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{s-1}{m}}\right) \end{aligned}$$

and for a deadline at position q with $s+1 \le q \le m$:

$$\begin{aligned} \Delta(q) &= d_j - d_j \cdot \left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{q}{m}} \cdot \left(1 - \frac{1}{1+\varepsilon}\right) - \left(d_j - d_j \cdot \left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{q-1}{m}} \cdot z \cdot \left(1 - \frac{1}{1+\varepsilon}\right)\right) \\ &= d_j \cdot \left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{q-1}{m}-1} \cdot \left(z - \left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{1}{m}}\right) \end{aligned}$$

The total difference over the sequence of deadlines is

$$\begin{aligned} \sum_{q=s+1}^{m} \Delta(q) &= d_j \cdot \left(z - \left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{1}{m}}\right) \cdot \frac{\varepsilon}{1+\varepsilon} \cdot \sum_{q=s+1}^{m} \left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{q-1}{m}} \\ &= d_j \cdot \frac{z - \left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{1}{m}}}{\left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{1}{m}} - 1} \cdot \left(1 - \left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{s}{m}-1}\right). \end{aligned}$$

$$\sum_{q=s}^{m} \Delta(q) = d_j \cdot \frac{z - \left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{1}{m}}}{\left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{1}{m}} - 1} \cdot \left(\left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{s-m}{m}} - \left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{s}{m}} + 1 - \left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{s-m}{m}}\right) \ge 0.$$

Due to the restrictions on z, we have $\sum_{q=s}^{m} \Delta_{\text{min}}(q) = 0$ if and only if $s = 0$ holds. Therefore, we obtain the largest contribution to $V_{\text{min}}(d_j)$ with a sequence of geometrically increasing deadlines. For this sequence, we calculate $V_{\text{min}}(d_j)$:

$$V_{\text{min}}(d_j) = d_j \cdot f(m,\varepsilon) + \sum_{i=1}^{m-1} \left(d_j - \frac{\varepsilon}{1+\varepsilon} \cdot \left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{i}{m}} \cdot d_j\right) = d_j \cdot m$$

This result contradicts our assumption. ◀

Algorithm 1 Admission_Control

```
1: for each newly submitted job $J_j$ do
2:     tentatively accept $J_j$
3:     for each accepted job $J_i$ do
4:         if $d_i \geq d_j$ then
5:             calculate $v = V_{\text{accept}}(d_i)$
6:             if $v > f(m,\varepsilon) \cdot d_i$ then
7:                 reject $J_j$
```

4 Upper Bound of the Online Algorithm

In this section we present our online algorithm for the $P_m|\varepsilon, \text{online}, \text{pmtn}|\sum p_j(1-U_j)$ problem. First we address the admission control case with all jobs arriving at time 0, i.e., the online property only considers the sequence of jobs but no progression of time. Our Algorithm 1 Admission_Control simply applies the threshold of Lemma 5: If no accepted job (including the newly submitted one) exceeds the threshold then the job is accepted, otherwise it is rejected. Since no job is large, a job J_j does not influence $V_{\text{accept}}(t)$ for $t < d_j$. Due to the first paragraph of the proof of Lemma 5, we only must consider time instances that are deadlines. Having accepted the jobs, a schedule can be generated by applying Lemma 4. Next we prove the competitive factor of this algorithm.

▶ **Theorem 6.** *The $P_m|\varepsilon$, online, pmtn$|\sum p_j \cdot (1-U_j)$ problem with $r_j = 0$ for all jobs admits a deterministic online admission control algorithm with competitive ratio at most $\frac{m\cdot(1+\varepsilon)}{\sum_{i=0}^{m-1} g(\frac{i}{m})}$.*

Proof. Since any algorithm produces an optimal result if it accepts all jobs we assume that some jobs are rejected. Since the deadline condition Inequality (4) is valid for every submitted job J_j, algorithm Admission_Control only rejects a job J_j if there is a $t \geq d_j$ with

$$\begin{aligned} V_{\text{accept}}(t) &\geq t \cdot f(m,\varepsilon) - p_j \geq t \cdot \left(f(m,\varepsilon) - \frac{1}{1+\varepsilon}\right) \\ &\geq \frac{t}{1+\varepsilon} \cdot \left(\varepsilon \cdot \sum_{h=1}^{m} \left(\frac{1+\varepsilon}{\varepsilon}\right)^{\frac{h}{m}} - 1\right) = \frac{t}{1+\varepsilon} \cdot \sum_{h=0}^{m-1} g(\frac{h}{m}) \end{aligned}$$

Due to Equations (3) and (6), we have $V_{\text{min}}(\tau) \geq V_{\text{accept}}(\tau)$ for all $\tau \geq 0$ while no algorithm can yield more than $V_{\text{min}}(\tau) = \tau \cdot m$. In the following we consider two intervals: $I_1 = [0, t_s)$ with t_s being the largest time with $V_{\text{min}}(t) = t \cdot \frac{1}{1+\varepsilon} \cdot \sum_{h=0}^{m-1} g(\frac{h}{m})$. Interval $I_2 = [t_s, d_\nu)$ only exists for $d_\nu > t_s$. If this case the maximum total processing in I_2 is $V' = V_{\text{min}}(d_\nu) - V_{\text{min}}(t_s)$ otherwise we say $V' = 0$. Note that $t_s > d_j$ holds. Since there is no rejected job with deadline t_s or larger it is not possible to increase V' in an optimal schedule. Then we have

$$c_A \leq \frac{t_s \cdot m + V'}{t_s \cdot \frac{1}{1+\varepsilon} \cdot \sum_{h=0}^{m-1} g(\frac{h}{m}) + V'} < \frac{t_s \cdot m}{t_s \cdot \frac{1}{1+\varepsilon} \cdot \sum_{h=0}^{m-1} g(\frac{h}{m})} = \frac{m}{\frac{1}{1+\varepsilon} \cdot \sum_{h=0}^{m-1} g(\frac{h}{m})}. \quad ◀$$

For the general case with progression of time we use Algorithm 2 Online_Utilization. Lines 9 to 12 of Algorithm Online_Utilization are identical to Algorithm Admission_Control. Our reference time t_{ref} is the latest submission time (Line 2). Since progression of time may produce large jobs we must also examine the acceptance condition for those time instances at which the large part of a job ends due to the first paragraph of the proof of Lemma 5, see

Algorithm 2 Online_Utilization

```
1: for each newly submitted job $J_j$ do
2:     $t_{ref} = r_j$;
3:     $U_j = 0$;
4:     for each job $J_i$ with $U_i = 0$ do
5:         if $p_i > (d_i - t_{ref})/(1+\varepsilon)$ and $(d_i - t_{ref})/(1+\varepsilon) + t_{ref} \geq d_j$ then
6:             calculate $v = V_{\text{accept}}((d_i - t_{ref})/(1+\varepsilon) + t_{ref})$
7:             if $v > f(m,\varepsilon) \cdot (d_i - t_{ref})/(1+\varepsilon)$ then
8:                 $U_j = 1$
9:         if $d_i \geq d_j$ then
10:            calculate $v = V_{\text{accept}}(d_i)$
11:            if $v > f(m,\varepsilon) \cdot (d_i - t_{ref})$ then
12:                $U_j = 1$
13:        if $d_i > d_j - p_j$ and $d_i < d_j$ then
14:            calculate $v = V_{\text{min}}(d_i)$
15:            if $v > m \cdot (d_i - t_{ref})$ then
16:                $U_j = 1$
```

Lines 5 to 8. We must use the current reference time when determining these time instances (Lines 5 and 6).

Progression of the reference time may lead to large jobs and it may produce a violation of the acceptance condition for some time instances. To prevent such a situation, we introduce the additional *legal test* based on Lemma 4 for all jobs that have passed the acceptance test of Lemma 5, see Lines 13 to 16 in Algorithm 2 Online_Utilization. If a job passes all tests it is accepted.

▶ **Lemma 7.** *There is a legal schedule for all jobs accepted by Algorithm 2 Online_Utilization.*

Proof. The proof directly follows from Lemma 4. ◀

In the next theorem we derive the competitive factor of our algorithm.

▶ **Theorem 1** (restated). *The $P_m|\varepsilon, online, pmtn|\sum p_j \cdot (1-U_j)$ problem admits a deterministic online algorithm with competitive ratio at most* $\max\{\frac{m\cdot(1+\varepsilon)}{\sum_{i=0}^{m-1} g(\frac{i}{m})}, \frac{4}{3}\}$.

Proof. We use induction on the number of submission times. For a single submission time, the claim holds due to Theorem 6. Therefore, we assume validity of the claim for k different submission times. As in the proof of Theorem 6 let t_s be the largest time instance with $V_{\text{min}}(t_s) = t_s \cdot \frac{1}{1+\varepsilon} \cdot \sum_{h=0}^{m-1} g(\frac{h}{m})$ with respect to any previous reference time (release date of a job). If we use the expression *with respect to a reference time* then we reduce all times by the reference time when calculating $V_{\text{min}}(t)$ and $V_{\text{accept}}(t_s)$ while we keep the original values for the purpose of time comparison. We divide the rest of our proof into four cases.

1. There is no rejection of any job with release date r_{k+1} due to the legal test in Algorithm 2 Online_Utilization and $t_s \geq r_{k+1}$. Since the progression of the reference time to r_{k+1} cannot increase t_s in a non-delay schedule without the acceptance of a new job we can assume that all job parts that are not completed before r_{k+1} have release date r_{k+1} and apply the proof of Theorem 6 to all jobs with release date r_{k+1}.

2. There is no rejection of any job with release date r_{k+1} due to the legal test in Algorithm 2 Online_Utilization and $t_s < r_{k+1}$. We assume that the optimal schedule uses all available resources before t_s. Then we define V' as in the proof of Theorem 6 and split all job parts contributing to V' into a job part that can be executed before r_{k+1} and a job part that must be executed after r_{k+1}. We denote the total processing time of the first type of job parts by $V'(t_s)$. If our scheduling algorithm executes the $V'(t_s)$ completely within interval $[t_s, r_{k+1})$ then we can apply the proof of Theorem 6 to all jobs with release date r_{k+1} and obtain the claim. Therefore, we assume that not the total processing time of $V'(t_s)$ is scheduled within interval $[t_s, r_{k+1})$. Such situation is only possible if a job J_j from $V'(t_s)$ with processing time p_j is started at time $\tau > r_{k+1} - p_j$. In a non-delay schedule such delayed start requires all machines to be busy in interval $[t_s, \tau)$. In order to maximize the total processing time of $V'(t_s)$ that is scheduled after r_{k+1} we assume that we have b long jobs each with the maximum processing time $r_{k+1} - t_s$ and each starting at time τ while all other jobs contributing to $V'(t_s)$ are only executed in interval $[t_s, \tau)$, that is, we have $V'(t_s) = (\tau - t_s) \cdot m + (r_{k+1} - t_s) \cdot b \leq m \cdot (r_{k+1} - t_s)$. We denote the total processing time of jobs with release date r_{k+1} in our schedule including $V' \backslash V'(t_s)$ by $\bar{V}$. Due to Theorem 6, the total processing time in the optimal schedule cannot exceed $c_A \cdot (\bar{V} + b \cdot (\tau - t_s))$. This situation cannot influence the competitive factor if the following condition holds:

$$\begin{aligned} c_A &\geq \frac{c_A \cdot (\bar{V} + b \cdot (\tau - t_s)) + b \cdot (r_{k+1} - t_s) + m \cdot (\tau - t_s)}{\bar{V} + b \cdot (r_{k+1} - t_s) + m \cdot (\tau - t_s)} \\ \Leftrightarrow 1 - \frac{1}{c_A} &\geq \frac{\frac{b}{m} \cdot \frac{\tau - t_s}{r_{k+1} - t_s}}{\frac{b}{m} + \frac{\tau - t_s}{r_{k+1} - t_s}} \end{aligned}$$

The term on the right side has the maximum value for $\frac{b}{m} = \frac{\tau - t_s}{r_{k+1} - t_s} = \frac{1}{2}$ resulting in $c_A \geq \frac{4}{3}$, see also Hussein and Schwiegelshohn [12]. Therefore, the competitive factor is not affected for small ε.

3. A least one job with release date r_{k+1} is rejected due to the legal test in Algorithm 2 Online_Utilization and $t_s \leq r_{k+1}$. This case leads to a contradiction since we have $V_{\text{accept}}(t) \leq V_{\min}(t) \leq t \cdot \frac{1}{1+\varepsilon} \cdot \sum_{h=0}^{m-1} g(\frac{h}{m})$ for all $t \geq r_{k+1}$ with respect to reference time r_{k+1}. Therefore, we can apply Lemma 5 for all jobs with release date r_{k+1} and any accepted job with release date r_{k+1} passes the legal test.

4. A least one job with release date r_{k+1} is rejected due to the legal test in Algorithm 2 Online_Utilization and $t_s \leq r_{k+1}$. Before discussing this case we consider the worst case example in the proof of Lemma 5 for $V_{\min}(t_s) = t_s \cdot \frac{1}{1+\varepsilon} \cdot \sum_{h=0}^{m-1} g(\frac{h}{m})$ and a single release time. Let $t_0 = t_s \cdot \frac{\varepsilon}{1+\varepsilon}$. Then we have for $t \geq t_0$

$$\begin{aligned} V_{\min}(t) &\leq m \cdot \left(\int_{t_0}^{t} \left(1 - \log_{\frac{1+\varepsilon}{\varepsilon}} \left(\frac{y}{t_0} \right) \right) dy + t_0 \right) \\ &\leq m \cdot \left(t - \frac{1}{\ln \frac{1+\varepsilon}{\varepsilon}} \cdot \left(t \cdot \ln \frac{t}{t_0} - t + t_0 \right) \right). \end{aligned} \tag{8}$$

Progression of time without accepting any new jobs can only increase the bound for $V_{\min}(t)$. Since any new accepted job must also pass the acceptance test of Lemma 5, Inequality (8) must always hold for a possibly new t_s. Therefore, any job J_h with release date r_{k+1} and a deadline $d_h \geq t_s$ cannot fail the legal test. ◀

5 Lower Bounds for Deterministic Online Algorithms

▶ **Theorem 2** (restated). *Any deterministic online algorithm for the $P_m|\varepsilon, online, pmtn|\sum p_j \cdot (1-U_j)$ problem has a competitive ratio of at least $\frac{\lfloor m\cdot(1+\varepsilon)\rfloor}{\sum_{i=0}^{m-1} g(\frac{i}{m})} \cdot (1-\delta)$ for any $\delta > 0$.*

Proof. In our proof the job sequence consists of several series of jobs all with $d_j \geq p_j \cdot (1+\varepsilon)$. In every step the adversary submits identical jobs until we have either accepted the planned number of jobs or until $\lfloor m \cdot (1+\varepsilon)\rfloor$ jobs have been submitted. We will show that the latter case produces a competitive ratio $c_A \geq \frac{\lfloor m\cdot(1+\varepsilon)\rfloor}{\sum_{i=0}^{m-1} g(\frac{i}{m})}$. Therefore, we are forced to accept the desired number of jobs.

Furthermore, we must show that there is a legal schedule for all accepted jobs. To this end, we use Lemma 4 with deadlines $d_i = \frac{1+\varepsilon}{\varepsilon} \cdot g(\frac{i-1}{m}) = g(\frac{i-1}{m}+1)$ for $1 \leq i \leq m$ and $d_{m+1} = (1+\varepsilon)\cdot(\frac{1+\varepsilon}{\varepsilon} - \delta)$ for an arbitrarily small $\delta > 0$. Since there is only a single accepted job for every deadline d_i with $1 < i \leq m$, less than m jobs contribute to the total processing time in $V_{\min}(d_{i+1}) - V_{\min}(d_i)$ for $1 \leq i < m$. Therefore, we must only consider $V_{\min}(d_1)$ to show the existence of a legal schedule.

1. The adversary submits a job with processing time $\sum_{i=0}^{m-1} g(\frac{i}{m}) - \lfloor\sum_{i=0}^{m-1} g(\frac{i}{m})\rfloor < 1$ and deadline $(1+\varepsilon)$. We must accept this job to prevent $c_A \to \infty$.
2. The adversary submits identical jobs with processing time 1 and deadline $1+\varepsilon$ until $\lfloor\sum_{i=0}^{m-1} g(\frac{i}{m})\rfloor + 1$ such jobs have been accepted. If we accept this number of jobs then our total processing time is
$$\sum_{J_j\in\mathcal{J}} (1-U(j))\cdot p_j = \sum_{i=0}^{m-1} g(\frac{i}{m}) + 1 = \sum_{i=1}^{m} g(\frac{i}{m}) = V_{\min}(1+\varepsilon),$$
see Equation (1). There is a legal schedule since we have $V_{\min}(1+\varepsilon) < m\cdot g(1) = m\cdot(1+\varepsilon)$. If we accept at most $\lfloor\sum_{i=0}^{m-1} g(\frac{i}{m})\rfloor$ of these jobs then we have a total processing time $\sum_{J_j\in\mathcal{J}}(1-U(j))\cdot p_j = \sum_{i=0}^{m-1} g(\frac{i}{m})$ and a competitive factor
$$c_A = \frac{\lfloor m\cdot(1+\varepsilon)\rfloor}{\sum_{i=0}^{m-1} g(\frac{i}{m})}.$$
3. The adversary executes $m-1$ similar submission iterations. We assume that at the beginning of iteration k we have
$$\sum_{J_j\in\mathcal{J}} (1-U(j))\cdot p_j = \sum_{i=k}^{m+k-1} g(\frac{i}{m}) \text{ and}$$
$$V_{\min}(1+\varepsilon) = \sum_{i=1}^{m} g(\frac{i}{m}) + (k-1)\cdot(1+\varepsilon) - \sum_{i=1}^{k-1} g(\frac{i}{m}),$$
respectively. Clearly, this assumption holds for $k=1$. The adversary submits jobs with processing time $p = \frac{1}{\varepsilon}g(\frac{k}{m})$ and deadline $d = \frac{1+\varepsilon}{\varepsilon}g(\frac{k}{m}) = g(\frac{k}{m}+1)$ until we accept one job. If we do not accept any of these jobs then the total processing time remains unchanged and we obtain
$$c_A = \frac{\lfloor m\cdot(1+\varepsilon)\rfloor\cdot\frac{1}{\varepsilon}g(\frac{k}{m})}{\sum_{i=k}^{m+k-1} g(\frac{i}{m})} = \frac{\lfloor m\cdot(1+\varepsilon)\rfloor}{\sum_{i=0}^{m-1} g(\frac{i}{m})}.$$
Otherwise we have
$$\sum_{J_j\in\mathcal{J}} (1-U(j))\cdot p_j = \sum_{i=k}^{m+k-1} g(\frac{i}{m}) + \frac{1}{\varepsilon}\cdot g(\frac{k}{m}) = \sum_{i=k+1}^{m+k} g(\frac{i}{m})$$
$$\Leftrightarrow V_{\min}(1+\varepsilon) = \sum_{i=1}^{m} g(\frac{i}{m}) + k\cdot(1+\varepsilon) - \sum_{i=1}^{k} g(\frac{i}{m})$$
due to $1+\varepsilon-(d-p) = 1+\varepsilon - g(\frac{k}{m})$ and Equation (1).

4. After iteration $m-1$ we have

$$\sum_{J_j \in \mathcal{J}} (1-U(j)) \cdot p_j = \sum_{i=m}^{2m-1} g(\frac{i}{m}) \text{ and}$$

$$V_{\min}(1+\varepsilon) = g(1) + (m-1)\cdot(1+\varepsilon) = m\cdot(1+\varepsilon).$$

Finally, the adversary submits $\lfloor m\cdot(1+\varepsilon)\rfloor$ jobs with processing time $p = \frac{1+\varepsilon}{\varepsilon}\cdot(1-\delta)$ and deadline $d_{m+1} = (1+\varepsilon)\cdot\frac{1+\varepsilon}{\varepsilon}\cdot(1-\delta)$ for an arbitrarily small $\delta > 0$. Since any such job must start at the latest at time

$$d_{m+1} - p = \varepsilon\cdot\left(\frac{1+\varepsilon}{\varepsilon}\cdot(1-\delta)\right) < (1+\varepsilon),$$

we cannot accept any such job and obtain the competitive factor

$$c_A = \frac{\lfloor m\cdot(1+\varepsilon)\rfloor\cdot\frac{1+\varepsilon}{\varepsilon}\cdot(1-\delta)}{\sum_{i=m}^{2m-1} g(\frac{i}{m})} = \frac{\lfloor m\cdot(1+\varepsilon)\rfloor}{\sum_{i=0}^{m-1} g(\frac{i}{m})}\cdot(1-\delta). \blacktriangleleft$$

We now turn our attention to the greedy acceptance policy. Here, we accept any job that can be computed prior to its deadline without delaying the currently accepted jobs beyond their respective deadlines.

▶ **Theorem 3** (restated). *The greedy acceptance policy for the $P_m|\varepsilon, online, pmtn|\sum p_j\cdot(1-U_j)$ problem has a competitive ratio of at least $\frac{1+\varepsilon}{\varepsilon} - \delta$ for any $\delta > 0$.*

Proof. We consider an arbitrarily small $\delta > 0$ and use at the beginning a sequence of $1+\lceil m\cdot(1+\varepsilon)\rceil$ jobs with the following processing times:

$$p_1 = \varepsilon\cdot m\cdot\delta$$

$$p_2 = \begin{cases} m\cdot(1+\varepsilon) - \lfloor m\cdot(1+\varepsilon)\rfloor - \varepsilon\cdot m\cdot\delta & \text{for } m\cdot(1+\varepsilon) \neq \lfloor m\cdot(1+\varepsilon)\rfloor \\ 1-\varepsilon\cdot m\cdot\delta & \text{for } m\cdot(1+\varepsilon) = \lfloor m\cdot(1+\varepsilon)\rfloor \end{cases}$$

$$p_3 = p_4 = \ldots = p_{1+\lceil m\cdot(1+\varepsilon)\rceil} = 1$$

All jobs $p_1, \ldots p_{1+\lceil m\cdot(1+\varepsilon)\rceil}$ have deadline $1+\varepsilon$ and must be accepted according to our policy since we have

$$V_{\min}(1+\varepsilon) = \sum_{i=1}^{1+\lceil m\cdot(1+\varepsilon)\rceil} p_i = m\cdot(1+\varepsilon). \tag{9}$$

Then the adversary submits m identical jobs with processing times $p = \frac{1+\varepsilon}{\varepsilon} - \delta$ and deadline $d = (1+\varepsilon)\cdot p = \frac{(1+\varepsilon)^2}{\varepsilon} - (1+\varepsilon)\cdot\delta$. Since the acceptance of any one of these jobs leads to

$$V_{\min}(1+\varepsilon) = m\cdot(1+\varepsilon) + 1 + \varepsilon - (d-p) = m\cdot(1+\varepsilon) + \varepsilon\cdot\delta > m\cdot(1+\varepsilon) \tag{10}$$

we must reject everyone of these jobs.

However, the rejection of job J_1 allows acceptance of all other jobs since we have $V_{\min}(1+\varepsilon) = m\cdot(1+\varepsilon)$ and $V_{\min}(\frac{(1+\varepsilon)^2}{\varepsilon} - (1+\varepsilon)\cdot\delta) = m\cdot\left(\frac{(1+\varepsilon)^2}{\varepsilon} - (1+\varepsilon)\cdot\delta\right)$ due to Equations (9) and (10).

Therefore, the greedy acceptance algorithm has a competitive factor of at least

$$c_{A_{\text{greedy}}} = \frac{m\cdot\left(\frac{(1+\varepsilon)^2}{\varepsilon} - (1+\varepsilon)\cdot\delta\right)}{m\cdot(1+\varepsilon)} = \frac{1+\varepsilon}{\varepsilon} - \delta. \blacktriangleleft$$

Since the lower bound of the competitive factor for the common preemption model is identical to the competitive factor for preemption without migration, see Lemma 5 and DasGupta and Palis [7], we can state that a greedy acceptance policy cannot exploit the benefits of migration.

References

1 S. Albers and M. Hellwig. On the value of job migration in online makespan minimization. In *Proc. of ESA*, pages 84–95, 2012.

2 J. H. Anderson, V. Bud, and U. C. Devi. An EDF-based restricted-migration scheduling algorithm for multiprocessor soft real-time systems. *Real-Time Systems*, 38(2):85–131, 2008.

3 S. K. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. E. Rosier, D. Shasha, and F. Wang. On the competitiveness of on-line real-time task scheduling. *Real-Time Systems*, 4(2):125–144, 1992.

4 S.K. Baruah and J.R. Haritsa. Scheduling for overload in real-time systems. *IEEE Trans. Computers*, 46(9):1034–1039, 1997.

5 P. Brucker and S. Knust. Complexity results for scheduling problems. `http://www2.informatik.uni-osnabrueck.de/knust/class/`, 2009. [Online; accessed 11-April-2016].

6 B. Chen, A. van Vliet, and G. J. Woeginger. An optimal algorithm for preemptive on-line scheduling. *Oper. Res. Lett.*, 18(3):127–131, 1995.

7 B. DasGupta and M.A. Palis. Online real-time preemptive scheduling of jobs with deadlines on multiple machines. *Journal of Scheduling*, 4(6):297–312, 2001.

8 R. I. Davis and A. Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.*, 43(4):35, 2011.

9 L. Epstein and A. Levin. Robust algorithms for preemptive scheduling. *Algorithmica*, 69(1):26–57, 2014.

10 S.A. Goldman, J. Parwatikar, and S. Suri. Online scheduling with hard deadlines. *Journal of Algorithms*, 34(2):370–389, 2000.

11 M.H. Goldwasser. Patience is a virtue: the effect of slack on competitiveness for admission control. In *Proc. of SODA*, pages 396–405, 1999.

12 M.E. Hussein and U. Schwiegelshohn. Utilization of nonclairvoyant online schedules. *Theor. Comput. Sci.*, 362(1-3):238–247, 2006.

13 B. Kalyanasundaram and K. Pruhs. Eliminating migration in multi-processor scheduling. *J. Algorithms*, 38(1):2–24, 2001.

14 B. Kalyanasundaram and K. Pruhs. Maximizing job completions online. *J. Algorithms*, 49(1):63–85, 2003.

15 T. Kawaguchi and S. Kyan. Worst case bound of an LRF schedule for the mean weighted flow-time problem. *SIAM Journal on Computing*, 15(4):1119–1129, 1986.

16 J.H. Kim and K.Y. Chwa. On-line deadline scheduling on multiple resources. In *Proc. of COCOON*, pages 443–452, 2001.

17 G. Koren and D. Shasha. MOCA: A multiprocessor on-line competitive algorithm for real-time system scheduling. *Theor. Comput. Sci.*, 128(1&2):75–97, 1994.

18 E. L. Lawler. A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. *Annals of Operations Research*, 26(1):125–133, 1990.

19 J. Lee. Online deadline scheduling: multiple machines and randomization. In *Proc. of SPAA*, pages 19–23, 2003.

20 R.J. Lipton and A. Tomkins. Online interval scheduling. In *Proc. of SODA*, pages 302–311, 1994.

21 M.L. Pinedo. *Scheduling: Theory, Algorithms, and Systems.* Springer Science+Business Media, forth edition, 2010.

22 P. Sanders, N. Sivadasan, and M. Skutella. Online scheduling with bounded migration. *Math. Oper. Res.*, 34(2):481–498, 2009.

23 U. Schwiegelshohn. An alternative proof of the Kawaguchi-Kyan bound for the Largest-Ratio-First rule. *Oper. Res. Lett.*, 39(4):255–259, 2011. `doi:10.1016/j.orl.2011.06.007`.

24 M. Skutella and J. Verschae. A robust PTAS for machine covering and packing. In *Proc. of ESA*, pages 36–47, 2010. `doi:10.1007/978-3-642-15775-2_4`.

25 B. Sotomayor, R. S. Montero, I. M. Llorente, and I. T. Foster. Virtual infrastructure management in private and hybrid clouds. *IEEE Internet Computing*, 13(5):14–22, 2009.

26 P. Valente and G. Lipari. An upper bound to the lateness of soft real-time tasks scheduled by EDF on multiprocessors. In *Proc. of RTSS*, pages 311–320, 2005.

27 G. J. Woeginger. On-line scheduling of jobs with fixed start and end times. *Theor. Comput. Sci.*, 130(1):5–16, 1994.

Sampling-Based Bottleneck Pathfinding with Applications to Fréchet Matching*

Kiril Solovey[1] **and Dan Halperin**[2]

1 **Blavatnik School of Computer Science, Tel Aviv University, Tel Aviv, Israel**
`kirilsol@post.tau.ac.il`

2 **Blavatnik School of Computer Science, Tel Aviv University, Tel Aviv, Israel**
`danha@post.tau.ac.il`

Abstract

We describe a general probabilistic framework to address a variety of Fréchet-distance optimization problems. Specifically, we are interested in finding minimal *bottleneck*-paths in *d*-dimensional Euclidean space between given start and goal points, namely paths that minimize the maximal value over a continuous cost map. We present an efficient and simple sampling-based framework for this problem, which is inspired by, and draws ideas from, techniques for robot motion planning. We extend the framework to handle not only standard bottleneck pathfinding, but also the more demanding case, where the path needs to be monotone in all dimensions. Finally, we provide experimental results of the framework on several types of problems.

1998 ACM Subject Classification I.3.5 Computational Geometry and Object Modeling

Keywords and phrases Computational geometry, Fréchet distances, sampling-based algorithms, random geometric graphs, bottleneck pathfinding

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.76

1 Introduction

This paper studies the problem of finding near-optimal paths in *d*-dimensional Euclidean space. Specifically, we are interested in *bottleneck* paths which minimize the maximal value the path obtains over a generally-defined continuous cost map. As an example, suppose that one wishes to plan a hiking route in a mountainous region between two camping grounds, such that the highest altitude along the path is minimized [17]. In this case, the map assigns to each two-dimensional point its altitude. A similar setting, albeit much more complex, requires to find a pathway of low energy for a given protein molecule (see, e.g., [37]).

Our main motivation for studying bottleneck optimization over cost maps is its tight relation to the Fréchet distance (or matching), which is a popular and widely studied similarity measure in computational geometry. The problem has applications to various domains such as path simplification [19], protein alignment [27], handwritten-text search [48], and signature verification [53]. The Fréchet distance, which was initially defined for curves, is often considered to be a more informative measure than the popular *Hausdorff* distance as it takes into consideration not only each curve as a whole but also the location and the ordering of points along it. Usually one is interested not only in the Fréchet distance between two given curves, but also in the parametrization which attains the optimal alignment.

* This work has been supported in part by the Israel Science Foundation (grant no. 1102/11), by the German-Israeli Foundation (grant no. 1150-82.6/2011), and by the Hermann Minkowski-Minerva Center for Geometry at Tel Aviv University. Kiril Solovey is also supported by the Clore Israel Foundation.

Since its introduction by Alt and Godau [2] in 1995, a vast number of works has been devoted to the subject, and many algorithms have been developed to tackle various settings of the problem. However, from a practical standpoint the problem is far from being solved: for many natural extensions of the Fréchet problem only prohibitively-costly algorithms are known. Furthermore, in some cases it was shown, via hardness proofs, that efforts for finding polynomial-time algorithms are doomed to fail. For some variants of the problem efficient algorithms are known to exist, however their implementation requires complex geometric machinery that relies on geometric kernels with infinite precision [30].

Contribution. We describe a generic, efficient and simple algorithmic framework for solving pathfinding optimization problems over cost maps. The framework is inspired by, and draws ideas from, sampling-based methods for robot motion planning. We provide experimental results of the framework on various scenarios. Furthermore, we theoretically analyze the framework and show that the cost of the obtained solution converges to the optimum, as the number of samples increases. We also consider the more demanding case, where paths need to be monotone in all dimensions.

Organization. In Section 2 we review related work. In Section 3 we provide a formal definition of the bottleneck pathfinding problem. In Section 4 we describe an algorithmic framework for solving this problem. In Section 5 we provide an analysis of the method. Finally, in Section 6 we report on experimental results.

2 Related work

This section is devoted to related work on Fréchet distance and robot motion planning.

2.1 Fréchet distance

The *Fréchet distance* between two curves is often described by an analogy to a person walking her dog: each of the two creatures is required to walk along a predefined path and the person wishes to know the length of the shortest *leash* which will make this walk possible. In many cases one also likes to know how to advance along the path given the short leash.

Formally, let $\sigma_1, \sigma_2 : [0,1] \rightarrow \mathbb{R}^d$ be two continuous curves. We wish to find a traversal along the two curves which minimizes the distance between the two traversal points. The traversal is defined by two continuous parametrizations $\alpha_1, \alpha_2 : [0,1] \rightarrow [0,1]$ of σ_1, σ_2 respectively, where for a given point in time $\tau \in [0,1]$, the positions of the person and her dog are specified by $\sigma_1(\alpha_1(\tau))$ and $\sigma_2(\alpha_2(\tau))$, respectively. The *Fréchet distance* between σ_1, σ_2 is defined by the expression

$$\min_{\alpha_1,\alpha_2:[0,1]\rightarrow[0,1]} \max_{\tau\in[0,1]} \|\sigma_1(\alpha_1(\tau)) - \sigma_2(\alpha_2(\tau))\|_2.$$

Alt and Godau [2] described an $O(n^2 \log n)$-time algorithm for the setting of two polygonal curves, where n is the number of vertices in each of the two curve. Buchin et al. [12] described a different method for solving this problem for the same running time. Recently, Buchin et al. [10] developed an algorithm with a slightly improved running time $O(n^2 \log^2 \log n)$. Har-Peled and Raichel [22] introduced a simpler randomized algorithm with running time of $O(n^2 \log n)$. Bringmann [6] showed that an algorithm with running time of $O(n^{2-\delta})$, for some constant $\delta > 0$, does not exist, unless a widely accepted conjecture, termed

SETH [25], is wrong. In a following work [7] this conditional lower bound was extended to $(1+\varepsilon)$-approximation algorithms of the Fréchet problem, where $\varepsilon \leq 0.399$.

The notion of Fréchet distance can be extended to k curves in various ways. One natural extension can be described figuratively as having a pack of k dogs, where each of the dogs has to walk along a predefined path, and every pair of dogs is connected with a leash. The goal now is to find a parametrization which minimizes the length of the longest leash. Dumitrescu and Rote [18] introduced a generalization of the Alt-Godau algorithm to this case, which runs in $O(kn^k \log n)$ time, i.e., exponential in the number of input curves. They also describe a 2-approximation algorithm with a much lower running time of $O(k^2n^2 \log n)$. In the work of Har-Peled and Raichel [22] mentioned above they also consider the case of k input curves and devise an $O(n^k)$ algorithm. Notably, their technique is flexible enough to cope with different Fréchet-type goal functions over the k curves. Furthermore, their algorithm is also applicable when the k curves are replaced with k *simplicial complexes*, and the problem is to find k curves—one in each complex—which minimize the given goal function. A recent work [9], which extends the conditional lower bound mentioned earlier for the setting of multiple curves, suggests that a running time that is exponential in the number of curves is unavoidable.

The notion of Fréchet distance can be generalized to more complex objects. Buchin et al. [13] considered the problem of finding a mapping between two simple polygons, which minimizes the maximal distance between a point and its image in the other polygon. More formally, given two simple polygons $P, Q \subset \mathbb{R}^2$ the problem consists of finding a mapping $\delta : P \rightarrow Q$ which minimizes the expression $\max_{p\in P} \|p - \delta(p)\|_2$, subject to various constraints on δ. They introduced a polynomial-time algorithm for this case. In a different paper, Buchin et al. [11] showed that the decision problem is NP-hard for more complex geometric objects, e.g., pairs of polygons with holes in the plane or pairs of two-dimensional terrains. Another interesting NP-hard problem that was studied by Sherette and Wenk [41] is *curve embedding* in which one wishes to find an embedding of a curve in $\mathbb{R}^3$ to a given plane, which minimizes the Fréchet distance with the curve. In a similar setting Meulemans [34] showed that it is NP-hard to decide whether there exists a simple cycle in a plane-embedded graph that has at most a given Fréchet distance to a simple closed curve.

The Fréchet distance between curves in the presence of obstacles have earned some attention. Cook and Wenk [16] studied the *geodesic* variant, which consists of a simple polygon and two polygonal curves inside it. As in the standard formulation, the main goal is to minimize the length of the leash, but now the leash may wrap or bend around obstacles. Their algorithm has running time of $O(m + n^2 \log mn \log n)$, where m is the complexity of the polygon and n is defined as the total complexity of the two curves, as before. The more complex *homotopic* setting is a special case of the aforementioned geodesic setting, with the additional constraint that the leash must continuously deform. Chambers et al. [14] considered this problem for the specific setting of two curves in planar environment with polygonal obstacles. They developed an algorithm whose running time is $O(N^9 \log N)$, where $N = n + m$ for n and m as defined above.

2.2 Motion planning

Motion planning is a fundamental problem in robotics. In its most basic form, the problem consists of finding a collision-free path for a robot $\mathcal{R}$ in a workspace environment $\mathcal{W}$ cluttered with obstacles. Typically, the problem is approached from the configuration space $\mathcal{C}$—the set of all robot configurations. The problem can be reformulated as finding a continuous curve in $\mathcal{C}$, which entirely consists of collision-free configurations and represents a path for the robot from a given start configuration to another, target, configuration. An important attribute of the problem is the number of *degrees of freedom* of $\mathcal{R}$, using which one can specify every configuration in $\mathcal{C}$. Typically the dimension of $\mathcal{C}$ equals the number of degrees of freedom.

For some cases of the problem, which involve a small number of degrees of freedom, efficient and exact analytical techniques exist (see, e.g., [4, 21, 40]), which are guaranteed to find a solution if one exists, or report that none exists otherwise. Recently, it was shown [46, 1, 50] that efficient and complete techniques can be developed for the *multi-robot* motion-planning problem, which entails many degrees of freedom, by making several simplifying assumptions on the separation of the start and target positions. However, it is known that the general setting of the motion-planning problem is computationally intractable (see, e.g., [38, 23, 47, 43]) with respect to the number of degrees of freedom.

Sampling-based algorithms for motion planning, which were first described about two decades ago, have revolutionized the field of robotics by providing simple yet effective tools to cope with challenging problems involving many degrees of freedom. Such algorithms (see, e.g., PRM by Kavraki et al. [29], RRT by Kuffner and LaValle [32], and EST by Hsu et al. [24]) explore the high-dimensional configuration space by random sampling and connecting nearby samples, which result in a graph data structure that can be viewed as an approximation of the *free space*—a subspace of $\mathcal{C}$, which consists entirely of collision-free configurations. While such techniques have weaker theoretical guarantees than analytical methods, many of them are *probabilistically complete*, i.e., guaranteed to find a solution if one exists, given sufficient processing time. More recently, *asymptotically optimal* sampling-based algorithms, whose solution converges to the optimum, for various criteria, have started to emerge: Karaman and Frazzoli introduced the RRT* and PRM* [28] algorithms, which are asymptotically optimal variants of RRT and PRM. Following their footsteps Arslan and Tsiotras introduced RRT# [3]. A different approach was taken by Janson and Pavone who introduced the FMT* algorithm [26], which was later refined by Salzman and Halperin [39].

3 Problem statement

In this section we describe the general problem of bottleneck pathfinding over a given cost map, to which we describe an algorithmic framework in Section 4. We conclude this section we several concrete examples of the problems that will be used for experiments in Section 6.

We start with several basic definitions. Given $x, y \in \mathbb{R}^d$, for some fixed dimension $d \geq 2$, let $\|x - y\|_2$ denote the Euclidean distance between two points. Denote by $\mathcal{B}_r(x)$ the d-dimensional Euclidean ball of radius $r > 0$ centered at $x \in \mathbb{R}^d$ and $\mathcal{B}_r(\Gamma) = \bigcup_{x \in \Gamma} \mathcal{B}_r(x)$ for any $\Gamma \subseteq \mathbb{R}^d$. We will use the terms "path" and "curve" interchangeably, to refer to a continuous curve in $\mathbb{R}^d$ parametrized over $[0, 1]$. Given a curve $\sigma : [0, 1] \to \mathbb{R}^d$ define $\mathcal{B}_r(\sigma) = \bigcup_{\tau \in [0,1]} \mathcal{B}_r(\sigma(\tau))$. Additionally, denote the image of a curve σ by $\text{Im}(\sigma) = \bigcup_{\tau \in [0,1]} \{\sigma(\tau)\}$. Let $A_1, A_2, \ldots$ be random variables in some probability space and let B be an event depending on A_n. We say that B occurs *almost surely* (a.s., in short) if $\lim_{n \to \infty} \Pr[B(A_n)] = 1$.

Let $\mathcal{M} : [0, 1]^d \to \mathbb{R}$ be a *cost map* that assigns to each point in $[0, 1]^d$ a real value. For simplicity, we assume that the domain of $\mathcal{M}$ is a d-dimensional unit hypercube. Let $S, T \in [0, 1]^d$ denote the start and target points. Denote by $\Sigma(S, T)$ the collection of paths that start in S and end in T. Formally, every $\sigma \in \Sigma(S, T)$ is a continuous path $\sigma : [0, 1] \to [0, 1]^d$, where $\sigma(0) = S, \sigma(1) = T$. Given a path σ we use the notation $\mathcal{M}(\sigma) = \max_{\tau \in [0,1]} \mathcal{M}(\sigma(\tau))$ to represent its bottleneck cost.

In some applications, monotone paths are desired. For instance, in the classical problem of Fréchet matching between two curves it is often the case that backward motion along the curves is forbidden. Here we consider monotonicity in all d coordinates of points along the path. Formally, given two points $p, p' \in \mathbb{R}^d$, where $p = (p_1, \ldots, p_d), p' = (p'_1, \ldots, p'_d)$, we use the notation $p \preceq p'$ to indicate that $p_i \leq p'_i$, for every $1 \leq i \leq d$. A path $\sigma \in \Sigma(S,T)$ is said to be *monotone* if for every $0 \leq \tau \leq \tau' \leq 1$ it holds that $\sigma(\tau) \preceq \sigma(\tau')$.

▶ **Definition 1.** Given the triplet $\langle \mathcal{M}, S, T \rangle$, the *bottleneck-pathfinding problem* (BPP, for short) consists of finding a path $\sigma \in \Sigma(S,T)$ which minimizes the expression $\max_{\tau\in[0,1]} \mathcal{M}(\sigma(\tau))$. A special case of the bottleneck pathfinding problem, termed *strong*-BPP, requires that the path will be *monotone*.

3.1 Examples

We provide three examples of BPPs, which will be used for experiments in Section 6. Each example is paired with the d-dimensional configuration space $\mathcal{C} := [0,1]^d$, start and target points $S, T \in \mathcal{C}$, and a cost map $\mathcal{M} : [0,1]^d \rightarrow \mathbb{R}$. The examples below are defined for two-dimensional input objects, but can generalized to higher dimensions.

Problem 1: We start with the classical *Fréchet distance among k curves* (see, e.g., [22]). Let $\sigma_1, \ldots, \sigma_k : [0,1] \rightarrow [0,1]^2$ be k continuous curves embedded in Euclidean plane. Here $\mathcal{C} = [0,1]^k$ is defined as the Cartesian product of the various positions along the k curves. Namely, a point $P = (p_1, \ldots, p_k) \in \mathcal{C}$ describes the location $\sigma_i(p_i)$ along σ_i, for each $1 \leq i \leq k$. To every such P we assign the cost $\mathcal{M}(P) = \max_{1\leq i<j\leq k} \|\sigma_i(p_i) - \sigma_j(p_j)\|_2$. We note that more complex formulations of $\mathcal{M}$ can be used, depending on the exact application. The start and target positions are defined to be $S = (\sigma_1(0), \ldots, \sigma_k(0)), T = (\sigma_1(1), \ldots, \sigma_k(2))$.

Problem 2: We introduce the problem of *Fréchet distance with visibility*, whose basis is similar to **P1** with $k = 3$. In addition to the curves, we are given a subspace $\mathcal{F} \subseteq [0,1]^2$. The goal is to find a traversal of the curves which minimizes $\mathcal{M}$ as defined in **P1**, with the additional constraint that the traversal point along σ_1 must be "seen" by one of the traversal points of σ_2, σ_3. Formally, for every $P = (p_1, p_2, p_3) \in \mathcal{C}$ it must hold that $p_1p_2 \subset \mathcal{F}$ *or* $p_1p_3 \subset \mathcal{F}$ (but not necessarily both), where p_ip_j is the straight-line path from p_i to p_j.

Problem 3: In *curve embedding* (see, [41, 34]), the input consists of a curve $\sigma : [0,1] \rightarrow [0,1]^2$, a subspace $\mathcal{F} \subseteq [0,1]^2$ and a pair of two-dimensional points $s, t \in \mathcal{F}$. A point $P = (p_1, p_2, p_3) \in \mathcal{C} = [0,1]^3$ describes the location $\sigma(p_1)$ along σ and the point $(p_2, p_3) \in \mathcal{F}$. The BPP is defined for the start and target points $S = (0, s), T = (1, t) \in \mathcal{C}$ and the cost map $\mathcal{M}(P) = \|\sigma(p_1) - (p_2, p_3)\|_2$.

4 Algorithmic framework

In this section we describe an algorithmic framework that will be used for solving standard and strong regimes of BPP (Definition 1). The framework can be viewed as a variant of the PRM algorithm [28], and we chose to describe it here in full detail for completeness. However, the analysis provided in Section 5 is brand new.

The framework consists of three conceptually simple steps: In the first step, we construct a random graph embedded in $[0,1]^d$, whose vertices consist of the start and target points S, T, and of a collection of randomly sampled points; the edges connect points that are separated by a distance of at most a given connection threshold r_n. In the second step the edges of the graph are assigned with weights corresponding to their bottleneck cost over $\mathcal{M}$.

In the third and final step, the discrete graph is searched for a path connecting S to T which minimizes the bottleneck cost.

Before proceeding to a more elaborate description of the framework we provide a formal definition of the random graphs that are at the heart of the technique. Let $\mathcal{X}_n = \{X_1, \ldots, X_n\}$ be n points chosen independently and uniformly at random from the Euclidean d-dimensional cube $[0,1]^d$. The following definition corresponds to the standard and well-studied model of random geometric graphs (see, e.g., [36, 51, 5] and the literature review in [45]).

▶ **Definition 2.** The *random geometric graph* (RGG) $\mathcal{G}_n = \mathcal{G}(\mathcal{X}_n; r_n)$ is a *directed* graph with vertex set $\mathcal{X}_n$ and edge set $\{(x,y) : x \neq y,\ x, y \in \mathcal{X}_n,\ \|x-y\|_2 \leq r_n\}$.

We are ready to describe the framework, which has two parameters: n represents the number of samples generated and r_n defines the Euclidean connection radius used in the construction of the graphs. In the next section we show that for a range of values of r_n, which is a function of the number of samples n, the cost of the returned solution converges to the optimum, as n tends to infinity. The framework consists of the following steps:

Step I: We construct the RGG $\mathcal{G}_n = (\mathcal{X}_n \cup \{S,T\}; r_n)$. For the purpose of generating $\mathcal{G}_n$ a collection of n samples $\mathcal{X}_n$ is generated and a nearest-neighbor structure is employed to find for every $x \in \mathcal{X}_n \cup \{S,T\}$ the set of samples that located within a Euclidean distance of r_n from it.

Step II: We assign to each edge of the graph the bottleneck cost of the straight-line path connecting its endpoints under $\mathcal{M}$. In particular, for the standard BPP, for every edge (x,y) the cost $\max_{\tau\in[0,1]} \mathcal{M}(x+\tau(y-x))$ is assigned. The same applies for strong-BPP, unless $x \not\preceq y$, in which case the value $+\infty$ is assigned.

Step III: For the final step we find a path over $\mathcal{G}_n$ from S to T which minimizes the bottleneck cost. Several efficient algorithms solving this problem exist (see, e.g., [52, 15]).

5 Theoretical foundations

We study the behavior of the framework for the standard and the strong case of BPP (Definition 1). Recall the framework uses the two parameters n and r_n, which specify the number of samples and the connection radius.We establish a range of connection radii, r_n, for which the cost of the returned solution is guaranteed to converge to a relaxed notion of the optimum.

The analysis below does not restrict itself to a specific type of cost maps $\mathcal{M}$, e.g., continuous or smooth. Thus, due to the stochastic nature of the framework, and the general definition of $\mathcal{M}$, we cannot guarantee that the returned solution will tend to the absolute optimum. As an example consider the cost map $\mathcal{M}$ such that for a given $x = (x_1, x_2)$, $\mathcal{M}(x) = 0$ if $x_1 = x_2$, and $\mathcal{M}(x) = 1$ otherwise. For the start and target points $S = (0.1, 0.1), T = (0.9, 0.9)$ the optimal solution is a subset of the diagonal. Obviously, the probability of having a single point of $\mathcal{X}_n$, let alone a whole path in $\mathcal{G}_n$, that lie on the diagonal is equal to 0.

We can however guarantee convergence to a *robustly-optimal* path, which is defined below. Informally, such paths have "well-behaved" neighborhoods, in terms of the value of $\mathcal{M}$. We provide below a formal definition of this notion for the bottleneck cost function. Recall that given a path σ the notation $\mathcal{M}(\sigma)$ represents its bottleneck cost.

▶ **Definition 3.** Given the triplet $\langle \mathcal{M}, S, T\rangle$, a path $\sigma \in \Sigma(S,T)$ is called *robust* if for every $\varepsilon > 0$ there exists $\delta > 0$ such that $\mathcal{M}(\sigma') \leq (1+\varepsilon)\mathcal{M}(\sigma)$, for any $\sigma' \in \Sigma(S,T)$ such that $\text{Im}(\sigma') \subset \mathcal{B}_\delta(\sigma)$. A path that attains the infimum cost, over all robust paths, is termed *robustly optimal.*

5.1 (Standard) Bottleneck cost

For a given triplet $\langle \mathcal{M}, S, T\rangle$ representing an instance of BPP, denote by σ^* a robustly-optimal solution. Note that we do not require here that σ^* or the returned solution will be monotone. We obtain the following result. All logarithms stated henceforth are to base e.

▶ **Theorem 4.** *Let $\mathcal{G}_n = \mathcal{G}(\mathcal{X}_n \cup \{S,T\}; r_n)$ be an RGG with*

$$r_n = \gamma \left(\frac{\log n}{n}\right)^{1/d}, \quad \gamma > 2(2d\theta_d)^{-1/d},$$

where θ_d denotes the Lebesgue measure of a unit ball in $\mathbb{R}^d$. Then $\mathcal{G}_n$ contains a path $\sigma_n \in \Sigma(S,T)$ such that $\mathcal{M}(\sigma_n) = (1+o(1))\mathcal{M}(\sigma^*)$, a.s.

We mention that this connection radius is also essential for connectivity of RGGs, i.e., a smaller radius results in a graph that is disconnected with high probability (see, e.g.,[8]). This fact is instrumental to our proof. We also mention that a result similar to Theorem 4 can be obtained through a different proof technique [28], albeit with a larger value of the constant γ.

For simplicity, we assume for the purpose of the proof that exists a finite constant $\delta' > 0$ such that $\mathcal{B}_{\delta'}(\sigma^*) \subset [0,1]^d$, namely the robustly-optimal solution is at least δ' away from the boundary of the domain $[0,1]^d$. This constraint can be easily relaxed by transforming $\langle \mathcal{M}, S, T\rangle$ into an equivalent instance $\langle \mathcal{M}', S', T'\rangle$ where this condition is met. In particular the original input can be embedded to a cube of side length $1-\varepsilon$ for some constant $\varepsilon > 0$, which is centered in the middle of $[0,1]^d$. The cost along the boundaries of the smaller cube should be extended to the remaining parts of the $[0,1]^d$ cube.

Given an RGG $\mathcal{G}_n$ and a subset $\Gamma \subset [0,1]^d$ denote by $\mathcal{G}_n(\Gamma)$ the graph obtained from the intersection of $\mathcal{G}_n$ and Γ: it consists of the vertices of $\mathcal{G}_n$ that are contained in Γ and the subset of edges of $\mathcal{G}_n$ that are fully contained in Γ. Each edge is considered as a straight-line segment connecting its two end points.

A main ingredient in the proof of Theorem 4 is the following Lemma. We employ the *localization-tessellation* framework [45], which was developed by the authors. The framework allows to extend properties of RGGs to domains with complex geometry and topology.

▶ **Lemma 5.** *Let $\mathcal{G}_n$ be the RGG defined in Theorem 4. Additionally let $\Gamma \subset [0,1]^d$ be a fixed subset, where $S,T \in \Gamma$, and let $\rho > 0$ be some fixed constant, such that $\mathcal{B}_\rho(\Gamma) \subset [0,1]^d$. Then S,T are connected in $\mathcal{G}_n(\mathcal{B}_\rho(\Gamma))$ a.s.*

Proof. We rely on the well-known result that $\mathcal{G}_n$ is connected a.s. in the domain $[0,1]^d$ for the given connection radius r_n (see, e.g., [45, Theorem 1]). We then use Lemma 1 and Theorem 6 in [45] which state that if $\mathcal{G}_n$ is connected a.s., and is *localizable* (see, Definition 6 therein), then S,T are connected a.s. over $\mathcal{G}_n(\mathcal{B}_\rho(\Gamma))$. ◀

Proof of Theorem 4. We first show that for any $\varepsilon > 0$ it follows that $\mathcal{M}(\sigma_n) \leq (1+\varepsilon)\mathcal{M}(\sigma^*)$ a.s. Fix some $\varepsilon > 0$. Due to the fact that σ^* is robustly optimal, there exists $\delta_\varepsilon > 0$ independent of n such that for every $\sigma \in \Sigma(S,T)$ such that $\text{Im}(\sigma) \subset \mathcal{B}_{\delta_\varepsilon}(\sigma^*)$ we have that $\mathcal{M} \leq (1+\varepsilon)\mathcal{M}(\sigma^*)$ a.s. Additionally, recall that there exists some $\delta' > 0$ such $\mathcal{B}_{\delta'}(\sigma^*) \subset [0,1]^d$.

Set $\delta = \min\{\delta_\varepsilon, \delta'\}$ and define the sets $\Gamma_{\delta/2} = \mathcal{B}_{\delta/2}(\sigma^*), \Gamma_\delta = \mathcal{B}_\delta(\sigma^*)$ and notice that $S, T \in \Gamma_{\delta/2}$. By Lemma 5 we have that S, T are connected in $\mathcal{G}_n(\Gamma_\delta)$. Moreover, a path connecting S, T in $\mathcal{G}_n(\Gamma_\delta)$ must a have a bottleneck cost of at most $(1+\varepsilon)\mathcal{M}(\sigma^*)$.

We have shown that for any fixed $\varepsilon > 0$, $\mathcal{M}(\sigma_n) \leq (1+\varepsilon)\mathcal{M}(\sigma^*)$ a.s. By defining the sequence $\varepsilon_i = 1/i$ one can extend the previous result and show that $\mathcal{M}(\sigma_n) \leq (1+o(1))\mathcal{M}(\sigma^*)$. This part is technical and its details are omitted (see a similar proof in [44, Theorem 6]). This concludes the proof. ◀

5.2 Strong bottleneck cost

We now focus on the strong case of the problem, where the solution is restricted to paths that are monotone in each of the d coordinates. Denote by $\vec{\sigma}^*$ the robustly-optimal monotone solution for a given instance $\langle \mathcal{M}, S, T\rangle$.

► **Theorem 6.** *Let $\mathcal{G}_n = \mathcal{G}(\mathcal{X}_n \cup \{S,T\}; r_n)$ be an RGG with $r_n = \omega(1)\left(\frac{\log n}{n}\right)^{1/d}$. Then $\mathcal{G}_n$ contains a monotone path $\vec{\sigma}_n \in \Sigma(S,T)$ such that $\mathcal{M}(\vec{\sigma}_n) = (1+o(1))\mathcal{M}(\vec{\sigma}^*)$, a.s.*

Let $x, x' \in [0,1]^d$ be two points such that $x \preceq x'$. For a given $\delta > 0$ the notation $x \preceq_\delta x'$ indicates that $\delta = \min\{x'_i - x_i\}_{i=1}^d$, where $x = (x_1, \ldots, x_d), x' = (x'_1, \ldots, x'_d)$. Given two points $x, x' \in [0,1]^d$, such that $x \preceq x'$, denote by $\mathcal{H}(x,x')$ the d-dimensional box $[x_1, x'_1] \times \ldots \times [x_d, x'_d]$. In addition to the assumption that the robustly-optimal solution $\vec{\sigma}^*$ is separated from the boundary of $[0,1]^d$ that we have taken in the previous analysis, we also assume that there exists a constant $0 < \delta'' \leq 1$ such that $S \preceq_{\delta''} T$.

In preparation for the main proof we prove the following lemma.

► **Lemma 7.** *Choose any[1] $f_n \in \omega(1)$ and set $r_n = \omega(1)\left(\frac{\log n}{n}\right)^{1/d}$. Let $q, q' \in [0,1]^d$ be two points such that $q \preceq_\delta q'$, where δ is independent of n. Then a.s. there exist $X, X' \in \mathcal{X}_n$ with the following properties: (i) $\|X - q\|_2 \leq r_n/2, \|X' - q'\| \leq r_n/2$; (ii) $q \preceq X, X' \preceq q'$; (iii) X, X' are connected in $\mathcal{G}_n$ with a monotone path.*

Proof. We apply a tessellation argument similar to the one used to show that the standard (and undirected) RGG is connected (see, e.g., [51, Section 2.4]). Set $\ell = \left\lceil \frac{2\|q'-q\|_2}{r_n} \right\rceil$ and observe that $\ell \leq 2\sqrt{d}/r_n$. Define the normalized vector $\vec{v} = \frac{q'-q}{\|q'-q\|_2}$ and let $H_1, \ldots, H_\ell$ be a sequence of ℓ hyperboxes, where

$$H_j = \mathcal{H}\left(q + (j-1)\cdot\frac{r_n}{2}\cdot\vec{v}, q + j\cdot\frac{r_n}{2}\cdot\vec{v}\right),$$

for every $1 \leq j \leq \ell$ (see Figure 1). Observe that for every $1 \leq j < \ell$ and every $X_j \in H_j, X_{j+1} \in H_{j+1}$, we have

$$X_j \preceq X_{j+1}, \|X_{j+1} - X_j\|_2 \leq r_n. \tag{1}$$

We show that for every $1 \leq j \leq \ell$ it follows that $\mathcal{X}_n \cap H_j \neq \emptyset$, a.s. We start by bounding the volume of H_j. Denote by $c_1, \ldots, c_d$ the side lengths of H_j, and denote by $\delta_1, \ldots, \delta_d$ the

[1] For instance, f_n can be either one of the following functions: $\log n, \log^* n$, or the inverse Ackerman function $\alpha(n)$.

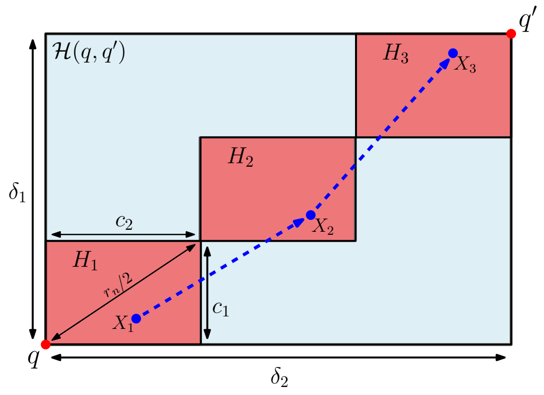


Figure 1 Visualization of the proof of Lemma 7 for $d = 2$. The blue rectangle represents $\mathcal{H}(q, q')$ and the three red rectangles represent $H_1, \ldots, H_\ell$ for $\ell = 3$ (the small value of ℓ was selected for the clarity of visualization and in reality $r_n \ll \delta_1$). The length of the largest diagonal in each of the small rectangles is $r_n/2$, which implies that a distance between $X_j \in H_j, X_{j+1} \in H_{j+1}$ is at most r_n. The blue dashed arrows represent the directed graph edges $(X_1, X_2), (X_2, X_3)$ which correspond to a monotone path connecting X_1 to X_3.

side lengths of $\mathcal{H}(q, q')$. Note that δ_i is independent of n and $c_i = \delta_i/\ell$. Consequently, we can represent $c_i = \alpha_i r_n$, where $\alpha_i > 0$ is constant, for every $1 \leq i \leq d$. Thus, $|H_j| = c r_n^d$ for some constant $c > 0$. Now,

$$\Pr[\mathcal{X}_n \cap H_j = \emptyset] = (1 - |H_j|)^n \leq \exp\{-n|H_j|\} = \exp\{-\omega(1) \cdot c \log n\} \leq n^{-1}.$$

In the last transition we used the fact that the function $f_n \in \omega(1)$ can "absorb" any constant c. We are ready to show that every H_i contains a point from $\mathcal{X}_n$ a.s.:

$$\begin{aligned}\Pr[\exists H_j : \mathcal{X}_n \cap H_j = \emptyset] &\leq \sum_{j=1}^{\ell} \Pr[\mathcal{X}_n \cap H_i = \emptyset] \\ &\leq \ell \cdot n^{-1} \leq \frac{2\sqrt{d}}{r_n} \cdot n^{-1} \\ &= \frac{2\sqrt{d}}{\omega(1) \cdot n^{1-1/d} \log^{1/d} n}.\end{aligned}$$

Thus, a.s. there exists for every $1 \leq j \leq \ell$ a point $x_j \in H_j$. Observe that $X := X_1, X' := X_\ell$ satisfy (i),(ii). Condition (iii) follows from Equation 1. ◀

Proof of Theorem 6. Similarly to the proof of Theorem 4, we fix $\varepsilon > 0$ and select $\delta \leq \min\{\delta', \delta''\}$ such that $\mathcal{M}(\vec{\sigma}) \leq (1+\varepsilon)\mathcal{M}(\vec{\sigma}^*)$ for every $\vec{\sigma} \in \vec{\Sigma}(S, T)$ with $\mathrm{Im}(\vec{\sigma}) \subset \mathcal{B}_\delta(\vec{\sigma}^*) \subset [0,1]^d$.

The crux of this proof is that there exists a sequence of k points $q_1, \ldots, q_k \in \mathrm{Im}(\vec{\sigma}^*)$, where $S = q_1, T = q_k$, such that $q_j \prec_{\delta/2} q_{j+1}$ for every $1 \leq j < k$ (see Figure 2). Moreover, due to fact that $\vec{\sigma}^*$ is monotone we can determine that such k is finite and independent of n. Thus, by Lemma 7, for every $1 \leq j < k$ there exist $X_j, X'_j \in \mathcal{X}_n$ which satisfy the following conditions a.s.: (i) $\|X_j - q_j\|_2 \leq r_n/2, \|X'_j - q_{j+1}\|_2 \leq r_n/2$; (ii) $q_j \preceq X_j, X'_j \preceq q_{j+1}$; (iii) X_j, X'_j are connected in $\mathcal{G}_n$. By conditions (i),(ii), for every $1 \leq j < k$ the graph $\mathcal{G}_n$ contains the edge (X'_j, X_{j+1}). Combined with condition (iii) this implies that S is connected to T in $\mathcal{G}_n$ a.s.

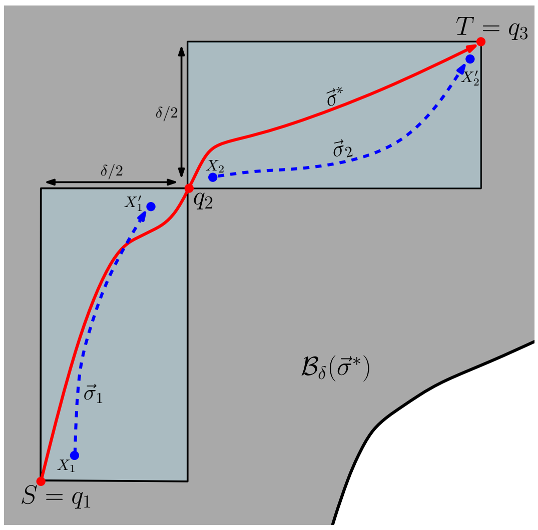


Figure 2 Visualization of the proof of Theorem 6 for $d = 2$ and $k = 3$. The red curve represents $\vec{\sigma}^*$, on which lie the points q_1, q_2, q_3 such that $q_1 \prec_{\delta/2} q_2 \prec_{\delta/2} q_3$. The dashed blue curves represent $\vec{\sigma}_1, \vec{\sigma}_2$. The gray area represents $\mathcal{B}_\delta(\vec{\sigma}^*)$.

It remains to show that the path constructed above has a cost of at most $(1+\varepsilon)\mathcal{M}(\vec{\sigma}^*)$. For every $1 \leq j \leq k$ denote by $\vec{\sigma}_j$ the path induced by Lemma 7 from X_j to X_j', i.e., $\vec{\sigma}_j(0) = X_j, \vec{\sigma}_j(1) = X_j'$ and $\text{Im}(\vec{\sigma}_j) \subset H_i$. Additionally, for every $1 \leq j < k$ denote by $\vec{\sigma}_j'$ the straight-line segment (sub-path) from X_j' to X_{j+1}. Now, define $\vec{\sigma}$ to be a concatenation of $\vec{\sigma}_1, \vec{\sigma}_1', \ldots, \vec{\sigma}_{k-1}, \vec{\sigma}_{k-1}', \vec{\sigma}_k$. We showed in the previous paragraph that such a path exists in $\mathcal{G}_n$ a.s. Observe that for every $1 \leq j \leq k$ it holds that $\vec{\sigma}_j \subset \mathcal{H}(q_j, q_{j+1})$, where $\mathcal{H}(q_j, q_{j+1}) \subset \mathcal{B}_{\delta/2}(\vec{\sigma}^*)$. This implies that $\mathcal{M}(\vec{\sigma}_i) \leq (1+\varepsilon)\mathcal{M}(\vec{\sigma}^*)$. Additionally, recall that for every $1 \leq j < k$ it holds that $\|X_j' - q_{j+1}\|_2 \leq r_n/2, \|X_{j+1} - q_{j+1}\|_2 \leq r_n/2$, which implies that $\text{Im}(\vec{\sigma}_j') \subset \mathcal{B}_{r_n}(q_{j+1}) \subset \mathcal{B}_\delta(\vec{\sigma}^*)$, and consequently $\mathcal{M}(\sigma_j') \leq (1+\varepsilon)\mathcal{M}(\vec{\sigma}^*)$. Finally, $\mathcal{M}(\vec{\sigma}_n) \leq \mathcal{M}(\vec{\sigma}) \leq (1+\varepsilon)\mathcal{M}(\vec{\sigma}^*)$. This concludes the proof. ◀

6 Experimental results

In this section we validate the theoretical results that were described in the previous section. We observe that the framework can cope with complex scenarios involving two or three degrees of freedom ($d \in \{2, 3\}$), and converges quickly to the optimum.

Before proceeding to the results we provide details regrading the implementation. We implemented the framework in C++, and tested it on scenarios involving two-dimensional objects. Nearest-neighbor search, which is used for the construction of RGGs, was implemented using FLANN [35]. We note that other nearest-neighbor search data structures that are tailored for the implementation of RGGs exist (see, e.g., [31]). Geometric objects, such as points, curves, and polygons were represented with CGAL [49]. For the representation of graphs and related algorithms we used BOOST [42]. Experiments were conducted on a PC with Intel i7-2600 3.4GHz processor with 8GB of memory, running a 64-bit Windows 7 OS.

We proceed to describe the implementation involving the computation of non-trivial cost maps. For curve embedding we used PQP [20] for collision detection, i.e., determining whether a given point lies in the forbidden region $[0, 1]^2 \setminus \mathcal{F}$. Finally, the cost of an edge with respect to a given cost map was approximated by dense sampling along the edge, as is customary in motion planning (see, e.g., [33]).

The majority of running time (over %90) in the experiments below is devoted to the computation of $\mathcal{M}$ for given point samples or edges. Thus, we report only the overall running time in the following experiments. We mention that we also implemented a simple grid-based method for the purpose of comparison with the framework. However, it performed poorly in easy scenarios and did not terminate in hard cases. Thus, we chose to omit theses results here.

Unless stated otherwise, we use in the experiments the connection radius which is described in Theorem 4, and denote it by r_n^*. This applies both to the standard and strong regimes of the problem. A discussion regarding the connection radius in the strong regime appears below in Section 6.3.

6.1 Various scenarios

In this set of experiments we demonstrate the flexibility of the framework and test it on the three different scenarios. We emphasize that we employ a shared code framework to solve these three problems and the ones described later. The only difference in the implementation lies in the type of cost function used. The following problems are solved using a planner for the *strong* case of BPP.

Figure 3 (left) depicts an instance of **P1** (see Section 3.1), which consists of two geometrically-identical curves (red and blue). The curves are bounded in $[0, 1]^2$ and the red curve is translated by $(0.05, 0.05)$ from the blue curve. The optimal solution has a cost of 0.07, in which the curves are traversed identically. Our program was able to produce a solution of cost 0.126 in 27 seconds and $n = 100{,}000$ samples. Results reported throughout this section are the averaged over 10 trials.

Figure 4 (left) depicts an instance of **P2**. The goal is to find a traversal of the three curves such that the traversal point along the purple curve is visible from either the blue or red curve, while of course minimizing the lengths of the leashes between the three curves. Note that the view can be obstructed by the gray rectangular obstacles. A trivial, albeit poor, solution is to move the point along the purple curve from start to end, while the traversal point of, say, the red curve stays put in the start position. A much better solution, which maintains short leashes, is described as follows: we move along the purple curve until reaching the first resting point, indicated by the leftmost black disc. Then we move along the red curve until we reach to the position directly below the black circle. Only then we move along the blue curve from start until reaching the point directly below the first black disc. We use a similar parametrization with respect to the second "pit stop", and so on. Such a solution was obtained by our program in 11 seconds using $n = 20{,}000$ samples.

Figure 4 (right) depicts an instance of **P3**. The input consists of a curve (depicted in red), and polygonal obstacles (depicted in gray). The solution obtained by our program after 600 seconds with $n = 100{,}000$, is drawn in blue.

6.2 Increasing difficulty

Here we focus on **P1** for two curves in the standard regime. We study how the difficulty of the problem affects the running time and the rate of convergence of the returned cost. We start with a base scenario, depicted in Figure 3 (right), and gradually increase its difficulty. In the depicted scenario the bottom (blue) curve consists of five circular loops of radius 0.15, where the entrance and exit point to each circle is indicated by a bullet. The top curve

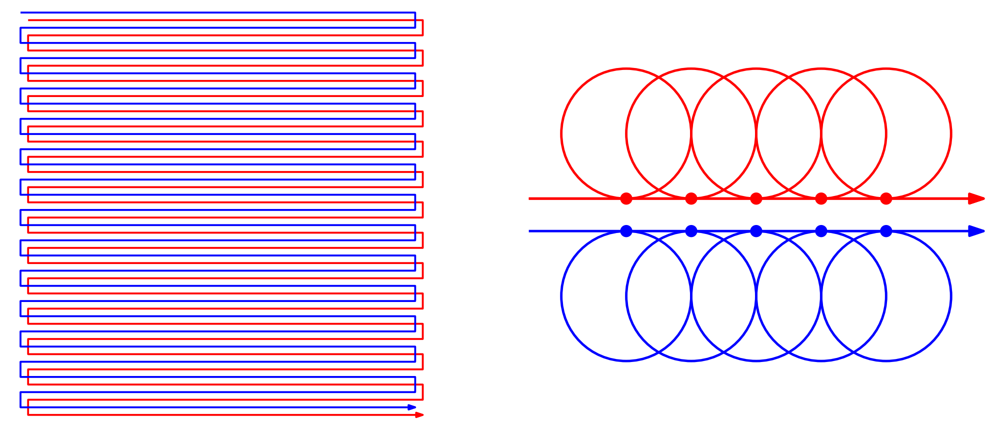


Figure 3 Scenarios involving two curve.

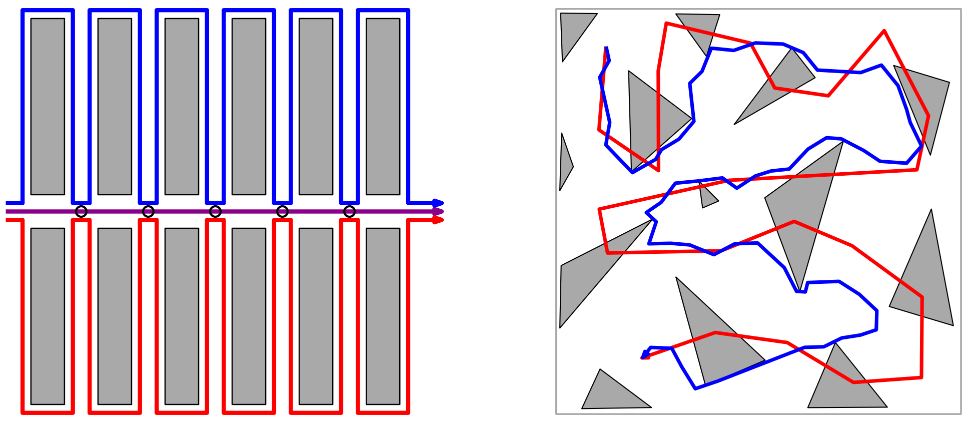


Figure 4 Scenarios involving curves and obstacles.

is similarly defined, and the two curves are separated by a vertical distance of 0.04. The optimal matching of cost 0.34 is obtained in the following manner: when a given circle of the red curve is traversed, the position along the blue curve is fixed to the entrance point of the circle directly below the traversed circle, and vice versa. In a similar fashion we construct scenarios with 10,20,40 and 80 loops in each curve.

In Figure 5 we report for each of the scenarios the cost of the obtained solution as a function of the number of samples n. We set $n = 2^i$ for the integer value i between 12 and 18. For $i = 12$ and $i = 18$ the running times were roughly 2 and 66 seconds, respectively. In between, the values were linearly proportional to the number of samples (results omitted). Observe that as the difficulty of the problem increases the convergence rate of the cost slightly decreases, but overall a value near the optimum is reached fairly quickly.

6.3 Connection radius in the strong regime

Here we consider the *strong* regime and study the behavior of the framework for varying connection radii. For this purpose, we use the two-curves scenario with 20 loops that was described in Section 6.2. We set the connection radius to $r_n := g_n \cdot r_n^*$, where r_n^* is the radius of the standard regime (see Theorem 4). We set $g_n \in \{1, 1.1, \log\log n + 1, \sqrt{\log n}\}$. Results are depicted in Figure 6.

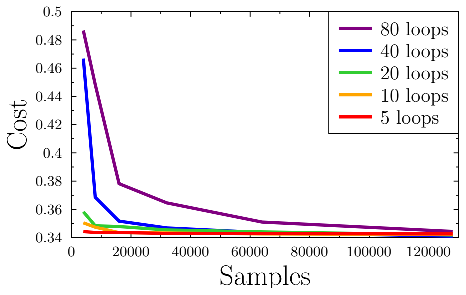


Figure 5 Results for scenarios of increasing difficulty, as described in Section 6.2.

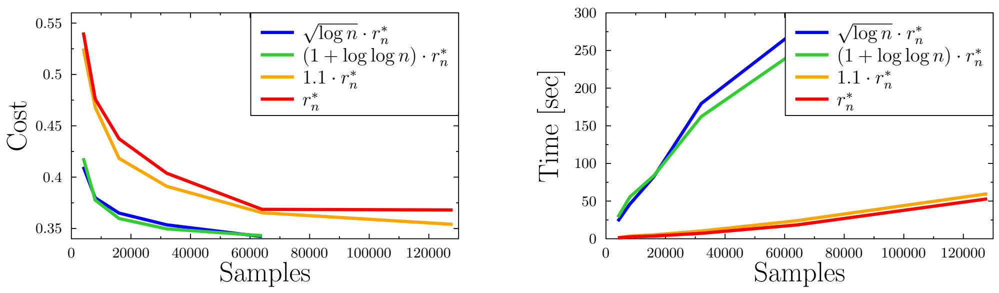


Figure 6 Results for varying connection radii in the strong regime, as described in Section 6.3.

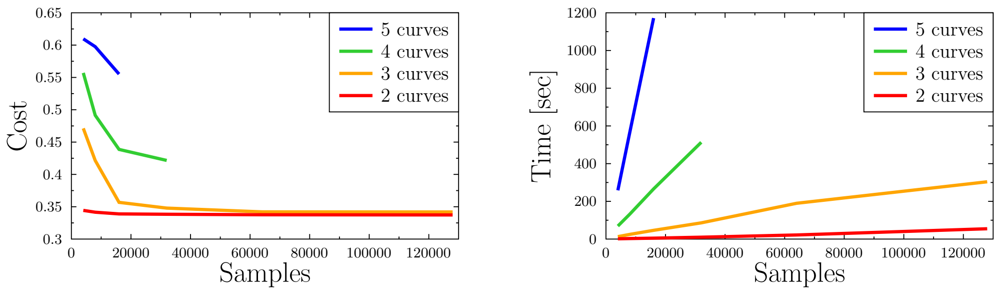


Figure 7 Figures for the first set of experiments, as described in Section 6.4.

Not surprisingly, larger values of r_n lead to quicker convergence, in terms of the number of samples required, to the optimum. However, this comes at the price of a denser RGG, which results in poor running times. Note that the program terminated due to lack of space for the two largest functions of g_n for $n = 128{,}000$. Interestingly, the connection radius r_n^* of the standard regime seems to converge to the optimum, albeit slowly. This leads to the question whether such a function also results in connectivity in the strong regime. Note that our proof of the convergence in the strong regime requires a larger value of r_n (see Theorem 6).

6.4 Increasing dimensionality

We test how the dimension of the configuration space d affects the performance. For this purpose we study the behavior of the framework on *weak* k-curve Fréchet distance with k ranging from 2 to 5. For $k = 2$ we use the scenario described in Section 6.2 with 10 loops. For $k = 3$ we add another copy of the blue curve, for $k = 4$ an additional copy of the red curve, and another blue curve for $k = 5$. We report running time and cost in Figure 7 for various values of n, as described earlier.

Note that that for $k = 4$ the program ran out of memory for $n = 64{,}000$, and for $k = 5$ around $n = 32{,}000$. This phenomena occurs since the connection radius obtained in Theorem 4 grows exponentially in d. In particular, for $r_n = \gamma \left(\frac{\log n}{n}\right)^{1/d}$, where $\gamma = 2(2d\theta_d)^{-1/d}$, each sample has in expectancy $\Theta(2^d \log n)$ neighbors in the obtained RGG.

References

1 Aviv Adler, Mark de Berg, Dan Halperin, and Kiril Solovey. Efficient multi-robot motion planning for unlabeled discs in simple polygons. *IEEE Trans. Automation Science and Engineering*, 12(4):1309–1317, 2015.

2 Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *Int. J. Comput. Geometry Appl.*, 5:75–91, 1995.

3 Oktay Arslan and Panagiotis Tsiotras. Use of relaxation methods in sampling-based algorithms for optimal motion planning. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 2421–2428. IEEE, 2013.

4 Francis Avnaim, Jean-Daniel Boissonnat, and Bernard Faverjon. A practical exact motion planning algorithm for polygonal objects amidst polygonal obstacles. In *Robotics and Automation, 1988. Proceedings., 1988 IEEE International Conference on*, pages 1656–1661. IEEE, 1988.

5 Paul Balister, Amites Sarkar, and Béla Bollobás. Percolation, connectivity, coverage and colouring of random geometric graphs. In Béla Bollobás, Robert Kozma, and Dezso Miklós, editors, *Handbook of Large-Scale Random Networks*, pages 117–142. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

6 Karl Bringmann. Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails. In *Foundations of Computer Science*, pages 661–670, 2014.

7 Karl Bringmann and Wolfgang Mulzer. Approximability of the discrete Fréchet distance. *Journal of Computational Geometry*, 7(2):46–76, 2016.

8 Nicolas Broutin, Luc Devroye, Nicolas Fraiman, and Gábor Lugosi. Connectivity threshold of bluetooth graphs. *Random Struct. Algorithms*, 44(1):45–66, 2014.

9 Kevin Buchin, Maike Buchin, Maximilian Konzack, Wolfgang Mulzer, and André Schulz. Fine-grained analysis of problems on curves. In *EuroCG, Lugano, Switzerland*, 2016.

10 Kevin Buchin, Maike Buchin, Wouter Meulemans, and Wolfgang Mulzer. Four soviets walk the dog – with an application to Alt's conjecture. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 1399–1413, 2014.

11 Kevin Buchin, Maike Buchin, and André Schulz. Fréchet distance of surfaces: Some simple hard cases. In *European Symposium of Algorithms*, pages 63–74, 2010.

12 Kevin Buchin, Maike Buchin, Rolf van Leusden, Wouter Meulemans, and Wolfgang Mulzer. Computing the Fréchet distance with a retractable leash. In *European Symposium of Algorithms*, pages 241–252, 2013.

13 Kevin Buchin, Maike Buchin, and Carola Wenk. Computing the Fréchet distance between simple polygons. *Comput. Geom.*, 41(1-2):2–20, 2008.

14 Erin W. Chambers, Éric Colin de Verdière, Jeff Erickson, Sylvain Lazard, Francis Lazarus, and Shripad Thite. Homotopic Fréchet distance between curves or, walking your dog in the woods in polynomial time. *Comput. Geom.*, 43(3):295–311, 2010.

15 Shiri Chechik, Haim Kaplan, Mikkel Thorup, Or Zamir, and Uri Zwick. Bottleneck paths and trees and deterministic graphical games. In *Symposium on Theoretical Aspects of Computer Science*, pages 27:1–27:13, 2016.

16 Atlas F. Cook and Carola Wenk. Geodesic Fréchet distance inside a simple polygon. *ACM Transactions on Algorithms*, 7(1):9, 2010.

17 Mark de Berg and Marc J. van Kreveld. Trekking in the alps without freezing or getting tired. *Algorithmica*, 18(3):306–323, 1997.

18 Adrian Dumitrescu and Günter Rote. On the Fréchet distance of a set of curves. In *Canadian Conference on Computational Geometry*, pages 162–165, 2004.

19 Chenglin Fan, Omrit Filtser, Matthew J. Katz, Tim Wylie, and Binhai Zhu. On the chain pair simplification problem. In *Symposium on Algorithms and Data Structures*, pages 351–362, 2015.

20 GAMMA group. PQP – a proximity query package, 1999. University of North Carolina at Chapel Hill, USA.

21 Dan Halperin and Micha Sharir. A near-quadratic algorithm for planning the motion of a polygon in a polygonal environment. *Discrete & Computational Geometry*, 16(2):121–134, 1996.

22 Sariel Har-Peled and Benjamin Raichel. The Fréchet distance revisited and extended. *ACM Transactions on Algorithms*, 10(1):3, 2014.

23 John E. Hopcroft, Jacob T. Schwartz, and Micha Sharir. On the complexity of motion planning for multiple independent objects; PSPACE-hardness of the "Warehouseman's problem". *International Journal of Robotics Research*, 3(4):76–88, 1984.

24 David Hsu, Jean-Claude Latombe, and Rajeev Motwani. Path planning in expansive configuration spaces. *Int. J. Comput. Geometry Appl.*, 9(4/5):495–512, 1999.

25 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

26 Lucas Janson, Edward Schmerling, Ashley A. Clark, and Marco Pavone. Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *I. J. Robotic Res.*, 34(7):883–921, 2015.

27 Minghui Jiang, Ying Xu, and Binhai Zhu. Protein structure–structure alignment with discrete Fréchet distance. *Journal of bioinformatics and computational biology*, 6(01):51–64, 2008.

28 Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research*, 30(7):846–894, 2011.

29 Lydia E. Kavraki, Petr Švestka, Jean-Claude Latombe, and Mark H. Overmars. Probabilistic roadmaps for path planning in high dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

30 Lutz Kettner, Kurt Mehlhorn, Sylvain Pion, Stefan Schirra, and Chee-Keng Yap. Classroom examples of robustness problems in geometric computations. *Comput. Geom.*, 40(1):61–78, 2008.

31 Michal Kleinbort, Oren Salzman, and Dan Halperin. Efficient high-quality motion planning by fast all-pairs r-nearest-neighbors. In *IEEE International Conference on Robotics and Automation*, pages 2985–2990, 2015.

32 James J. Kuffner and Steven M. LaValle. RRT-Connect: An efficient approach to single-query path planning. In *International Conference on Robotics and Automation (ICRA)*, pages 995–1001, 2000.

33 S. M. LaValle. *Planning algorithms.* Cambridge University Press, 2006.

34 Wouter Meulemans. Map matching with simplicity constraints. *CoRR*, abs/1306.2827, 2013.

35 M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISSAPP*, pages 331–340. INSTICC Press, 2009.

36 Mathew Penrose. *Random geometric graphs*, volume 5. Oxford University Press, 2003.

37 Barak Raveh, Angela Enosh, Ora Schueler-Furman, and Dan Halperin. Rapid sampling of molecular motions with prior information constraints. *PLoS Computational Biology*, 5(2), 2009.

38 John H Reif. Complexity of the mover's problem and generalizations: Extended abstract. In *Foundations of Computer Science*, pages 421–427, 1979.

39 Oren Salzman and Dan Halperin. Asymptotically-optimal motion planning using lower bounds on cost. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4167–4172, 2015.

40 Micha Sharir. Algorithmic motion planning. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry, Second Edition.*, pages 1037–1064. Chapman and Hall/CRC, 2004.

41 Jessica Sherette and Carola Wenk. Simple curve embedding. *CoRR*, abs/1303.0821, 2013.

42 J. G. Siek, L.-Q. Lee, and A. Lumsdaine. *The Boost Graph Library User Guide and Reference Manual*. Addison-Wesley, 2002.

43 Kiril Solovey and Dan Halperin. On the hardness of unlabeled multi-robot motion planning. In *Robotics: Science and Systems (RSS)*, 2015.

44 Kiril Solovey, Oren Salzman, and Dan Halperin. Finding a needle in an exponential haystack: Discrete RRT for exploration of implicit roadmaps in multi-robot motion planning. *International Journal of Robotics Research*, 35(5):501–513, 2016.

45 Kiril Solovey, Oren Salzman, and Dan Halperin. New perspective on sampling-based motion planning via random geometric graphs. In *Robotics: Science and Systems (RSS)*, Ann Arbor, Michigan, 2016. doi:10.15607/RSS.2016.XII.003.

46 Kiril Solovey, Jingjin Yu, Or Zamir, and Dan Halperin. Motion planning for unlabeled discs with optimality guarantees. In *Robotics: Science and Systems (RSS)*, 2015.

47 Paul G. Spirakis and Chee-Keng Yap. Strong NP-hardness of moving many discs. *Information Processing Letters*, 19(1):55–59, 1984.

48 R Sriraghavendra, K Karthik, and Chiranjib Bhattacharyya. Fréchet distance based approach for searching online handwritten documents. In *Document Analysis and Recognition*, volume 1, pages 461–465. IEEE, 2007.

49 The CGAL Project. *CGAL user and reference manual*. CGAL editorial board, 4.8 edition, 2016. URL: http://www.cgal.org/.

50 Matthew Turpin, Nathan Michael, and Vijay Kumar. Concurrent assignment and planning of trajectories for large teams of interchangeable robots. In *International Conference on Robotics and Automation (ICRA)*, pages 842–848, 2013.

51 Mark Walters. Random geometric graphs. In Robin Chapman, editor, *Surveys in Combinatorics 2011*, chapter 8, pages 365–401. Cambridge University Press, 2011.

52 Virginia Vassilevska Williams. *Efficient Algorithms for Path Problems in Weighted Graphs*. Ph.D. thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA, 2008.

53 Jianbin Zheng, Xiaolei Gao, Enqi Zhan, and Zhangcan Huang. Algorithm of on-line handwriting signature verification based on discrete Fréchet distance. In *Advances in Computation and Intelligence*, pages 461–469. Springer, 2008.

On the Geodesic Centers of Polygonal Domains

Haitao Wang[*]

Department of Computer Science, Utah State University, Logan, UT, USA
haitao.wang@usu.edu

Abstract

In this paper, we study the problem of computing Euclidean geodesic centers of a polygonal domain $\mathcal{P}$ of n vertices. We give a necessary condition for a point being a geodesic center. We show that there is at most one geodesic center among all points of $\mathcal{P}$ that have topologically-equivalent shortest path maps. This implies that the total number of geodesic centers is bounded by the size of the shortest path map equivalence decomposition of $\mathcal{P}$, which is known to be $O(n^{10})$. One key observation is a *π-range property* on shortest path lengths when points are moving. With these observations, we propose an algorithm that computes all geodesic centers in $O(n^{11}\log n)$ time. Previously, an algorithm of $O(n^{12+\epsilon})$ time was known for this problem, for any $\epsilon > 0$.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, I.1.2 Algorithms, I.3.5 Computational Geometry and Object Modeling

Keywords and phrases geodesic centers, shortest paths, polygonal domains

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.77

1 Introduction

Let $\mathcal{P}$ be a polygonal domain with a total of h holes and n vertices, i.e., $\mathcal{P}$ is a multiply-connected region whose boundary is a union of n line segments, forming $h+1$ closed polygonal cycles. A simple polygon is a special case of a polygonal domain with $h = 0$. For any two points s and t, a *shortest path* or *geodesic path* from s to t is a path in $\mathcal{P}$ whose Euclidean length is minimum among all paths from s to t in $\mathcal{P}$; we let $d(s,t)$ denote the Euclidean length of any shortest path from s to t and we also say that $d(s,t)$ is the *shortest path distance or geodesic distance* from s to t.

A point s is a *geodesic center* of $\mathcal{P}$ if s minimizes the value $\max_{t\in\mathcal{P}} d(s,t)$, i.e., the maximum geodesic distance from s to all points of $\mathcal{P}$.

In this paper, we study the problem of computing the geodesic centers of $\mathcal{P}$. The problem in simple polygons has been well studied. It is known that for any point in a simple polygon, its farthest point must be a vertex of the polygon [20]. It has been shown that the geodesic center in a simple polygon is unique and has at least two farthest points [18]. Due to these helpful observations, efficient algorithms for finding geodesic centers in simple polygons have been developed. Asano and Toussaint [2] gave an $O(n^4\log n)$ time algorithm for the problem, and later Pollack, Sharir, and Rote [18] improved the algorithm to $O(n\log n)$ time. Recently, Ahn et al. [1] solved the problem in linear time.

Finding a geodesic center in a polygonal domain $\mathcal{P}$ is much more difficult. This is partially due to that a farthest point of a point in $\mathcal{P}$ may be in the interior of $\mathcal{P}$ [3]. Also, it is easy to construct an example where the geodesic center of $\mathcal{P}$ is not unique (e.g., see Fig. 1). Bae, Korman, and Okamoto [4] gave the first-known algorithm that can compute a geodesic center

[*] This research was supported in part by NSF under Grant CCF-1317143.

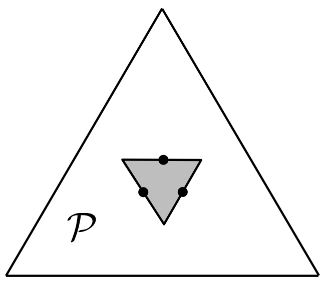


Figure 1 The boundary of $\mathcal{P}$ consists of an outer and inner equilateral triangles with their geometric centers co-located. Each of the three thick points is a geodesic center of $\mathcal{P}$.

in $O(n^{12+\epsilon})$ time for any $\epsilon > 0$. They first showed that for any point its farthest points must be vertices of its shortest path map in $\mathcal{P}$. Then, they considered the shortest path map equivalence decomposition (or SPM-equivalence decomposition) [7], denoted by $\mathcal{D}_{spm}$; for each cell of $\mathcal{D}_{spm}$, they computed the upper envelope of $O(n)$ graphs in 3D space, which takes $O(n^{2+\epsilon})$ time [10], to search a geodesic center in the cell. Since the size of $\mathcal{D}_{spm}$ is $O(n^{10})$ [7], their algorithm runs in $O(n^{12+\epsilon})$ time.

A closely related concept is the *geodesic diameter*, which is the maximum geodesic distance over all pairs of points in $\mathcal{P}$, i.e., $\max_{s,t,\in\mathcal{P}} d(s,t)$. In simple polygons, due to the property that there always exists a pair of vertices of $\mathcal{P}$ whose geodesic distance is equal to the geodesic diameter, efficient algorithms have been given for computing the geodesic diameters. Chazelle [6] gave the first algorithm that runs in $O(n^2)$ time. Later, Suri [20] presented an $O(n \log n)$-time algorithm. The problem was eventually solved in $O(n)$ time by Hershberger and Suri [11]. Computing the geodesic diameter in a polygonal domain $\mathcal{P}$ is much more difficult, partially because the diameter can be realized by two points in the interior of $\mathcal{P}$, in which case there are at least five distinct shortest paths between the two points [3]. As for the geodesic center, this makes it difficult to discretize the search space. By an exhaustive-search method, Bae, Korman, and Okamoto [3] gave the first algorithm for computing the diameter of $\mathcal{P}$, which runs in $O(n^{7.73})$ or $O(n^7(\log n + h))$ time.

Refer to [5, 8, 13, 15, 16, 17, 19] for other variations of geodesic diameter and center problems (e.g., the L_1 metric and the link distance case).

1.1 Our Contributions

We conduct a "comprehensive" study on geodesic centers of $\mathcal{P}$. We discover many interesting observations, and some of them may be even surprising. For example, we show that even if a geodesic center is in the interior of $\mathcal{P}$, it may have only one farthest point, which is somewhat counter-intuitive. We give a necessary condition for a point being a geodesic center. We show that there is at most one geodesic center among all points of $\mathcal{P}$ that have topologically-equivalent shortest path maps in $\mathcal{P}$. This immediately implies that the interior of each cell or each edge of the SPM-equivalence decomposition $\mathcal{D}_{spm}$ can contain at most one geodesic center, and thus, the total number of geodesic centers of $\mathcal{P}$ is bounded by the combinatorial size of $\mathcal{D}_{spm}$, which is known to be $O(n^{10})$ [7]. Previously, the only known upper bound on the total number of geodesic centers of $\mathcal{P}$ is $O(n^{12})$, which is suggested by the algorithm in [4].

These observations are all more or less due to an interesting observation, which we call the *π-range property* and is one key contribution of this paper. Here we only demonstrate an application of the π-range property. Let s and t be two points in the interior of $\mathcal{P}$ such

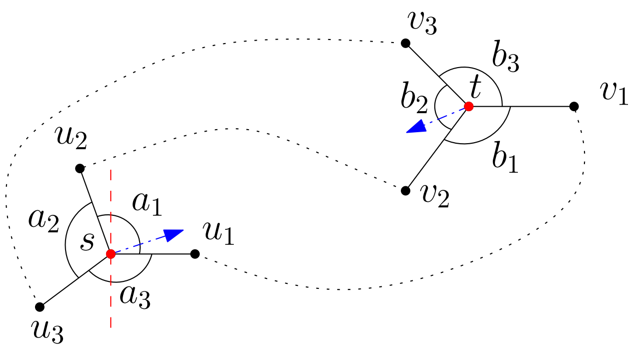


Figure 2 Illustrating the π-range property. Suppose there are three shortest s-t paths through vertices u_i and v_i with $i = 1, 2, 3$, respectively. If s and t move along the blue arrows simultaneously (possibly with different speeds), then all three shortest paths strictly decreases (it is difficult to tell whether this is true from the figure, so these two blue directions here are only for illustration purpose). The special case happens when the six angles a_i and b_i satisfy $a_i = b_i$ for $i = 1, 2, 3$.

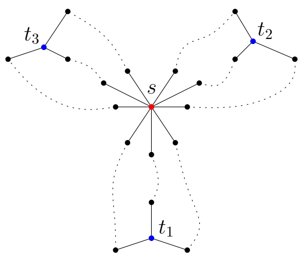


Figure 3 Illustrating a geodesic center s with three farthest points t_1, t_2, t_3 such that all these four points are in the interior of $\mathcal{P}$. There are three shortest paths from s to each of t_1, t_2, t_3.

that t is a farthest point of s in $\mathcal{P}$. Refer to Fig. 2 for an example. Suppose there are three shortest paths from s to t as shown in Fig. 2. The π-range property says that unless a special case happens, there exists an open range of exactly size π (e.g., delimited by the right open half-plane bounded by the vertical line through s in Fig. 2) such that if s moves along any direction in the range for an infinitesimal distance, we can always find a direction to move t such that the lengths of *all three* shortest paths strictly decrease. Further, if the special case does not happen, we can explicitly determine the above range of size π. In fact, it is the special case that makes it possible for a geodesic center having only one farthest point.

With these observations, we propose an exhaustive-search algorithm to compute a set S of *candidate points* such that all geodesic centers must be in S. For example, refer to Fig. 3, where a geodesic center s has three farthest points t_1, t_2, t_3 and all these four points are in the interior of $\mathcal{P}$. The nine shortest paths from s to t_1, t_2, t_3 provide a system of eight equations, which give eight (independent) constraints that can determine the four points s, t_1, t_2, t_3 if we consider the coordinates of these points as eight variables. This suggests our exhaustive-search to compute candidate points for such a geodesic center s. However, if a geodesic center s has only one farthest point (e.g., Fig. 2), then we have only three shortest paths (in the non-degenerate case), which give only two constraints. In order to determine s and t, which have four variables, we need two more constraints. It turns out the π-range property (i.e., the special case) provides exactly two more constraints (on the angles as shown in Fig. 2). In this way, we can still compute candidate points for such s. Also, if s has two farthest points, we will need one more constraint, which is also provided by the π-range property (the non-special case).

The number of candidate points in S is $O(n^{11})$. To find all geodesic centers from S, a straightforward solution is to compute the shortest path map for every point of S, which takes $O(n^{12} \log n)$ time in total. Again, with the help of the π-range property, we propose a *pruning algorithm* to eliminate most points from S in $O(n^{11} \log n)$ time such that none of the eliminated points is a geodesic center and the number of the remaining points of S is only $O(n^{10})$. Consequently, we can find all geodesic centers in additional $O(n^{11} \log n)$ time.

Although we improve the previous $O(n^{12+\epsilon})$ time algorithm in [4] by a factor of roughly $n^{1+\epsilon}$, the running time is still huge. We feel that our observations (e.g., the π-range property) may be more interesting than the algorithm itself. We suspect that they may also find other applications. The paper is lengthy, which is mainly due to a considerable number of cases depending on whether a geodesic center and its farthest points are in the interior, on an edge, or at vertices of $\mathcal{P}$, although the essential idea is quite similar for all these cases.

The rest of the paper is organized as follows. In Section 2, we introduce notation and review some concepts. In Section 3, we give our observations. We present the π-range property in Section 4. Computing the candidate points is discussed in Section 5. Finally, we find all geodesic centers from the candidate points in Section 6. Due to the space limit, many details and proofs are omitted but can be found in the full paper.

2 Preliminaries

Consider any point $s \in \mathcal{P}$. Let $d_{\max}(s)$ be the maximum geodesic distance from s to all points of $\mathcal{P}$, i.e., $d_{\max}(s) = \max_{t\in\mathcal{P}} d(s,t)$. A point $t \in \mathcal{P}$ is a *farthest point* of s if $d(s,t) = d_{\max}(s)$. Let $F(s)$ denote the set of all farthest points of s. For any two points p and q in $\mathcal{P}$, we say that p is *visible* to q if the line segment $\overline{pq}$ is in $\mathcal{P}$ and the interior of $\overline{pq}$ does not contain any vertex of $\mathcal{P}$. We use $|pq|$ to denote the (Euclidean) length of any line segment $\overline{pq}$. Note that two points s and t in $\mathcal{P}$ may have more than one shortest path between them, and if not specified, we use $\pi(s,t)$ to denote any such shortest path.

For simplicity of discussion, we make a general position assumption that any two vertices of $\mathcal{P}$ have only one shortest path and no three vertices of $\mathcal{P}$ are on the same line.

Denote by $\mathcal{I}$ the set of all interior points of $\mathcal{P}$, $\mathcal{V}$ the set of all vertices of $\mathcal{P}$, and E the set of all relatively interior points on the edges of $\mathcal{P}$ (i.e., E is the boundary of $\mathcal{P}$ minus $\mathcal{V}$).

Shortest path maps. A *shortest path map* of a point $s \in \mathcal{P}$ [7], denoted by $\mathit{SPM}(s)$, is a decomposition of $\mathcal{P}$ into regions (or cells) such that in each cell σ, the combinatorial structures of shortest paths from s to all points t in σ are the same, and more specifically, the sequence of obstacle vertices along $\pi(s,t)$ is fixed for all t in σ. Further, the *root* of σ, denoted by $r(\sigma)$, is the last vertex of $\mathcal{V} \cup \{s\}$ in the path $\pi(s,t)$ for any point $t \in \sigma$ (hence $\pi(s,t) = \pi(s, r(\sigma)) \cup \overline{r(\sigma)t}$; note that $r(\sigma)$ is s if s is visible to t). As in [7], we classify each edge of σ into three types: a portion of an edge of $\mathcal{P}$, *an extension segment*, which is a line segment extended from $r(\sigma)$ along the opposite direction from $r(\sigma)$ to the vertex of $\pi(s,t)$ preceding $r(\sigma)$, and *a bisector curve/edge* that is a hyperbolic arc. For each point t in a bisector edge of $\mathit{SPM}(s)$, t is on the common boundary of two cells and there are two shortest paths from s to t through the roots of the two cells, respectively (and neither path contains both roots). The *vertices* of $\mathit{SPM}(s)$ include $\mathcal{V} \cup \{s\}$ and all intersections of edges of $\mathit{SPM}(s)$. If a vertex t of $\mathit{SPM}(s)$ is an intersection of two or more bisector edges, then there are more than two shortest paths from s to t. The map $\mathit{SPM}(s)$ has $O(n)$ vertices, edges, and cells, and can be computed in $O(n \log n)$ time [12]. It was shown [4] that any farthest point of s in $\mathcal{P}$ must be a vertex of $\mathit{SPM}(s)$. For differentiation, we will refer to the vertices of $\mathcal{V}$ as *polygon vertices* and refer to the edges of E as *polygon edges*.

The SPM-equivalence decomposition $\mathcal{D}_{spm}$ [7] is a subdivision of $\mathcal{P}$ into regions such that for all points s in the interior of the same region or edge of $\mathcal{D}_{spm}$, the shortest path maps of s are topologically equivalent. Chiang and Mitchell [7] showed that the combinatorial size of $\mathcal{D}_{spm}$ is bounded by $O(n^{10})$ and $\mathcal{D}_{spm}$ can be computed in $O(n^{11})$ time.

Directions and ranges. We will have intensive discussions on moving points along certain directions. For any direction r, we represent r by the angle $\alpha(r) \in [0, 2\pi)$ counterclockwise from the positive direction of the x-axis. For convenience, whenever we are talking about an angle α, unless otherwise specified, depending on the context we may refer to any angle $\alpha + 2\pi \cdot k$ for $k \in \mathbb{Z}$. For any two angles α_1 and α_2 with $\alpha_1 \leq \alpha_2 < \alpha_1 + 2\pi$, the interval $[\alpha_1, \alpha_2]$ represents a *direction range* that includes all directions whose angles are in $[\alpha_1, \alpha_2]$, and $\alpha_2 - \alpha_1$ is called the *size* of the range. Note that the range can also be open (e.g., (α_1, α_2)) and the size of any direction range is no more than 2π.

Consider a half-plane h whose bounding line is through a point s in the plane. We say h *delimits* a range of size π of directions for s that consists of all directions along which s will move towards inside h. If h is an open half-plane, then the range is open as well.

A direction r for $s \in \mathcal{P}$ is called a *free direction* of s if we move s along r for an infinitesimal distance then s is still in $\mathcal{P}$. We use $R_f(s)$ to denote the range of all free directions of s. Clearly, if $s \in \mathcal{I}$, $R_f(s)$ contains all directions; if $s \in E$, $R_f(s)$ is a (closed) range of size π; if $s \in V$, $R_f(s)$ is delimited by the two incident polygon edges of s.

3 Observations

Consider any point $s \in \mathcal{P}$ and let t be any farthest point of s. Recall that t is a vertex of $\mathit{SPM}(s)$ [4]. Suppose we move s infinitesimally along a free direction r to a new point s'. Since $|ss'|$ is infinitesimal, we can assume that s and s' are in the same cell σ of $\mathcal{D}_{spm}$. Further, if s is in the interior of σ, then s' is also in the interior of σ. Regardless of whether s is in the interior of σ or not, there is a vertex $t' \in \mathit{SPM}(s')$ *corresponding to* the vertex t of $\mathit{SPM}(s)$ in the following sense [7]: If the line segment $\overline{s't'}$ is a shortest path from s' to t', then $\overline{st}$ is a shortest path from s to t; otherwise, if $s', u_1, u_2, \ldots, u_k, t'$ is the sequence of the vertices of $\mathcal{V} \cup \{s', t'\}$ in a shortest path from s' to t', then $s, u_1, u_2, \ldots, u_k, t$ is also the sequence of the vertices of $\mathcal{V} \cup \{s, t\}$ in a shortest path from s to t.

In the case that s is on the boundary of σ while s' is in the interior of σ, there might be more than one such vertex $t' \in \mathcal{D}_{spm}$ corresponding to t (refer to [7] for the details) and we use $M_t(s')$ to denote the set of all such vertices t'. We should point out that although a vertex in $\mathit{SPM}(s)$ may correspond to more than one vertex in $\mathit{SPM}(s')$, any vertex in $\mathit{SPM}(s')$ can correspond to one and only one vertex in $\mathit{SPM}(s)$ (because s' is always in the interior of σ).

We introduce the following definition which is crucial to the paper.

▶ **Definition 1.** A free direction r is an *admissible direction* of s with respect to t if as we move s infinitesimally along r to a new point s', $d(s', t') < d(s, t)$ holds for each $t' \in M_t(s')$.

For any $t \in F(s)$, let $R(s,t)$ denote the set of all admissible directions of s with respect to t; let $R(s) = \bigcap_{t \in F(s)} R(s,t)$. The following Lemma 3, which gives a necessary condition for a point being a geodesic center of $\mathcal{P}$, explains why we consider admissible directions. Before presenting Lemma 3, we introduce some notation and Observation 2.

Consider any two points s and t in $\mathcal{P}$. Suppose the vertices of $\mathcal{V} \cup \{s, t\}$ along a shortest s-t path $\pi(s,t)$ are $s = u_0, u_1, \ldots, u_k = t$. By our definition of "visibility", s is visible to t if and only if $k = 1$. If s is not visible to t, then $k \neq 1$ and we call u_1 an *s-pivot* and u_{k-1} a *t-pivot* of $\pi(s,t)$. It is possible that there are multiple shortest paths between s and t, and thus there might be multiple s-pivots and t-pivots for (s,t). We use $U_s(t)$ and $U_t(s)$ to denote the sets of all s-pivots and t-pivots for (s,t), respectively. By our above definition, for any $u \in U_s(t)$, the line segment $\overline{su}$ does not contain any polygon vertex in its interior.

We have the following observation. Similar results have been given in [3].

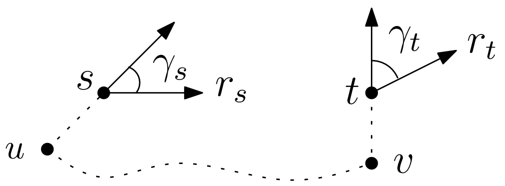


Figure 4 The definitions of the angles γ_s and γ_t.

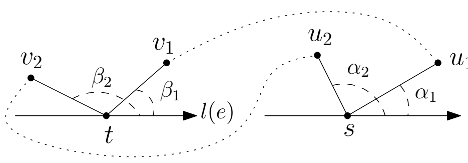


Figure 5 The definitions of the angles.

▶ **Observation 2.** Suppose t is a farthest point of a point s.

1. If t is in $\mathcal{I}$, then $|U_t(s)| \geq 3$ and t must be in the interior of the convex hull of the vertices of $U_t(s)$.
2. If t is in E, say, $t \in e$ for a polygon edge e of E, then $|U_t(s)| \geq 2$ and $U_t(s)$ has at least one vertex in the open half-plane bounded by the supporting line of e and containing the interior of $\mathcal{P}$ in the small neighborhood of e. Further, $U_t(s)$ has at least one vertex in each of the two open half-planes bounded by the line through t and perpendicular to e.

▶ **Lemma 3.** *If s is a geodesic center of $\mathcal{P}$, then $R(s) = \emptyset$.*

As explained in Section 1, we will compute candidate points for geodesic centers. As a necessary condition, Lemma 3 will be helpful for computing those candidate points.

Consider any point $s \in \mathcal{P}$. Let t be a farthest point of s. To determine the admissible direction range $R(s,t)$, we will give a sufficient condition for a direction being in $R(s,t)$. We first assume that s is not visible to t, and as will be seen later, the other case is trivial.

Let u and v respectively be the s-pivot and the t-pivot of (s,t) in a shortest s-t path $\pi(s,t)$. Clearly, $d(s,t) = |su| + d(u,v) + |vt|$. We define $d_{u,v}(s,t) = |su| + d(u,v) + |vt|$ as a function of $s \in \mathbb{R}^2$ and $t \in \mathbb{R}^2$. Suppose we move s along a free direction r_s with the unit speed and move t along a free direction r_t with a speed $\tau \geq 0$. Let γ_s denote the smaller angle between the following two rays originated from s (e.g., see Fig. 4): one with direction r_s and one with direction from u to s. Similarly, let γ_t denote the smaller angle between the following two rays originated from t: one with direction r_t and one with direction from v to t. In fact, as discussed in [3], if we consider $d(s,t)$ as a four-variate function, the triple (r_s, r_t, τ) corresponds to a vector ρ in $\mathbb{R}^4$, and the directional derivative of $d_{u,v}(s,t)$ at $(s,t) \in \mathbb{R}^4$ along ρ, denoted by $d'_{u,v}(s,t)$, and the second directional derivative of $d_{u,v}(s,t)$ at (s,t) along ρ, denoted by $d''_{u,v}(s,t)$, are

$$d'_{u,v}(s,t) = \cos\gamma_s + \tau\cos\gamma_t, \quad d''_{u,v}(s,t) = \frac{\sin^2\gamma_s}{|su|} + \tau \cdot \frac{\sin^2\gamma_t}{|tv|}. \tag{1}$$

Since $\tau \geq 0$, $d''_{u,v}(s,t) \geq 0$. Further, if $\tau \neq 0$, then $d''_{u,v}(s,t) = 0$ if and only if $\sin^2\gamma_s = \sin^2\gamma_t = 0$, i.e., each of γ_s and γ_t is either 0 or π. Below, in order to make the discussions more intuitive, we choose to use the parameters r_s, r_t, and τ, instead of the vectors of $\mathbb{R}^4$.

For each vertex $u \in U_s(t)$, there must be a vertex $v \in U_t(s)$ such that the concatenation of $\overline{su}$, $\pi(u,v)$, and $\overline{vt}$ is a shortest path from s to t, and we call such a vertex v a *coupled t-pivot* of u (if u has more than one such vertex, then all of them are coupled t-pivots of u). Similarly, for each vertex $v \in U_t(s)$, we also define its coupled s-pivots in $U_s(t)$.

The following lemma provides a sufficient condition for a direction being in $R(s,t)$.

▶ **Lemma 4.** *Suppose t is a farthest point of s and s is not visible to t.*

1. *For $t \in \mathcal{I}$, a free direction r_s is in $R(s,t)$ if there is a free direction r_t for t with a speed $\tau \geq 0$ such that when we move s along r_s with unit speed and move t along r_t with speed τ, each vertex $v \in U_t(s)$ has a coupled s-pivot u with either $d'_{u,v}(s,t) < 0$, or $d'_{u,v}(s,t) = 0$ and $d''_{u,v}(s,t) = 0$.*
2. *For $t \in E$, a free direction r_s is in $R(s,t)$ if there is a free direction r_t for t that is parallel to the polygon edge of E containing t with a speed $\tau \geq 0$ such that when we move s along r_s with the unit speed and move t along r_t with speed τ, each vertex $v \in U_t(s)$ has a coupled s-pivot u with either $d'_{u,v}(s,t) < 0$, or $d'_{u,v}(s,t) = 0$ and $d''_{u,v}(s,t) = 0$.*
3. *For $t \in \mathcal{V}$, a free direction r_s is in $R(s,t)$ if we move s along r_s with the unit speed, each vertex $v \in U_t(s)$ has a coupled s-pivot u with either $d'_{u,v}(s,t) < 0$, or $d'_{u,v}(s,t) = 0$ and $d''_{u,v}(s,t) = 0$.*

Lemma 4 is on the case where s is not visible to t. If s is visible to t, the result is trivial.

▶ **Observation 5.** Suppose t is a farthest point of s and s is visible to t. Then t must be a polygon vertex of $\mathcal{V}$. Further, a free direction r_s of s is in $R(s,t)$ if and only if r_s is towards the interior of $h_s(t)$, where $h_s(t)$ is the open half-plane containing t and bounded by the line through s and perpendicular to $\overline{st}$.

By Observation 5, if s is visible to t, then the range $R(s,t)$ is the intersection of the free direction range $R_f(s)$ and an open range of size π delimited by the open half-plane $h_s(t)$.

The next lemma is proved by using Lemmas 3 and 4 as well as Observation 5.

▶ **Lemma 6.** *Among all points of $\mathcal{P}$ that have topologically equivalent shortest path maps in $\mathcal{P}$, there is at most one geodesic center. This implies that each cell or edge of $\mathcal{D}_{spm}$ contains at most one geodesic center in its interior, which further implies that the number of geodesic centers of $\mathcal{P}$ is $O(|\mathcal{D}_{spm}|)$, where $|\mathcal{D}_{spm}|$ is the combinatorial complexity of $\mathcal{D}_{spm}$.*

The following corollary implies that if t is a farthest point of s, then slightly moving s along a free direction that is not in $R(s,t)$ can never obtain a geodesic center.

▶ **Corollary 7.** *Suppose t is a farthest point of s. If we move s infinitesimally along a free direction that is not in $R(s,t)$, then $d_{\max}(s)$ will become strictly larger.*

To compute the candidate points, we need to determine $R(s,t)$. It turns out that it is sufficient to determine $R(s,t)$ when t is at a non-degenerate position of s in the following sense: Suppose t is a farthest point of s; we say that t is *non-degenerate* with respect to s if there are exactly three, two, and one shortest s-t paths for t in $\mathcal{I}$, E, and $\mathcal{V}$, respectively (by Observation 2, this implies that $|U_t(s)|$ is 3, 2, and 1, respectively for the three cases).

Lemma 4 gives a sufficient condition for a direction in $R(s,t)$. The following lemma gives both a sufficient and a necessary condition for a direction in $R(s,t)$ when t is non-degenerate, and the lemma will be used to explicitly compute the range $R(s,t)$ in Section 4. Note that Observation 5 already gives a way to determine $R(s,t)$ when s is visible to t.

▶ **Lemma 8.** *Suppose t is a non-degenerate farthest point of s and s is not visible to t. Then, a free direction r_s is in $R(s,t)$ if and only if*

1. *for $t \in \mathcal{I}$, there is a free direction r_t for t with a speed $\tau \geq 0$ such that when we move s along r_s with unit speed and move t along r_t with speed τ, each vertex $v \in U_t(s)$ has a coupled s-pivot u with $d'_{u,v}(s,t) < 0$.*
2. *for $t \in E$, there is a free direction r_t for t that is parallel to the polygon edge containing t with a speed $\tau \geq 0$ such that when we move s along r_s with unit speed and move t along r_t with speed τ, each vertex $v \in U_t(s)$ has a coupled s-pivot u with $d'_{u,v}(s,t) < 0$.*
3. *for $t \in \mathcal{V}$, when we move s along r_s with unit speed, each vertex $v \in U_t(s)$ has a coupled s-pivot u with $d'_{u,v}(s,t) < 0$.*

4 Determining the Range $R(s,t)$ and the π-Range Property

In this section, we determine the admissible direction range $R(s,t)$ for any point s and any of its non-degenerate farthest point t. In particular, we will present the π-range property.

Depending on whether t is in $\mathcal{V}$, E, and $\mathcal{I}$, there are three cases. Recall that $R_f(s)$ is the range of free directions of s. In each case, we will show that $R(s,t)$ is the intersection of $R_f(s)$ and an open range $R_\pi(s,t)$ of size π. We call $R_\pi(s,t)$ the *π-range*. As will be seen later, $R_\pi(s,t)$ can be explicitly determined based on the positions of s, t, and the vertices of $U_s(t)$ and $U_t(s)$. In fact, for each case, we will give a more general result on shortest path distance functions. These general results will be useful for computing the candidate points.

4.1 The Case $t \in \mathcal{V}$

We first discuss the case $t \in \mathcal{V}$. The result is relatively straightforward in this case. If s is visible to t, the π-range $R_\pi(s,t)$ is defined to be the open range of directions delimited by the open half-plane $h_s(t)$ as defined in Observation 5; by Observation 5, $R(s,t) = R_f(s) \cap R_\pi(s,t)$.

Below, we assume s is not visible to t. We first present a more general result on a shortest path distance function. Let s and t be any two points in $\mathcal{P}$ such that t is in $\mathcal{V}$ and s is not visible to t. Let $\pi(s,t)$ be any shortest s-t path in $\mathcal{P}$. Let u and v be the s-pivot and t-pivot in $\pi(s,t)$, respectively. Thus, $d_{u,v}(s,t) = |su| + d(u,v) + |vt|$. Now we consider $d_{u,v}(s,t)$ as a function of s and t in the entire plane $\mathbb{R}^2$ (not only in $\mathcal{P}$; namely, when we move s and t, they are allowed to move outside $\mathcal{P}$, but the function $d_{u,v}(s,t)$ is always defined as $|su| + d(u,v) + |vt|$, where $d(u,v)$ is a fixed value).

The *π-range* $R_\pi(s,t)$ is defined with respect to t and the path $\pi(s,t)$ as follows: a direction r_s for s is in $R_\pi(s,t)$ if $d'_{u,v}(s,t) < 0$ when we move s along r_s with unit speed.

▶ **Lemma 9.** *$R_\pi(s,t)$ is exactly the open range of size π delimited by $h_s(u)$, where $h_s(u)$ is the open half-plane containing u and bounded by the line through s and perpendicular to $\overline{su}$.*

Now we are back to our original problem to determine $R(s,t)$ for a non-degenerate farthest point t of s with $t \in \mathcal{V}$. Since t is non-degenerate and t is in $\mathcal{V}$, there is only one shortest path $\pi(s,t)$ from s to t. We define $R_\pi(s,t)$ as above. Based on Observation 5 and Lemmas 8(3), we have Lemma 10, and thus $R(s,t)$ can be determined by Observation 5 and Lemma 9.

▶ **Lemma 10.** *$R(s,t) = R_f(s) \cap R_\pi(s,t)$.*

4.2 The Case $t \in E$

The analysis for this case is substantially more complicated than the previous case, although the next case for $t \in \mathcal{I}$ is even more challenging. As in the previous case, we first present a more general result that is on two shortest path distance functions.

Let s and t be any two points in $\mathcal{P}$ such that t is in E and there are two shortest s-t paths $\pi_1(s,t)$ and $\pi_2(s,t)$ (this implies that s is not visible to t). Let e be the polygon edge containing t and let $l(e)$ denote the line containing e. For each $i = 1, 2$, let $\pi_i(s,t) = \pi_{u_i,v_i}(s,t)$, i.e., u_i and v_i are the s-pivot and t-pivot of $\pi_i(s,t)$, respectively. We further require the set $\{v_1, v_2\}$ to satisfy the same condition as $U_t(s)$ in Observation 2(2), i.e., $\{v_1, v_2\}$ has at least one vertex in the open half-plane bounded by $l(e)$ and containing the interior of $\mathcal{P}$ in the small neighborhood of e, and it has at least one vertex in each of the two open half-planes bounded by the line through t and perpendicular to e. We say that the two shortest paths $\pi_1(s,t)$ and $\pi_2(s,t)$ are *canonical* with respect to s and t if $\{v_1, v_2\}$ satisfies the above condition. In the following, we assume $\pi_1(s,t)$ and $\pi_2(s,t)$ are canonical. Note

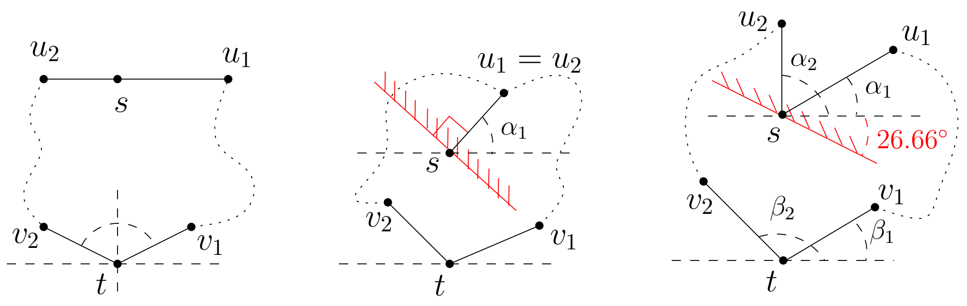


Figure 6 Illustrating some examples for Lemma 11. Left: the special case (the vertical line through t bisects $\angle v_1tv_2$); in this case, $R_\pi(s,t) = \emptyset$. Middle: the case where $\alpha_1 = \alpha_2$, and thus $\alpha = 0$, $\sin(\alpha) = 0$, and $u_1 = u_2$; in this case, $R_\pi(s,t) = (\alpha_1 - \pi/2, \alpha_1 + \pi/2)$, which is delimited by the open half-plane (marked with red color in the figure) bounded by the line through s and perpendicular to $\overline{su_1}$. Right: the most general case where $\alpha_1 = 30°$, $\alpha_2 = 90°$, $\beta_1 = 30°$, $\beta_2 = 135°$; by calculation, $\lambda \approx 1.3165$, $\arctan(\frac{\gamma}{\sin(\alpha)}) \approx 56.66°$, and thus, $R_\pi(s,t) \approx (\alpha_1 - 56.66°, \alpha_1 + 56.66°) = (-26.66°, 153.34°)$; the open half-plane that delimits $R_\pi(s,t)$ is marked with red color in the figure.

that the condition implies that $v_1 \neq v_2$. However, $u_1 = u_2$ is possible. For each $i = 1, 2$, we consider $d_{u_i,v_i}(s,t) = |su_i| + d(u_i, v_i) + |v_it|$ as a function of $s \in \mathbb{R}^2$ and $t \in e$.

In this case, the *π-range* $R_\pi(s,t)$ of s is defined with respect to t and the two paths $\pi_1(s,t)$ and $\pi_2(s,t)$ as follows: a direction r_s for s is in $R_\pi(s,t)$ if there exists a direction r_t parallel to e for t with a speed $\tau \geq 0$ such that when we move s along r_s with unit speed and move t along r_t with speed $\tau \geq 0$, $d'_{u_i,v_i}(s,t) < 0$ holds for $i = 1, 2$.

In Section 4.1, we showed that the π-range for the case $s \in \mathcal{V}$ is an open range of size π. Here we will show a similar result in Lemma 11 unless a special case happens. Although the result in Section 4.1 is quite straightforward, the result here for two functions $d_{u_i,v_i}(s,t)$ with $i = 1, 2$ is somewhat surprising. We first introduce some notation.

For any two points p and q in the plane, define $\overrightarrow{pq}$ as the direction from p to q.

Recall that the angle of any direction r is defined to be the angle in $[0, 2\pi)$ counterclockwise from the positive direction of the x-axis. Let α_1 denote the angle of the direction $\overrightarrow{su_1}$, and let α_2 denote the angle of the direction $\overrightarrow{su_2}$ (e.g., see Fig. 5). Note that by our way of defining pivot vertices, $\alpha_1 = \alpha_2$ if and only if $u_1 = u_2$.

Note that v_1 and v_2 are in a closed half-plane bounded by the line $l(e)$. We assign a direction to $l(e)$ such that each of v_1 and v_2 are to the left or on $l(e)$. Define β_i as the smallest angle to rotate $l(e)$ counterclockwise such that the direction of $l(e)$ becomes the same as $\overrightarrow{tv_i}$, for each $i = 1, 2$ (e.g., see Fig. 5). Hence, both β_1 and β_2 are in $[0, \pi]$. Without of loss of generality, we assume $\beta_1 \leq \beta_2$ (otherwise the analysis is symmetric). Since $\{v_1, v_2\}$ contains at least one vertex in each of the open half-planes bounded by the line through t and perpendicular to e, we have $\beta_1 \in [0, \pi/2)$ and $\beta_2 \in (\pi/2, \pi]$. Further, since at least one of v_1 and v_2 is not on $l(e)$, it is not possible that both $\beta = 0$ and $\beta = \pi$ hold.

Let $\alpha = \alpha_2 - \alpha_1$. We refer to the case where $\beta_1 + \beta_2 = \pi$ and $\alpha = \pm\pi$ (i.e., α is π or $-\pi$) as the *special case*. In the special case, s is on $\overline{u_1u_2}$ and the vertical line through t and perpendicular to $l(e)$ bisects the angle $\angle v_1tv_2$.

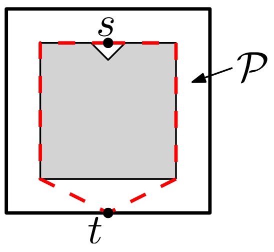


Figure 7 Illustrating an example in which a geodesic center s is in $\mathcal{I}$ and has only one farthest point t. The polygonal domain $\mathcal{P}$ is between two (very close) concentric squares plus an additional (very small) triangle so that s is in $\mathcal{I}$. The point s is at the middle of the top edge of the inner square, and t is at the middle of the bottom edge of the outer square. One can verify that s is a geodesic center and t is the only farthest point of s. The two shortest paths from s to t are shown with red dashed segments. Note that the middle point of every edge of the inner square is a geodesic center.

▶ **Lemma 11.** *The π-range $R_\pi(s,t)$ is determined as follows (e.g., see Fig. 6).*

$$R_\pi(s,t) = \begin{cases} (\alpha_1 - \arctan(\frac{\lambda}{\sin(\alpha)}), \alpha_1 - \arctan(\frac{\lambda}{\sin(\alpha)}) + \pi) & \textit{if } \sin(\alpha) > 0, \\ (\alpha_1 - \arctan(\frac{\lambda}{\sin(\alpha)}) - \pi, \alpha_1 - \arctan(\frac{\lambda}{\sin(\alpha)})) & \textit{if } \sin(\alpha) < 0, \\ (\alpha_1 - \pi/2, \alpha_1 + \pi/2) & \textit{if } \sin(\alpha) = 0 \textit{ and } \lambda > 0, \\ (\alpha_1 - 3\pi/2, \alpha_1 - \pi/2) & \textit{if } \sin(\alpha) = 0 \textit{ and } \lambda < 0, \\ \emptyset & \textit{if } \sin(\alpha) = 0 \textit{ and } \lambda = 0, \end{cases}$$

where $\lambda = \cos\alpha - \frac{\cos\beta_2}{\cos\beta_1}$. Further, $\alpha = \pm\pi$ and $\beta_1 + \beta_2 = \pi$ (i.e., the special case) if and only if $\sin(\alpha) = 0$ and $\lambda = 0$.

By Lemma 11, if the special case happens, $R_\pi(s,t) = \emptyset$; otherwise, it is an open range of size π. Since $\alpha = 0$ if and only if $u_1 = u_2$, the case $u_1 = u_2$ is also covered by the lemma.

Now consider our original problem of determining the range $R(s,t)$ for a non-degenerate farthest point $t \in E$ of s. By Observation 5, s is not visible to t. Further, s and t have exactly two shortest paths $\pi_1(s,t)$ and $\pi_2(s,t)$. Clearly, by Observation 2(2), the two paths are canonical. Therefore, the π-range $R_\pi(s,t)$ of s with respect to t and the two shortest paths $\pi_1(s,t)$ and $\pi_2(s,t)$ can be determined by Lemma 11. Lemma 8(2) leads to Lemma 12.

▶ **Lemma 12.** *$R(s,t) = R_\pi(s,t) \cap R_f(s)$.*

Suppose t is the only farthest point of s and t is non-degenerate. By Lemma 11, if the special case happens, $R_\pi(s,t) = \emptyset$ and $R(s,t) = \emptyset$. By Corollary 7, if we move s along any free direction infinitesimally, $d_{\max}(s)$ will be strictly increasing. Therefore, it is possible that the point s, which is in $\mathcal{I}$ and has only one farthest point, is a geodesic center. It is not difficult to construct such an example by following the left figure of Fig. 6; e.g., see Fig. 7. Hence, we have the following corollary.

▶ **Corollary 13.** *It is possible that a geodesic center is in $\mathcal{I}$ and has only one farthest point.*

4.3 The Case $t \in \mathcal{I}$

The analysis for this case is substantially more difficult than the case $t \in E$. As before, we first present a more general result that is on three shortest path distance functions.

Let s and t be two points in $\mathcal{P}$ such that t is in $\mathcal{I}$ and there are three shortest s-t paths $\pi_1(s,t)$, $\pi_2(s,t)$, and $\pi_3(s,t)$ (this implies that s is not visible to t). For each $i = 1,2,3$, let $\pi_i(s,t) = \pi_{u_i,v_i}(s,t)$, i.e., u_i and v_i are the s-pivot and t-pivot of $\pi_i(s,t)$, respectively. We say that the three paths are *canonical* with respect to s and t if they have two properties:

1. t is in the interior of the triangle $\triangle v_1v_2v_3$.
2. Suppose we reorder the indices such that v_1, v_2, and v_3 are *clockwise* around t, then u_1, u_2, and u_3 are *counterclockwise* around s (e.g., see Fig. 2).

The above first property implies that v_1, v_2, and v_3 are distinct, but this may not be true for u_1, u_2, u_3. In the following, we assume that the three shortest paths $\pi_i(s,t)$ with $1 \leq i \leq 3$ are canonical, and we reorder the indices as in the above second property. For each $i = 1,2,3$, we consider $d_{u_i,v_i}(s,t) = |su_i| + d(u_i,v_i) + |v_it|$ as a function of $s \in \mathbb{R}^2$ and $t \in \mathbb{R}^2$. In this case, the *π-range* $R_\pi(s,t)$ of s is defined with respect to t and the three paths $\pi_i(s,t)$ for $i = 1,2,3$ as follows: a direction r_s for s is in $R_\pi(s,t)$ if there exists a direction r_t for t with a speed $\tau \geq 0$ such that when we move s along r_s with unit speed and move t along r_t with speed τ, $d'_{u_i,v_i} < 0$ holds for $i = 1,2,3$.

As Lemma 11 in the previous cases, we will have a similar lemma (Lemma 14), which says that unless a special case happens $R_\pi(s,t)$ is an open range of size exactly π. The proof is much more challenging. Before presenting Lemma 14, we introduce some notation.

For each $i = 1,2,3$, let β_i denote the angle of the direction $\overrightarrow{tv_i}$ (i.e., the angle of $\overrightarrow{tv_i}$ counterclockwise from the positive x-axis). Further, we define three angles b_i for $i = 1,2,3$ as follows (e.g., see Fig. 2). Define b_1 as the smallest angle we need to rotate the direction $\overrightarrow{tv_1}$ *clockwise* to $\overrightarrow{tv_2}$; define b_2 as the smallest angle we need to rotate the direction $\overrightarrow{tv_2}$ clockwise to $\overrightarrow{tv_3}$; define b_3 as the smallest angle we need to rotate the direction $\overrightarrow{tv_3}$ clockwise to $\overrightarrow{tv_1}$.

For any two angles α' and α'', we use $\alpha' \equiv \alpha''$ to denote $\alpha' = \alpha'' \mod 2\pi$.

It is easy to see that $b_1 \equiv \beta_1 - \beta_2$, $b_2 \equiv \beta_2 - \beta_3$, and $b_3 \equiv \beta_3 - \beta_1$. Note that since t is in the interior of $\triangle v_1v_2v_3$, it holds that $b_i \in (0,\pi)$ for $i = 1,2,3$. Note that $b_1 + b_2 + b_3 = 2\pi$.

For each $i = 1,2,3$, let α_i denote the angle of the direction $\overrightarrow{su_i}$. According to our definition of pivot vertices, $u_i = u_j$ if and only if $\alpha_i = \alpha_j$ for any two $i,j \in \{1,2,3\}$. We define three angles a_i for $i = 1,2,3$ as follows (e.g., see Fig. 2). Define a_1 as the smallest angle we need to rotate the direction $\overrightarrow{su_1}$ *counterclockwise* to $\overrightarrow{su_2}$; define a_2 as the smallest angle we need to rotate the direction $\overrightarrow{su_2}$ clockwise to $\overrightarrow{su_3}$; define a_3 as the smallest angle we need to rotate the direction $\overrightarrow{su_3}$ clockwise to $\overrightarrow{su_1}$. Hence, $a_1 \equiv \alpha_2 - \alpha_1$, $a_2 \equiv \alpha_3 - \alpha_2$, and $a_3 \equiv \alpha_1 - \alpha_3$.

We refer to the case where $a_i = b_i$ for each $i = 1,2,3$ as the *special case.*

▶ **Lemma 14.** *The π-range $R_\pi(s,t)$ is determined as follows (e.g., see Fig. 8).*

$$R_\pi(s,t) = \begin{cases} (\alpha_1 - \arctan(\frac{\delta_1-\delta_2}{\delta}), \alpha_1 - \arctan(\frac{\delta_1-\delta_2}{\delta}) + \pi) & \text{if } \delta > 0, \\ (\alpha_1 - \arctan(\frac{\delta_1-\delta_2}{\delta}) - \pi, \alpha_1 - \arctan(\frac{\delta_1-\delta_2}{\delta})) & \text{if } \delta < 0, \\ (\alpha_1 - \pi/2, \alpha_1 + \pi/2) & \text{if } \delta = 0 \text{ and } \delta_1 > \delta_2, \\ (\alpha_1 - 3\pi/2, \alpha_1 - \pi/2) & \text{if } \delta = 0 \text{ and } \delta_1 < \delta_2, \\ \emptyset & \text{if } \delta = 0 \text{ and } \delta_1 = \delta_2, \end{cases}$$

where $\delta = \frac{\sin(\alpha_3-\alpha_1)}{\sin(\beta_3-\beta_1)} - \frac{\sin(\alpha_2-\alpha_1)}{\sin(\beta_2-\beta_1)}$, $\delta_1 = \frac{\cos(\beta_2-\beta_1)-\cos(\alpha_2-\alpha_1)}{\sin(\beta_2-\beta_1)}$, *and* $\delta_2 = \frac{\cos(\beta_3-\beta_1)-\cos(\alpha_3-\alpha_1)}{\sin(\beta_3-\beta_1)}$. *Further, $a_i = b_i$ for each $i = 1,2,3$ (i.e., the special case) if and only if $\delta = 0$ and $\delta_1 = \delta_2$.*

According to Lemma 14, if the special case happens, then $R_\pi(s,t)$ is empty; otherwise, it is an open range of size exactly π.

Now we are back to our original problem to determine the range $R(s,t)$ for a non-degenerate farthest point $t \in \mathcal{I}$ of s. Since there are exactly three shortest s-t paths $\pi_i(s,t)$

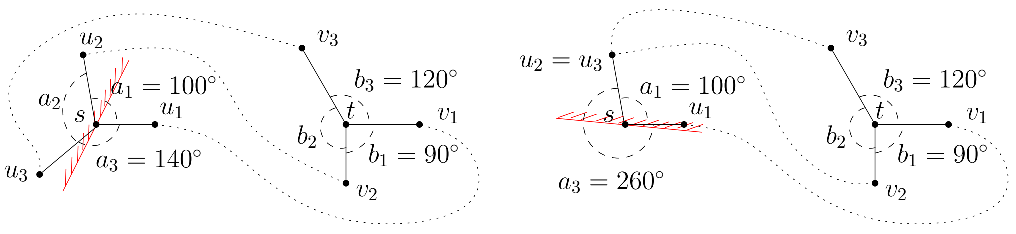


Figure 8 Illustrating two examples for Lemma 14. Left: The sizes of the angles of a_i and b_i for $1 \leq i \leq 3$ are already shown in the figure with $\alpha_1 = 0$. By calculation, $\delta \approx 0.2426$, $\delta_1 \approx -0.1736$, $\delta_2 \approx 0.3072$, $\arctan(\frac{\delta_1-\delta_2}{\delta}) \approx -63.23°$, and thus $R_\pi(s,t) \approx (\alpha_1 + 63.23°, \alpha_1 + 63.23° + 180°) = (63.23°, 243.23°)$. Right: a case where $\alpha_2 = \alpha_3$ with $\alpha_1 = 0$. Thus, $a_2 = 0$ and $u_2 = u_3$. The sizes of other angles are already shown in the figure. By calculation, $\delta \approx 2.1220$, $\delta_1 \approx -0.1736$, $\delta_2 \approx -0.3768$, $\arctan(\frac{\delta_1-\delta_2}{\delta}) \approx 5.47°$, and thus $R_\pi(s,t) \approx (\alpha_1 - 5.47°, \alpha_1 - 5.47° + 180°) = (-5.47°, 174.53°)$. The open half-planes that delimit $R_\pi(s,t)$ in both examples are marked with red color.

for $i = 1,2,3$, the three paths must be canonical. To see this, by Observation 2, t is in the interior of $\triangle v_1v_2v_3$. Further, it is easy to see that no two of the three paths cross each other since otherwise there would be more than three shortest s-t paths, this implies that the second property of the canonical paths holds. Let $R_\pi(s,t)$ be the π-range of s with respect to t and the above three shortest paths. By Lemma 8(1), we have the following lemma.

▶ **Lemma 15.** *$R(s,t) = R_\pi(s,t) \cap R_f(s)$.*

5 Computing the Candidate Points

In this section, with the help of the observations in Sections 3 and 4, we compute a set S of candidate points such that all geodesic centers must be in S.

Let s be any geodesic center. Recall that $F(s)$ is the set of all farthest points of s. Depending on whether s is in $\mathcal{V}$, E, or $\mathcal{I}$, the size $|F(s)|$, whether some points of $F(s)$ are in $\mathcal{V}$, E, or $\mathcal{I}$, whether s has a degenerate farthest point, there are a significant (but still constant) number of cases. For each case, we use an exhaustive-search approach to compute a set of candidate points such that s must be in the set. In particular, there are four cases, called *dominating cases*, for which the number of candidate points is $O(n^{11})$. But the total number of the candidate points for all other cases is only $O(n^{10})$. Therefore, the set S has a total of $O(n^{11})$ candidate points. We will show that S can be computed in $O(n^{11} \log n)$ time.

To find the geodesic centers in S, a straightforward algorithm works as follows. For each point $\hat{s} \in S$, we can compute $d_{\max}(\hat{s})$ in $O(n \log n)$ time by first computing the shortest path map $SPM(\hat{s})$ of $\hat{s}$ in $O(n \log n)$ time [12] and then obtaining the maximum geodesic distance from $\hat{s}$ to all vertices of $SPM(\hat{s})$. Since all geodesic centers are in S, the points of S with the smallest $d_{\max}(\hat{s})$ are geodesic centers of $\mathcal{P}$.

Since $|S| = O(n^{11})$, the above algorithm runs in $O(n^{12} \log n)$ time. Let S_d denote the set of the candidate points for the four dominating cases. Clearly, the bottleneck is on finding the geodesic centers from S_d. To improve the algorithm, when we compute the candidate points of S_d, we will maintain the corresponding *path information*. By using these path information and based on new observations, we will present in Section 6 an $O(n^{11} \log n)$ time "pruning algorithm" that can eliminate most of the points from S_d such that none of the eliminated points is a geodesic center and the number of remaining points in S_d is only $O(n^{10})$. Consequently, we can find all geodesic centers in additional $O(n^{11} \log n)$ time.

The dominating cases. In order to discuss our pruning algorithm in Section 6, we explain the four dominating cases as follows. Suppose s is a geodesic center in $\mathcal{I}$ that has the following three properties. (1) s does not have a degenerate farthest point. (2) For any farthest point t of s, the π-range $R_\pi(s,t) \neq \emptyset$ (i.e., the special cases in Lemmas 11 and 14 do not happen). (3) s has at least three farthest points. By Lemma 3, these properties further imply that s must have three farthest points t_1, t_2, t_3 such that $R_\pi(s,t_1) \cap R_\pi(s,t_2) \cap R_\pi(s,t_3) = \emptyset$ since the size of $R_\pi(s,t_i)$ is π for each t_i. Depending on whether each t_i for $i = 1,2,3$ is in $\mathcal{I}$, E, $\mathcal{V}$, there are several cases. We use (x,y,z) to refer to the case where x, y, and z points of t_1,t_2,t_3 are in $\mathcal{I}, E$, and $\mathcal{V}$, respectively, with $x+y+z=3$. For example, $(3,0,0)$ refers to the case where t_1,t_2,t_3 are all in $\mathcal{I}$. The four dominating cases are: $(3,0,0)$, $(2,1,0)$, $(1,2,0)$, and $(0,3,0)$. For any geodesic center s, if s does not belong to the dominating cases, then our algorithm guarantees that s is in $S \setminus S_d$. Below, due to the space limit, we only sketch how to compute candidate points for three "representative" cases, and other details are omitted.

Case 1. Consider the dominating case $(3,0,0)$ discussed above. We compute the candidate points for the geodesic center s as follows. For each $i=1,2,3$, since t_i is in $\mathcal{I}$, there are three shortest paths from s to t_i: $\pi_{u_{ij},v_{ij}}(s,t)$ with $j=1,2,3$ (i.e., u_{ij} and v_{ij} are the s-pivot and t-pivot, respectively). Hence, we have the following

$$\begin{aligned}
&|t_1v_{11}| + d(v_{11},u_{11}) + |u_{11}s| = |t_1v_{12}| + d(v_{12},u_{12}) + |u_{12}s| = |t_1v_{13}| + d(v_{13},u_{13}) + |u_{13}s|\\
=&|t_2v_{21}| + d(v_{21},u_{21}) + |u_{21}s| = |t_2v_{22}| + d(v_{22},u_{22}) + |u_{22}s| = |t_2v_{23}| + d(v_{23},u_{23}) + |u_{23}s|\\
=&|t_3v_{31}| + d(v_{31},u_{31}) + |u_{31}s| = |t_3v_{32}| + d(v_{32},u_{32}) + |u_{32}s| = |t_3v_{33}| + d(v_{33},u_{33}) + |u_{33}s|.
\end{aligned}$$

By considering the coordinates of s,t_1,t_2, and t_3 as eight variables, the above equations on the lengths of the nine shortest paths provide eight (independent) constraints, which are sufficient to compute all four points. Correspondingly, we compute the candidate points for s by the exhaustive-search algorithm below (similar methods were also used before, e.g., [3, 7]).

We enumerate all possible combinations of nine polygon vertices as v_{i1},v_{i2},v_{i3}, with $i=1,2,3$. For each combination, we compute the overlay of the shortest path maps of the nine vertices. The overlay is of size $O(n^2)$ and can be computed in $O(n^2\log n)$ time [3, 7]. For each cell C of the overlay, we obtain nine roots of the shortest path maps and consider them as u_{i1},u_{i2},u_{i3} for $i=1,2,3$. We form the above system of eight equations and solve it to obtain a constant number of quadruples $(\hat{s},\hat{t}_1,\hat{t}_2,\hat{t}_3)$ and each such $\hat{s}$ is a candidate point. In this way, for each combination of nine polygon vertices, we can obtain $O(n^2)$ candidate points in $O(n^2\log n)$ time. Since there are $O(n^9)$ combinations, we can compute $O(n^{11})$ candidate points in $O(n^{11}\log n)$ time and s must be one of these candidate points.

If this were not a dominating case, we would have done for this case. Since this is a dominating case, we need to maintain certain path information. To this end, we perform a "validation procedure" on each such quadruple $(\hat{s},\hat{t}_1,\hat{t}_2,\hat{t}_3)$ computed above, as follows.

In the above procedure for computing $(\hat{s},\hat{t}_1,\hat{t}_2,\hat{t}_3)$, we also obtain a path length, denoted by $d(\hat{s})$, which is equal to the value in the above equations, e.g., $d(\hat{s}) = |\hat{s}u_{11}|+d(u_{11},v_{11})+|v_{11}\hat{t}_1|$. First, we check whether $\hat{s}$ is in C, which can be done in $O(\log n)$ time by using a point location data structure [9, 14] with $O(n^2)$ time and space preprocessing on the overlay. If yes, for each t_i with $i=1,2,3$, we check whether $d(\hat{s})$ is equal to $d(\hat{s},\hat{t}_i)$, which can be computed in $O(\log n)$ time by using the two-point shortest path query data structure [7] with $O(n^{11})$ time and space preprocessing on $\mathcal{P}$. If yes, we check whether v_{i1},v_{i2},v_{i3} satisfy the condition in Observation 2(1), i.e., whether $\hat{t}_i$ is in the interior of the triangle $\triangle v_{i1}v_{i2}v_{i3}$ for each $i=1,2,3$. If yes, for each $\hat{t}_i$ with $i=1,2,3$, we check whether the order of the vertices of v_{i1},v_{i2},v_{i3} around $\hat{t}_i$

are consistent with the order of the vertices of u_{i1}, u_{i2}, u_{i3} (we say that the two orders are *consistent* if after reordering the indices, v_{i1}, v_{i2}, v_{i3} are clockwise around $\hat{t}_i$ while u_{i1}, u_{i2}, u_{i3} are counterclockwise around s; note that this consistency is needed for determining the π-range in Lemma 14). If yes, for each $\hat{t}_i$ with $i = 1, 2, 3$, we compute the π-range $R_\pi(s, \hat{t}_i)$ determined by Lemma 14, and then check whether $R_\pi(\hat{s}, \hat{t}_1) \cap R_\pi(\hat{s}, \hat{t}_2) \cap R_\pi(\hat{s}, \hat{t}_3)$ is empty. If yes, we say that the quadruple $(\hat{s}, \hat{t}_1, \hat{t}_2, \hat{t}_3)$ *passes* the validation procedure and we call $\hat{s}$ a *valid* candidate point and add $\hat{s}$ to the set S_d. In addition, we maintain the following *path information*: $d(\hat{s})$, $\hat{t}_i$, v_{ij}, and u_{ij}, with $1 \leq i \leq 3$ and $1 \leq j \leq 3$.

In the worst case, we can have $O(n^{11})$ valid candidate points for S_d. Note that the geodesic center s and the quadruple (s, t_1, t_2, t_3) discussed above must pass the validation procedure, and thus the quadruple (s, t_1, t_2, t_3) will be computed by our exhaustive-search algorithm and s will be computed as a valid candidate point in S_d.

Based on our validation procedure, the following observation summarizes the properties of the valid candidate points, which will be useful for our pruning algorithm in Section 6.

▶ **Observation 16.** Suppose $(\hat{s}, \hat{t}_1, \hat{t}_2, \hat{t}_3)$ a quadruple that passes the validation procedure, with u_{ij} and v_{ij}, $i = 1, 2, 3$ and $j = 1, 2, 3$ defined as above. Then the following hold.

1. For each $i = 1, 2, 3$, $\overline{\hat{s}u_{ij}} \cup \pi(u_{ij}, v_{ij}) \cup \overline{v_{ij}\hat{t}_i}$ is a shortest path from $\hat{s}$ to $\hat{t}_i$ for each $j = 1, 2, 3$.
2. For each $i = 1, 2, 3$, v_{i1}, v_{i2}, v_{i3} satisfy the condition of Observation 2(1), i.e., $\hat{t}_i$ is in the interior of the triangle $\triangle v_{i1}v_{i2}v_{i3}$.
3. $d(\hat{s}) = d(\hat{s}, \hat{t}_i)$ for each $i = 1, 2, 3$.
4. $R_\pi(\hat{s}, \hat{t}_1) \cap R_\pi(\hat{s}, \hat{t}_2) \cap R_\pi(\hat{s}, \hat{t}_1) = \emptyset$.

Case 2. Consider the case where s is a geodesic center that has only one farthest point t such that t is non-degenerate with respect to s, with both s and t in $\mathcal{I}$. We compute the candidate points for s as follows. We will need the π-range property in this case.

Since t is non-degenerate, there are exactly three shortest s-t paths: $\pi_{u_i,v_i}(s,t)$ with $i = 1, 2, 3$. We have $|su_1| + d(u_1, v_1) + |v_1t| = |su_2| + d(u_2, v_2) + |v_2t| = |su_3| + d(u_3, v_3) + |v_3t|$. If we consider the coordinates of s and t as four variables, the equations give two constraints, and to determine s and t, we need two more constraints, which are provided by the π-range property (this kind of situation does not appear in the previous work [3, 7] and thus they do not need the π-range property). Indeed, since t is the only farthest point of s, we have $R(s) = R(s, t)$. By Lemma 3, $R(s) = \emptyset$. Hence, $R(s, t) = \emptyset$. Since $s \in \mathcal{I}$, $R_\pi(s, t) = R(s, t) = \emptyset$. Since $t \in \mathcal{I}$, $R_\pi(s, t)$ is determined by Lemma 14. We define the angles a_i, b_i, for $i = 1, 2, 3$, in the same way as those for Lemma 14 in Section 4.3. By Lemma 14, $R_\pi(s, t) = \emptyset$ if and only if $a_i = b_i$ for $i = 1, 2, 3$. The identities of the three pairs of angles provide another two (independent) constraints. Using the above four constraints, we can determine s and t. Correspondingly, the candidate points for s can be computed in an exhaustive manner, as follows.

We enumerate all possible combinations of three polygon vertices as v_1, v_2, v_3. We compute the shortest path maps of v_1, v_2, and v_3 in $O(n \log n)$ time. Next we compute the overlay of the three shortest path maps. Then, for each cell of the overlay, we obtain the three roots of the cell in the three shortest path maps and consider them as u_1, u_2, u_2. Finally, we use the above four constraints to determine a constant number of pairs $(\hat{s}, t)$ (we assume this can be done in constant time since the angles a_i and b_i can be parameterized by the coordinates of $\hat{s}$ and t), and each such $\hat{s}$ is considered as a candidate point. In this way, for each combination of v_1, v_2, v_3, we can compute $O(n^2)$ candidate points in $O(n^2 \log n)$ time. Since there are $O(n^3)$ combinations, we can compute $O(n^5)$ candidate points in $O(n^5 \log n)$ time.

Case 3. Consider the case where s is a geodesic center that has only two farthest point t_1 and t_2 such that both t_1 and t_2 are non-degenerate, with s, t_1, t_2 all in $\mathcal{I}$.

For each t_i, $i = 1, 2$, there are exactly three shortest paths from s. The equations on the lengths of the six paths give five constraints for s, t_1, t_2 if we consider their coordinates as six variables. To determine s, t_1, t_2, we need one more constraint, which is provided by the π-range property as follows. Since s, t_1, t_2 are all in $\mathcal{I}$, for each $i = 1, 2$, $R(s, t_i) = R_\pi(s, t_i)$, and $R_\pi(s, t_i)$ is determined by Lemma 14. We assume neither $R_\pi(s, t_1)$ nor $R_\pi(s, t_2)$ is empty since otherwise the candidate points could be computed by the algorithm for the above Case 2. Hence, each $R_\pi(s, t_i)$, $i = 1, 2$, is an open range of size π. By Lemma 3, $R(s) = R(s, t_1) \cap R(s, t_2) = R_\pi(s, t_1) \cap R_\pi(s, t_2) = \emptyset$. Since each $R_\pi(s, t_i)$, $i = 1, 2$, is an open range of size π (i.e., it is delimited by an open half-plane whose bounding line contains s), to have $R_\pi(s, t_1) \cap R_\pi(s, t_2) = \emptyset$, the two bounding lines of the two half-planes delimiting the two π-ranges must be overlapped, and this provides the sixth constraint to determine s, t_1, t_2. Correspondingly, the candidate points for s can be computed in an exhaustive way.

6 Computing the Geodesic Centers

In this section, we find all geodesic centers from the candidate point set S. Recall that S_d is the set of candidate points for the four dominating cases. Let S' denote the set of candidate points for all other cases, and thus $S = S_d \cup S'$. As discussed in Section 5, $|S'| = O(n^{10})$ and we can find all geodesic centers in S' in $O(n^{11} \log n)$ time by computing shortest path maps. Below, we focus on finding all geodesic centers in S_d.

We first remove all points from S_d that are also in S', which can be done in $O(n^{11} \log n)$ time (e.g., by first sorting these points by their coordinates). Then, according to our definitions of the four dominating cases in Section 5, for any point $s \in S_d$, if s is a geodesic center, s does not have any degenerate farthest point since otherwise s was also in S' and thus would have already been removed from S_d. Recall that each point s of S_d is a valid candidate point and we have maintained its path information (in particular, the value $d(s)$).

We first perform the following *duplication-cleanup* procedure: for each point $s \in S_d$, if there are many copies of s, we only keep the one with the largest value $d(s)$ (if more than one copy has the largest value, we keep an arbitrary one). This procedure can be done in $O(n^{11} \log n)$ time (e.g., by first sorting all points of S_d by their coordinates). According to our algorithm for computing the candidate points of S_d, we have the following observation.

▶ **Lemma 17.** *After the duplication-cleanup procedure, for any point $s \in S_d$, if s is a geodesic center, then $d_{\max}(s) = d(s)$.*

Recall that all points of S_d are in $\mathcal{I}$. In the following, we give a *pruning algorithm* that can eliminate most of the points from S_d such that none of these eliminated points is a geodesic center and the number of remaining points of S_d is $O(n^{10})$. Our pruning algorithm relies on the property that each candidate point s of S_d is valid. Specifically, if s is computed for the dominating case $(3, 0, 0)$, then s is associated with the following path information $d(s)$, t_i, v_{ij}, and u_{ij} for $1 \leq i \leq 3$ and $1 \leq j \leq 3$, such that Observation 16 holds (i.e., s is $\hat{s}$ and each t_i is $\hat{t}_i$). For other three dominating cases (e.g., $(2, 1, 0)$, $(1, 2, 0)$, and $(0, 3, 0)$), there are similar properties. By using these properties, we have the following key lemmas.

▶ **Lemma 18.** *Let s be any point in S_d. If s is in the interior of a cell or an edge of $\mathcal{D}_{spm}$, then for any other point s' in the interior of the same cell or edge of $\mathcal{D}_{spm}$, it holds that $d_{\max}(s') > d(s)$.*

▶ **Lemma 19.** *For any two points s_1 and s_2 of S_d that are in the interior of the same cell or the same edge of $\mathcal{D}_{spm}$, if $d(s_1) < d(s_2)$, then s_1 cannot be a geodesic center, and if $d(s_1) = d(s_2)$, then neither s_1 nor s_2 is a geodesic center.*

Based on Lemma 19, our pruning algorithm for S_d works as follows. For each point s of S_d, we determine the cell, edge, or vertex of $\mathcal{D}_{spm}$ that contains s in its interior, which can be done in $O(\log n)$ time by using a point location data structure [9, 14] with $O(n^{10})$ time and space preprocessing on $\mathcal{D}_{spm}$. For each edge or cell, let S'_d be the set of points of S_d that are contained in its interior. We find the point s of S'_d with the largest value $d(s)$. If there are more than one such point in S'_d, we remove all points of S'_d from S_d; otherwise, remove all points of S'_d except s from S_d. By Lemma 19, none of the points of S_d that are removed above is a geodesic center. After the above pruning algorithm, S_d contains at most one point in the interior of each cell, edge, or vertex of $\mathcal{D}_{spm}$. Hence, $|S_d| = O(|\mathcal{D}_{spm}|)$. Since $|\mathcal{D}_{spm}| = O(n^{10})$ [7], we obtain $|S_d| = O(n^{10})$. Consequently, we can find all geodesic centers in S_d in $O(n^{11} \log n)$ time by computing shortest path maps.

We thus conclude that all geodesic centers of $\mathcal{P}$ can be computed in $O(n^{11} \log n)$ time.

▶ **Theorem 20.** *All geodesic centers of $\mathcal{P}$ can be computed in $O(n^{11} \log n)$ time.*

Acknowledgment. We wish to thank Yan Sun for the discussions on proving the π-range property.

References

1 H.-K. Ahn, L. Barba, P. Bose, J.-L. De Carufel, M. Korman, and E. Oh. A linear-time algorithm for the geodesic center of a simple polygon. In *Proc. of the 31st Annual Symposium on Computational Geometry (SoCG)*, pages 209–223, 2015.

2 T. Asano and G. Toussaint. Computing the geodesic center of a simple polygon. Technical Report SOCS-85.32, McGill University, Montreal, Canada, 1985.

3 S.W. Bae, M. Korman, and Y. Okamoto. The geodesic diameter of polygonal domains. *Discrete and Computational Geometry*, 50:306–329, 2013.

4 S.W. Bae, M. Korman, and Y. Okamoto. Computing the geodesic centers of a polygonal domain. In *Proc. of the 26th Canadian Conference on Computational Geometry (CCCG)*, 2014.

5 S.W. Bae, M. Korman, Y. Okamoto, and H. Wang. Computing the L_1 geodesic diameter and center of a simple polygon in linear time. *Computational Geometry: Theory and Applications*, 48:495–505, 2015.

6 B. Chazelle. A theorem on polygon cutting with applications. In *Proc. of the 23rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 339–349, 1982.

7 Y.-J. Chiang and J.S.B. Mitchell. Two-point Euclidean shortest path queries in the plane. In *Proc. of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 215–224, 1999.

8 H.N. Djidjev, A. Lingas, and J.-R. Sack. An $O(n \log n)$ algorithm for computing the link center of a simple polygon. *Discrete and Computational Geometry*, 8:131–152, 1992.

9 H. Edelsbrunner, L. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM Journal on Computing*, 15(2):317–340, 1986.

10 D. Halperin and M. Sharir. New bounds for lower envelopes in three dimensions, with applications to visibility in terrains. *Discrete and Computational Geometry*, 12:313–326, 1994.

11 J. Hershberger and S. Suri. Matrix searching with the shortest-path metric. *SIAM Journal on Computing*, 26(6):1612–1634, 1997.

12 J. Hershberger and S. Suri. An optimal algorithm for Euclidean shortest paths in the plane. *SIAM Journal on Computing*, 28(6):2215–2256, 1999.

13 Y. Ke. An efficient algorithm for link-distance problems. In *Proc. of the 5th Annual Symposium on Computational Geometry (SoCG)*, pages 69–78, 1989.

14 D. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, 12(1):28–35, 1983.

15 J.S.B. Mitchell. Geometric shortest paths and network optimization, in *Handbook of Computational Geometry,* J.-R Sack and J. Urrutia (eds.), pages 633–702. Elsevier, Amsterdam, the Netherlands, 2000.

16 B.J. Nilsson and S. Schuierer. Computing the rectilinear link diameter of a polygon. In *Proc. of the International Workshop on Computational Geometry - Methods, Algorithms and Applications*, pages 203–215, 1991.

17 B.J. Nilsson and S. Schuierer. An optimal algorithm for the rectilinear link center of a rectilinear polygon. *Computational Geometry: Theory and Applications*, 6:169–194, 1996.

18 R. Pollack, M. Sharir, and G. Rote. Computing the geodesic center of a simple polygon. *Discrete and Computational Geometry*, 4(1):611–626, 1989.

19 S. Schuierer. An optimal data structure for shortest rectilinear path queries in a simple rectilinear polygon. *International Journal of Compututational Geometry and Applications*, 6:205–226, 1996.

20 S. Suri. Computing geodesic furthest neighbors in simple polygons. *Journal of Computer and System Sciences*, 39:220–235, 1989.

The Complexity of the `k-means` Method[*]

Tim Roughgarden[1] and Joshua R. Wang[2]

1 **Department of Computer Science, Stanford University, Stanford, CA, USA**
`tim@cs.stanford.edu`

2 **Department of Computer Science, Stanford University, Stanford, CA, USA**
`joshua.wang@cs.stanford.edu`

Abstract

The `k-means` method is a widely used technique for clustering points in Euclidean space. While it is extremely fast in practice, its worst-case running time is exponential in the number of data points. We prove that the `k-means` method can implicitly solve PSPACE-complete problems, providing a complexity-theoretic explanation for its worst-case running time. Our result parallels recent work on the complexity of the simplex method for linear programming.

1998 ACM Subject Classification I.5.3 Clustering

Keywords and phrases k-means, PSPACE-complete

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.78

1 Introduction

The `k-means` method, also known as Lloyd's algorithm [15], is a widely used technique for clustering points in Euclidean space. It can be viewed as a local search algorithm for the problem of, given n data points in $\mathbb{R}^d$, choosing k centers in $\mathbb{R}^d$ to minimize the sum (or average) of squared Euclidean distances between each point and its closest center.[1] The method begins with an arbitrary set of k initial centers. Each point is then reassigned to the center closest to it, and each center is recomputed as the center of mass of its assigned points. Every iteration decreases the objective function value of the clustering, and these two steps are repeated until the algorithm stabilizes.

Three basic facts about the `k-means` method are:

1. It is extremely fast in practice, and for this reason is widely used, perhaps more than any other clustering algorithm. For example, Berkhin [6] states that it "is by far the most popular clustering algorithm used nowadays in scientific and industrial applications."
2. The worst-case running time of the method is exponential in the number of points. This was first proved by Arthur and Vassilvitskii [4], and extended to the plane by Vattani [18].
3. It has polynomial smoothed complexity in the sense of Spielman and Teng [17]: for every choice of data points, in expectation over Gaussian perturbations with standard deviation σ of these points, the running time of the method is polynomial in the input size and in $1/\sigma$ [3].[2]

* This work was supported in part by NSF grant CCF-1524062 and a Stanford Graduate Fellowship.

1 This problem is *NP*-hard, even in the plane [16].

2 We focus on properties that concern the running time of the `k-means` method. Like with any local search algorithm, one can also consider the approximation quality of the solution output by the method; see the well-known `k-means++` method [5] for an initialization technique with a provable guarantee, and [8, 7] for matching lower bounds on this particular method. Constant-factor guarantees are also known for different local search algorithms [13].

These three properties of the `k-means` method illustrate a clear parallel with the simplex method for linear programming. The simplex method is famously fast in practice, but Klee and Minty [14] showed that it has exponential worst-case running time. These lower bounds have since been extended to many different pivot rules (see e.g. Amenta and Ziegler [2]), and also to restricted classes of linear programs, such as minimum-cost flow [19]. On the other hand, both the average-case and smoothed running times of the simplex method are polynomial (see Spielman and Teng [17] and the references therein).

Disser and Skutella [9] initiated a fresh take on the worst-case exponential running time of the simplex method, by showing that it inadvertently solves problems that are much harder than linear programming. Specifically, they showed how to efficiently embed an instance of the (NP-complete) Partition problem into a linear program so that the trajectory of the simplex method immediately reveals the answer to the instance. In this sense, the simplex method can solve NP-hard problems, thereby providing an explanation of sorts for its worst-case running time. A similar line was taken by Adler et al. [1], who exhibited a pivot rule with which the simplex method can solve PSPACE-complete problems, and Fearnley and Savani [10], who proved analogous results with Dantzig's original pivot rule. These results echo earlier works on PLS-complete local search problems, where computing the specific local minimum computed by local search is a PSPACE-complete problem (assuming completeness is proved using a "tight" reduction, as in almost all known examples) [12], and the results of Goldberg et al. [11] showing that computing the outcome of various algorithms that solve $PPAD$-complete problems, such as the Lemke-Howson algorithm for computing a Nash equilibrium, are PSPACE-complete problems.

Our contribution is a proof that the `k-means` method, just like the simplex method, inadvertently solves PSPACE-complete problems. That is: computing the outcome of the `k-means` method, given an instance of `k-means` and an initialization for the k centers, is a PSPACE-complete problem.[3] Like with the earlier results on the simplex method, this result provides a new interpretation of the worst-case running time of the `k-means` method – it is exponential not because the work done is inherently wasteful, but rather because it solves a much harder problem than the one it was originally designed for. Our result also implies, under appropriate complexity assumptions, that there is no way of significantly "speeding up" the `k-means` method (in the worst case) without changing its final state.

▶ **Theorem 1.** *Given a `k-means` input $(\mathcal{X}, \mathcal{C})$, it is PSPACE-hard to determine the final cluster centers.*

2 Preliminaries

We briefly review the C-PATH problem, which serves as the starting point for our reduction. The C-PATH problem was used by Adler, Papadimitriou, and Rubinstein to show that determining whether a particular basis occurs on the path of the simplex algorithm, under certain pivoting rules, is PSPACE-complete [1]. The C-PATH problem is as follows: we are given as input a boolean circuit C with fan-in 2 which takes in its own input of n bits, and a target binary string t. For every input x, $C(x)$ is at most Hamming distance one from x, i.e. C computes an index (if any) to flip. Note that this means that C's input and output are the same size. Suppose we begin with the all-zeroes binary string and repeatedly apply C.

[3] Determining the complexity of computing *any* local minimum of the local search problem corresponding to the `k-means` method – not necessarily the local minimum computed by the method on a given initialization – is an intriguing open problem.

The sequence we get, $(0, C(0), C(C(0)), \ldots)$, is the path of C. We want to compute whether this path includes t.

▶ **Lemma 2** ([1]). *There is a family of circuits C of size polynomial in the number of inputs and of polynomial complexity such that C-PATH is PSPACE-complete.*

3 Reduction Sketch

In this section, we sketch the reduction we use to prove Theorem 1. For the sake of clarity and brevity, we omit some technical details here which are addressed in the full construction.

▶ **Theorem 1** (restated). *Given a `k-means` input $(\mathcal{X}, \mathcal{C})$, it is PSPACE-hard to determine the final cluster centers.*

Recall that we reduce from the C-PATH problem, where we have a circuit C and target binary string t. We know that if the path of C ever reaches t, it must do so within 2^n steps. Our plan of attack is to encode circuit C into `k-means`, and then use the reset gadget of Arthur and Vassilvitskii [4] to repeatedly run the encoded circuit on its own output. It is worth noting that although Vattani [18] showed that `k-means` can be made to run for an exponential number of iterations even in the plane, this planar construction is different in a fundamental detail that we depend on. In Arthur and Vassilvitskii's construction, a reset gadget is capable of resetting *all* earlier cluster centers. In Vattani's construciton, a reset gadget resets only the previous cluster center, but does so twice. These gadgets both suffice when the base instance is a single cluster, but only the former can handle a more complex base instance with multiple clusters.

Encoding the Circuit

One benefit of choosing the C-PATH problem is that encoding the circuit is simply a matter of encoding its gates. We use the location of a certain cluster center to represent a boolean value of our gate. When we compute that a gate evaluates to false, its cluster center moves from a starting location to a false region. If it evaluated to true, it would move into a disjoint true region instead.

We can assume without loss of generality that our circuit only uses NAND gates. We go through these gates in topological order; with each new NAND gate we introduce a new dimension to our `k-means` instance. Hence, the inputs to our current gate always lie in a lower-dimensional space. Our NAND gate has two inputs, each with their own false and true region in the space below. Suppose we place an intermediate point roughly $d - \epsilon$ units above each of these regions which are part of cluster i whose center is currently another d units above them, for some large distance $d > 0$ and tiebreaking constant $\epsilon > 0$. When the input cluster centers move to their false or true regions, they steal the respective point above them from cluster i. Depending on which points are stolen, the center of cluster i moves to a predictable location. With additional arranging of the intermediate points, the center moves to either a false region or a true region, the two of which are disjoint.

Repeatedly Running the Circuit

Unfortunately, we cannot immediately apply Arthur and Vassilvitskii's reset gadgets, because they are designed to return a set of cluster centers from specific final locations to initial locations [4]. We care about the final locations of our cluster centers because they represent the output value of our circuit and should affect the input value for the next iteration.

We solve this by, instead of using a single AV reset gadget to reset all cluster centers, using two AV reset gadgets for each input/output bit. One gadget detects when the output bit is false and returns the cluster center to its initial position while also setting the input bit to false. The other gadget does the same but for true bits. Furthermore, we can still layer these gadgets; gadgets reset all corresponding gadgets in previous layers. With only n layers, we can run the circuit 2^n times, enough to guarantee that t will appear if it is indeed in the path of the circuit.

As a final step, we add a gadget to track whether t has appeared. To do this, suppose we modify the circuit so that a special bit is 1 if and only if the input was t. We can use the same idea as with NAND gates; we add an intermediate point roughly $d - \epsilon$ units above the true region of this special bit, which are part of a cluster whose center is an additional d units above the intermediate point. The only other point in this cluster, in fact, is d units above the center. If the special bit ever becomes 1, the intermediate point will be stolen and the cluster center will move to the top point. After this, the center is $2d$ from any other point and can neither gain nor lose data points. All we need to check in the `k-means` output is the location of this center to know whether the path of C includes t.

4 Formal Reduction Proof

In this section, we formally prove Theorem 1. We follow the sketch given in Section 3. Missing tables can be found in Appendix A.

Encoding the Circuit

Recall that we begin with an instance of the C-PATH problem, (C, t) where we want to know if t is on the path of C. It will be convenient to convert C so that it only has NAND gates instead of the standard AND, OR, and NOT gates. We also require that each gate has a fan-out of at most two, but introduce a special SPLIT gate which takes in a single bit and outputs it back. This can be implemented with only a constant blowup in the number of gates, since fan-in larger than two can be simulated with a binary tree of SPLIT gates (and the number of nodes is at most twice the number of leaves). We require that the inputs of a NAND gate can only SPLIT gates, which can be guaranteed by inserting a SPLIT gate of fan-out one before each NAND gate. This at most doubles the number of gates. We also require that the inputs of a NAND gate must be at the same depth and that all outputs are at the same depth. One slightly inefficient, yet still polynomial method to guarantee this is to take every NAND gate and place it in its own layer. Inside a layer, there is only a single NAND gate, but we use SPLIT gates to pass on the other values. We add SPLIT gates after outputs which occur too early. This synchronizes the circuit with only a quadratic blowup in the number of gates.

We represent boolean values in our circuit with the location of a cluster center. Each cluster center serves to signal the output of a gate to only one other gate (why we bound the fan-out). Gate i uses cluster centers c_{2i-1} and c_{2i}. Cluster center c_j has an initial location $\mathbf{s}_j$, a false region centered at $\mathbf{b}_{j,0}$ with radius r_j, a true region centered at $\mathbf{b}_{j,1}$ with the same radius r_j, and a final location $\mathbf{t}_j$. At some timestep, the center will move from its initial location to either its false region or its true region, which serves as a signal to the gate that takes it as input. It then eventually moves to its final location. We guarantee that no two initial locations, false regions, true regions, or final locations overlap, even over different clusters. This property remains true when moving the final location towards the average of

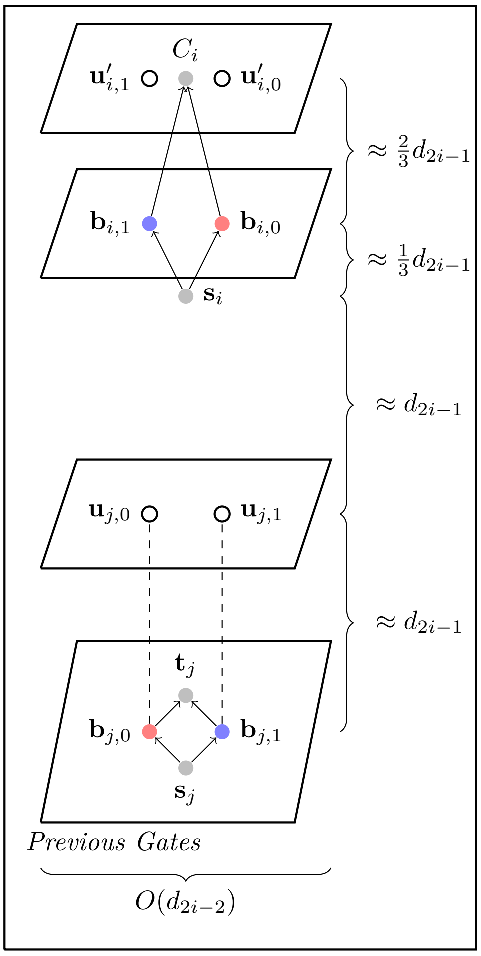


Figure 1 SPLIT Gadget, Upper Half.

the false and true region centers. Finally, we guarantee a cluster center is always the closest center to any of its locations or regions.

We construct gates in topological order. For each gate i, we introduce two new dimensions: $(2i-1)$ and $(2i)$. We let $\mathbf{e}_j$ be the standard basis vector for dimension j. We also grow the scale of our construction at each step; for each dimension, we choose a scaling factor $d_i > 0$ so that $d_1 \ll d_2 \ll \cdots \ll d_{2m}$. The data points we introduce with dimension i are within $O(d_i)$ of the origin. The idea is that d_{i+1} is large enough compared to d_i so that two points which differ by d_{i+1} in their $(i+1)^{th}$ coordinate and by $O(d_i)$ in their first i coordinates are still roughly d_{i+1} apart. We also use a small $\epsilon > 0$ to break ties (note $\epsilon \ll d_1$). We also use $d(\mathbf{u}, \mathbf{v})$ to represent the Euclidean distance between points $\mathbf{u}$ and $\mathbf{v}$.

SPLIT Gadget

We first explain the construction of the simpler SPLIT gadget. Suppose we have the i^{th} gate which takes the j^{th} cluster center as input. The data points and regions we use in this construction are listed in Table 1 and the upper half of the gadget is depicted in Figure 1.

Suppose that at time T, the j^{th} cluster center moves to its false or true region. We want to notice when this occurs, so we place an intermediate point $\mathbf{u}_{j,0}$ roughly $(d_{2i-1} - \epsilon)$ above the false region, an intermediate point $\mathbf{u}_{j,1}$ roughly $(d_{2i-1} - \epsilon)$ above the true region, an intermediate point $\mathbf{v}_{j,0}$ roughly $(d_{2i-1} - \epsilon)$ below the false region, and an intermediate point $\mathbf{v}_{j,1}$ roughly $(d_{2i-1} - \epsilon)$ below the true region. Note that the actual heights are actually scaled to account for the radius of each of these regions; *every* point in the region at most $(d_{2i-1} - \epsilon)$ away from its two intermediate points.

We want the $(2i - 1)^{th}$ cluster center to be d units away from the top two intermediate points and the $(2i)^{th}$ cluster center to be d units away from the bottom two intermediate points. For each intermediate point u, we add a counterbalancing point u' so that the average of the two is our desired initial center location.

At time $T + 1$, the j^{th} cluster steals either $\mathbf{u}_{j,0}$ and $\mathbf{v}_{j,0}$ or $\mathbf{u}_{j,1}$ and $\mathbf{v}_{j,1}$, depending on whether it was false or true. This causes the $(2i - 1)^{th}/(2i)^{th}$ cluster centers to move up/down to their respective false or true regions (which are actually balls of radius zero). This does not affect the $(2i - 1)$-coordinate of the j^{th} cluster since the two points it stole cancel out. However, it does move the center towards the center of the region it was in.

At time $T + 2$, because the $(2i - 1)^{th}$ and $(2i)^{th}$ cluster centers moved away from the lower-dimensional space, the other intermediate points are stolen by cluster j (recall we guarantee that cluster center j is the closest center to any of its regions, and in particular to the false or true region it was not in). This causes the $(2i - 1)^{th}/(2i)^{th}$ cluster centers to move up/down to their final locations. Again, this does not affect the $(2i - 1)$-coordinate of the j^{th} cluster because the points cancel out. However, it does move the center towards the center of the region it was not in. Notice we have affected the final location of the j^{th} cluster center, but we already assumed that moving it towards the average of the false and true regions would keep all locations and regions disjoint.

We see that we satisfy the assumptions made about the construction; the locations and regions we create are disjoint from all others because the other locations and regions are within $O(d_{2i-2})$ of the origin and all of ours are at least $\Omega(d_{2i-1})$ from the origin. This also makes our cluster center the closest to all of our locations and regions (since other centers cannot escape the lower-dimensional space due to our balancing). Finally, moving our final location towards the average of the false and true regions keeps it disjoint.

NAND Gadget

We now proceed to the constrution of the NAND gadget. Suppose we have the i^{th} gate which takes the j^{th} and k^{th} cluster centers as input. For simplicity, we concern ourselves with the data points and cluster center regions for cluster $(2i - 1)$ only. The $(2i)^{th}$ cluster's points and regions can be found by negating the $(2i - 1)^{th}$ and $(2i)^{th}$ coordinates. The points and regions we do present are listed in Table 2 and depicted in Figure 2. For this gadget, we talk about dimension $2i - 1$ as left/right and dimension $2i$ as up/down.

We have the same core plan as the SPLIT gadget. We add intermediate points above the two false regions and above the two true regions. We also slide these intermediate points left and right slightly; we shift intermediate points above false regions to the right and intermediate points above true regions to the left. Because $d_{2i-1} \ll d_{2i}$, these points are still roughly $d_{2i} - \epsilon$ from their respective regions.

The initial center of cluster $(2i - 1)$ is d_{2i} above all four intermediate points; we achieve this by adding counterbalancing points so that the average of a point and its counterbalancing partner is our desired center.

Suppose at time T both cluster centers j and k move to false or true regions. Then at time $T + 1$, they each steal an appropriate intermediate point above them. This causes the center of cluster $(2i - 1)$ to definitely upwards, and possibly left or right depending on what

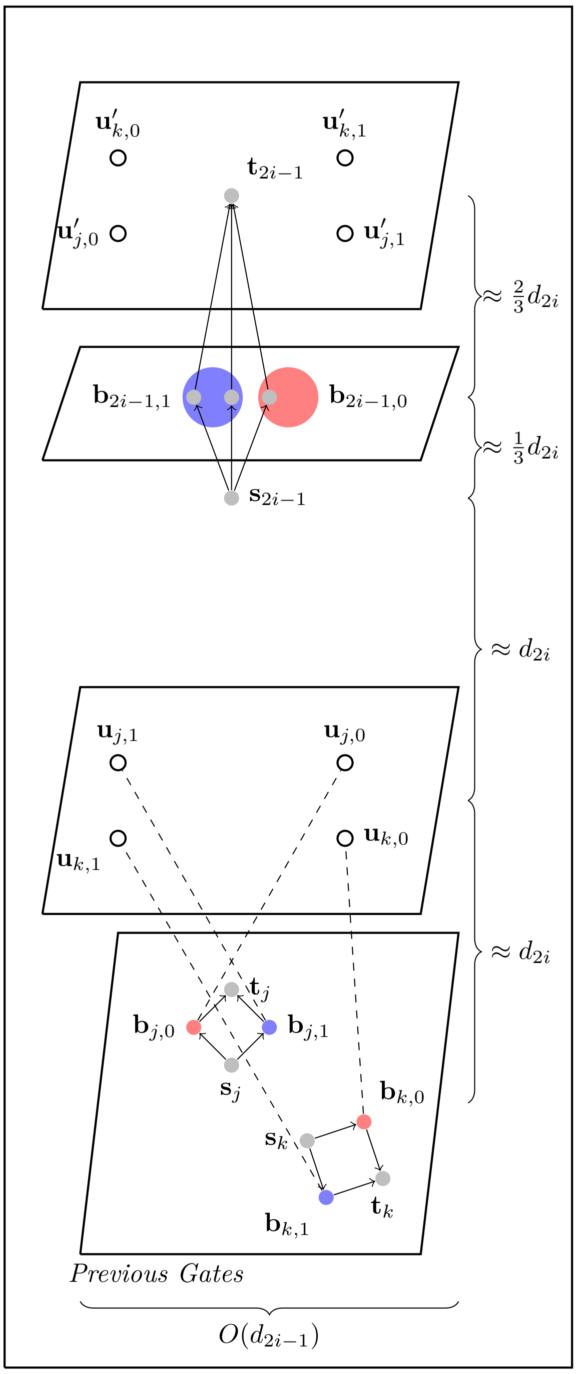


Figure 2 NAND Gadget, Upper Half.

values j and k had. If both j and k are false, they steal intermediate points on the right and our center moves to the left. If they are both true, our center moves to the right. If we have one of each, the center does not move left or right. We can place our false region to capture the right possibility and our true region to capture both the center and left possibilities.

At time $T+2$, cluster centers j and k steal the intermediate point for the region they did not enter, because cluster center $(2i-1)$ moved up and they are the closest center to any of

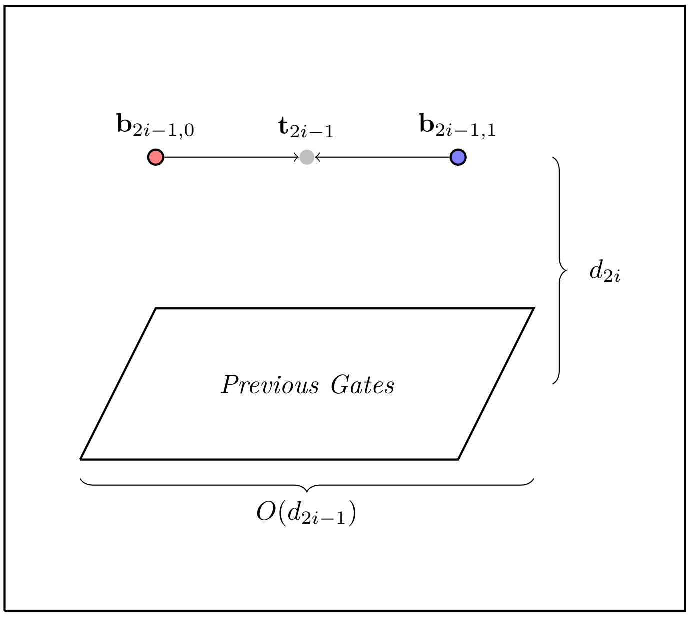


Figure 3 Input Gadget.

their respective regions. This moves the cluster center $(2i - 1)$ to its final location. Note that we have shifted the final location of cluster centers j and k, but this keeps locations and regions disjoint by assumption.

We satisfy all assumptions made about the construction for essentially the same reasons as before. The one difference is regarding moving our final location towards the average of the false and true regions. It remains disjoint because $d_{2i-1} \ll d_{2i}$, so it descends approximately straight from above, avoiding the false and true regions.

Input Gadget

We also provide a simple gadget which is used to signal the value of an input bit. We only use the $(2i - 1)^{th}$ cluster center and do not actually provide an initial location. Instead, the center is intended to start in either the false or true region, which signals its value. It then immediately moves to its final location. Dimension $(2i - 1)$ is used to separate the false and true regions while dimension $(2i)$ is used to separate the gadget from previous gadgets. The points and regions are listed in Table 3 and depicted in Figure 3. Note that unlike previous gadgets, there is no corresponding bottom half to this gadget, since we do not need to balance its effect on previous gadgets.

AV Reset Gadget Review

Before we describe how we repeatedly run the circuit, we briefly review the reset gadgets of Arthur and Vassilvitskii [4]. A configuration is signaling if at least one final cluster center is distinct from every cluster center arising in previous iterations. A configuration is super-signaling if all final cluster centers are distinct from every cluster center arising in previous iterations and there is an alternate initial configuration of centers that is essentially identical except at least one final cluster center is different.

AV gadgets are a result of two constructions (Lemma 3.3 and Lemma 3.4 in their paper). One construction (Lemma 3.3) converts a super-signaling configuration into a signaling configuration which runs from the initial configuration and then swaps to the alternate intial configuration. The other (Lemma 3.4) converts a signaling configuration into a super-signaling configuration. They share similar ideas to our gadget constructions above; intermediate points are placed above and below the expected locations. Similar to our NAND gadget shifting the intermediate points left/right, they shift the intermediate points in a *circle* using two additional dimensions. They then use a second set of intermediate points which are stolen according to how points were shifted in a circle. This enables them to correct every center to its new initial location. There are also two additional clusters which represent an alternate initial configuration.

Taken together, these two constructions take a lower-dimensional signaling configuration and make it run twice by resetting its final cluster positions to their initial positions. Layering n gadgets results in 2^n resets of the bottom-level circuit.

We make two key observations about the capabilities of AV reset gadgets. First, they need not reset the positions of all lower-dimensional clusters; we be selective and only reset some. Second, the signaling cluster may reach its distinct location one step before it reaches its final position. This works because intermediate points may be placed above and below its distinct location, and the signaling cluster will steal them and reach its final position in the same time step. The location of the intermediate point which corrects its final location to its initial location still uses its real final location. These observations enable us to use AV reset gadgets to set the input of the circuit to its previous output, despite our particular method of signalling boolean values.

Repeatedly Running the Circuit

We are now ready to explain how to repeatedly run our encoded circuit. We need to run it up to 2^n times to guarantee we reach t, if at all possible. We plan to do this with AV gadgets. Unfortunately, AV reset gadgets work by knowing the exact final locations of cluster centers and moving them to exact initial locations. We want to copy circuit output to input. Not only do we not have exact final locations, but we also want them to influence the new initial locations. This is solved by using one AV reset gadget chain per input bit and each boolean value it can take on. An AV reset gadget for bit i being false is signaled by the cluster center of output bit i entering its false region. It corrects the final location of the cluster center to the false region of input bit i and it corrects the final location of input bit i to the initial location of output bit i.

We want to run the circuit 2^n times, so a first attempt is to reset each (input bit, value) pair that many times. Unfortunately, this has unintended behavior if the path of C has unbalanced parity. For example, suppose we had a circuit C where $C(00) = 01$ and $C(01) = 00$. In four iterations, we follow the path $(00, 01, 00, 01, 00)$. The false AV reset gadget for the first input bit is now fully expended and stops resetting, but the AV reset gadgets for the second input bit still reset it, causing only part of the circuit input to be copied from the output. In a more complex example, this could evaluate the circuit on an input not actually in the path of C.

To avoid this, we transform C so that it follows a balanced path, i.e. in every bit it alternates between true and false. Circuit C_2 has an additional parity bit in its input. When this auxiliary bit is zero, it simply flips the entire input. When the auxiliary bit is one, it again flips the entire input, and then applies C_2 to the standard input. This can be be implemented with only polynomially many extra gates. More formally:

$$C_2(\mathbf{x}b_1) = \begin{cases} \bar{\mathbf{x}}1 & \text{if } b_1 = 0 \\ C(\bar{\mathbf{x}})0 & \text{if } b_1 = 1 \end{cases}$$

We now have $n+1$ input bits and plan to reset each 2^n times, for each value. It will also be convenient to keep track of whether we have reached t yet, so that we can simply examine the final `k-means` state. We add another auxiliary bit, which transitions from zero to one when the input is t. This also can be implemented with only polynomially many extra gates. Formally:

$$C_3(\mathbf{x}b_1b_2) = \begin{cases} C_2(\mathbf{x}b_1)0 & \text{if } b_2 = 0, \mathbf{x}b_1 \neq t0 \\ C_2(\mathbf{x}b_1)1 & \text{if } b_2 = 0, \mathbf{x}b_1 = t0 \\ C_2(\mathbf{x}b_1)1 & \text{if } b_2 = 1 \end{cases}$$

Note that for our second auxiliary bit, we want to reset a false value 2^{n+1} times and we do not worry about resetting a true value. Also, we can conveniently use the AV reset widget of this bit to reset the inner gates of the circuit. This completes our circuit reset construction.

Output Gadget

Our final gadget records whether $b_2 = 1$ at any point. Suppose that b_2 is represented by the position of cluster center i. We assume without loss of generality that it is computed by a SPLIT gate. We use one final additional dimension, with the largest scale. Suppose this is dimension D. We add points at $(\mathbf{b}_{i,1} + (d_D - \epsilon)\mathbf{e}_D)$ and $(\mathbf{b}_{i,1} + (3d_D - \epsilon)\mathbf{e}_D)$. We also add a final cluster center at $(\mathbf{b}_{i,1} + (2d_D - \epsilon)\mathbf{e}_D)$. The former point can only be stolen from this cluster if $b_2 = 1$, and when this happens the cluster center will move to the latter point. But the latter point is $2d_D$ from the former point, so the cluster center can never recapture it.

As review, the completed construction uses gadgets in the following order (from low-dimensional to high-dimensional):

1. $n+2$ input gadgets which represent the input bits,
2. $poly(n)$ NAND and SPLIT gadgets which represent gates and output bits,
3. $(n+1)(2n) + (n+1)$ reset gadgets to repeatedly run the circuit,
4. and one output gadget which represents the final result.

We have produced a polynomially-sized `k-means` instance from a C-PATH instance. The final state of our output gadget indicates the answer to the C-PATH instance, so computing the final state of `k-means` is enough to solve a PSPACE-complete problem. This completes the proof.

5 Conclusions

This paper proved that the `k-means` method inadvertently solves PSPACE-complete problems, echoing analogous results for the simplex method [1, 9, 10]. There are at least three interesting directions in which our result might be extended.

1. We conjecture that the following problem is PLS-complete: given an instance of `k-means`, compute an arbitrary local minimum of the `k-means` method. Such a result, if proved

using “tight” reductions[4] (see [12]), would imply our Theorem 1.[5]

2. The worst-case running time of the `k-means` method is exponential even in two dimensions [18], while our PSPACE-completeness reduction produces instances with a large number of dimensions. Is computing the outcome of `k-means` still PSPACE-complete in planar instances? Recall that Vattani’s reset gadgets work by resetting only the previous gadget, but doing so twice. Our reduction depended on the ability of AV reset gadgets to reset all previous clusters so that the entire circuit could be reset.
3. Does the problem of computing the outcome of `k-means` remain PSPACE-complete when the initial centers are chosen greedily, as in `k-means++` [5]?

References

1 Ilan Adler, Christos Papadimitriou, and Aviad Rubinstein. On simplex pivoting rules and complexity theory. In *Integer Programming and Combinatorial Optimization*, pages 13–24. Springer, 2014.

2 Nina Amenta and Gunter M Ziegler. Deformed products and maximal shadows of polytopes. *Contemporary Mathematics*, 223:57–90, 1999.

3 David Arthur, Bodo Manthey, and H Roglin. k-means has polynomial smoothed complexity. In *Foundations of Computer Science, 2009. FOCS’09. 50th Annual IEEE Symposium on*, pages 405–414. IEEE, 2009.

4 David Arthur and Sergei Vassilvitskii. How slow is the k-means method? In *Proceedings of the twenty-second annual symposium on Computational geometry*, pages 144–153. ACM, 2006.

5 David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

6 Pavel Berkhin. A survey of clustering data mining techniques. In *Grouping multidimensional data*, pages 25–71. Springer, 2006.

7 Anup Bhattacharya, Ragesh Jaiswal, and Nir Ailon. A tight lower bound instance for k-means++ in constant dimension. In *Theory and Applications of Models of Computation*, pages 7–22. Springer, 2014.

8 Tobias Brunsch and Heiko Röglin. A bad instance for k-means++. In *Theory and Applications of Models of Computation*, pages 344–352. Springer, 2011.

9 Yann Disser and Martin Skutella. The simplex algorithm is np-mighty. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 858–872. SIAM, 2015.

10 John Fearnley and Rahul Savani. The complexity of the simplex method. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, pages 201–208. ACM, 2015.

11 Paul W Goldberg, Christos H Papadimitriou, and Rahul Savani. The complexity of the homotopy method, equilibrium selection, and lemke-howson solutions. *ACM Transactions on Economics and Computation*, 1(2):9, 2013.

[4] In a tight PLS reduction, there is also a correspondence between improving moves in the two instances, not just between solutions.

[5] For this question to be interesting, it is important to rule out degenerate local minima. A partial solution is to insist that each of the k initial centers lies in the convex hull of the point set. (Otherwise, placing one center at the center of mass of the entire point set and the other $k-1$ centers “at infinity” is an easy-to-compute local minimum.) Similarly, dealing with (or avoiding) co-located centers requires some care.

12 David S Johnson, Christos H Papadimitriou, and Mihalis Yannakakis. How easy is local search? *Journal of computer and system sciences*, 37(1):79–100, 1988.

13 Tapas Kanungo, David M Mount, Nathan S Netanyahu, Christine D Piatko, Ruth Silverman, and Angela Y Wu. A local search approximation algorithm for k-means clustering. In *Proceedings of the eighteenth annual symposium on Computational geometry*, pages 10–18. ACM, 2002.

14 Victor Klee and George J Minty. How good is the simplex algorithm. Technical report, DTIC Document, 1970.

15 Stuart P Lloyd. Least squares quantization in pcm. *Information Theory, IEEE Transactions on*, 28(2):129–137, 1982.

16 Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan. The planar k-means problem is np-hard. In *WALCOM: Algorithms and Computation*, pages 274–285. Springer, 2009.

17 Daniel A Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM (JACM)*, 51(3):385–463, 2004.

18 Andrea Vattani. K-means requires exponentially many iterations even in the plane. *Discrete & Computational Geometry*, 45(4):596–616, 2011.

19 Norman Zadeh. A bad network problem for the simplex method and other minimum cost flow algorithms. *Mathematical Programming*, 5(1):255–266, 1973.

A Missing Reduction Tables

Table 1 SPLIT Gadget Points and Regions.

Data Point	**Location**	**Purpose**
$\mathbf{u}_{j,0}$	$\mathbf{b}_{j,0} + \left(\sqrt{(d_{2i-1}-\epsilon)^2 - r_j^2}\right)\mathbf{e}_{2i-1}$	Detect j is False
$\mathbf{u}_{j,1}$	$\mathbf{b}_{j,1} + \left(\sqrt{(d_{2i-1}-\epsilon)^2 - r_j^2}\right)\mathbf{e}_{2i-1}$	Detect j is True
$\mathbf{u}'_{j,0}$	$\mathbf{u}_{j,1} + \left(\sqrt{4d_{2i-1}^2 - d(\mathbf{u}_{j,0},\mathbf{u}_{j,1})}\right)\mathbf{e}_{2i-1}$	Counterbalance $\mathbf{u}_{j,0}$
$\mathbf{u}'_{j,1}$	$\mathbf{u}_{j,0} + \left(\sqrt{4d_{2i-1}^2 - d(\mathbf{u}_{j,0},\mathbf{u}_{j,1})}\right)\mathbf{e}_{2i-1}$	Counterbalance $\mathbf{u}_{j,1}$
$\mathbf{v}_{j,0}$	$\mathbf{b}_{j,0} - \left(\sqrt{(d_{2i-1}-\epsilon)^2 - r_j^2}\right)\mathbf{e}_{2i-1}$	Detect j is False
$\mathbf{v}_{j,1}$	$\mathbf{b}_{j,1} - \left(\sqrt{(d_{2i-1}-\epsilon)^2 - r_j^2}\right)\mathbf{e}_{2i-1}$	Detect j is True
$\mathbf{v}'_{j,0}$	$\mathbf{u}_{j,1} - \left(\sqrt{4d_{2i-1}^2 - d(\mathbf{u}_{j,0},\mathbf{u}_{j,1})}\right)\mathbf{e}_{2i-1}$	Counterbalance $\mathbf{v}_{j,0}$
$\mathbf{v}'_{j,1}$	$\mathbf{u}_{j,0} - \left(\sqrt{4d_{2i-1}^2 - d(\mathbf{u}_{j,0},\mathbf{u}_{j,1})}\right)\mathbf{e}_{2i-1}$	Counterbalance $\mathbf{v}_{j,1}$
Cluster Center Region	**Center**	**Radius**
$\mathbf{s}_{2i-1}$	$\frac{1}{4}\left(\mathbf{u}_{2i-1,0} + \mathbf{u}_{2i-1,1} + \mathbf{u}'_{j,0} + \mathbf{u}'_{j,1}\right)$	0
$\mathbf{b}_{2i-1,0}$	$\frac{1}{3}\left(\mathbf{u}_{2i-1,1} + \mathbf{u}'_{j,0} + \mathbf{u}'_{j,1}\right)$	0
$\mathbf{b}_{2i-1,1}$	$\frac{1}{3}\left(\mathbf{u}_{2i-1,0} + \mathbf{u}'_{j,0} + \mathbf{u}'_{j,1}\right)$	0
$\mathbf{t}_{2i-1}$	$\frac{1}{2}\left(\mathbf{u}'_{j,0} + \mathbf{u}'_{j,1}\right)$	0
$\mathbf{s}_{2i}$	$\frac{1}{4}\left(\mathbf{v}_{2i-1,0} + \mathbf{v}_{2i-1,1} + \mathbf{v}'_{j,0} + \mathbf{v}'_{j,1}\right)$	0
$\mathbf{b}_{2i,0}$	$\frac{1}{3}\left(\mathbf{v}_{2i-1,1} + \mathbf{v}'_{j,0} + \mathbf{v}'_{j,1}\right)$	0
$\mathbf{b}_{2i,1}$	$\frac{1}{3}\left(\mathbf{v}_{2i-1,0} + \mathbf{v}'_{j,0} + \mathbf{v}'_{j,1}\right)$	0
$\mathbf{t}_{2i}$	$\frac{1}{2}\left(\mathbf{v}'_{j,0} + \mathbf{v}'_{j,1}\right)$	0

Table 2 NAND Gadget Points and Regions, Upper Half.

Data Point	**Location**	**Purpose**
$\mathbf{u}_{j,0}$	$\mathbf{b}_{j,0} + d_{2i-1}\mathbf{e}_{2i-1} + (d_{2i} - \epsilon)\,\mathbf{e}_{2i}$	Detect j is False
$\mathbf{u}_{j,1}$	$\mathbf{b}_{j,1} - d_{2i-1}\mathbf{e}_{2i-1} + (d_{2i} - \epsilon)\,\mathbf{e}_{2i}$	Detect j is True
$\mathbf{u}_{k,0}$	$\mathbf{b}_{k,0} + d_{2i-1}\mathbf{e}_{2i-1} + (d_{2i} - \epsilon)\,\mathbf{e}_{2i}$	Detect k is False
$\mathbf{u}_{k,1}$	$\mathbf{b}_{k,1} - d_{2i-1}\mathbf{e}_{2i-1} + (d_{2i} - \epsilon)\,\mathbf{e}_{2i}$	Detect k is True
$\mathbf{u}'_{j,0}$	$-\mathbf{b}_{j,0} - d_{2i-1}\mathbf{e}_{2i-1} + (3d_{2i} - \epsilon)\,\mathbf{e}_{2i}$	Counterbalance $\mathbf{u}_{j,0}$
$\mathbf{u}'_{j,1}$	$-\mathbf{b}_{j,1} + d_{2i-1}\mathbf{e}_{2i-1} + (3d_{2i} - \epsilon)\,\mathbf{e}_{2i}$	Counterbalance $\mathbf{u}_{j,1}$
$\mathbf{u}'_{k,0}$	$-\mathbf{b}_{k,0} - d_{2i-1}\mathbf{e}_{2i-1} + (3d_{2i} - \epsilon)\,\mathbf{e}_{2i}$	Counterbalance $\mathbf{u}_{k,0}$
$\mathbf{u}'_{k,1}$	$-\mathbf{b}_{k,1} + d_{2i-1}\mathbf{e}_{2i-1} + (3d_{2i} - \epsilon)\,\mathbf{e}_{2i}$	Counterbalance $\mathbf{u}_{k,1}$
Cluster Center Region	**Center**	**Radius**
$\mathbf{s}_{2i-1}$	$(2d_{2i} - \epsilon)\mathbf{e}_{2i}$	0
$\mathbf{b}_{2i-1,0}$	$-\frac{1}{6}(\mathbf{b}_{j,1} + \mathbf{b}_{k,1}) + \frac{1}{2}d_{2i-1}\mathbf{e}_{2i-1} + (\frac{7}{3}d_{2i} - \epsilon)\mathbf{e}_{2i}$	$\frac{1}{5}d_{2i-1}$
$\mathbf{b}_{2i-1,1}$	$-\frac{1}{6}(\mathbf{b}_{j,0} + \mathbf{b}_{k,0}) - \frac{1}{6}d_{2i-1}\mathbf{e}_{2i-1} + (\frac{7}{3}d_{2i} - \epsilon)\mathbf{e}_{2i}$	$\frac{1}{5}d_{2i-1}$
$\mathbf{t}_{2i-1}$	$-\frac{1}{4}(\mathbf{b}_{j,0} + \mathbf{b}_{j,1} + \mathbf{b}_{k,0} + \mathbf{b}_{k,1}) + (3d_{2i} - \epsilon)\mathbf{e}_{2i}$	0

Table 3 Input Gadget Points and Regions.

Data Point	**Location**	**Purpose**
$\mathbf{v}_{2i-1,0}$	$-d_{2i-1}\mathbf{e}_{2i-1} + d_{2i}\mathbf{e}_{2i}$	False Point
$\mathbf{v}_{2i-1,1}$	$d_{2i-1}\mathbf{e}_{2i-1} + d_{2i}\mathbf{e}_{2i}$	True Point
Cluster Center Region	**Center**	**Radius**
$\mathbf{b}_{2i-1,0}$	$-d_{2i-1}\mathbf{e}_{2i-1} + d_{2i}\mathbf{e}_{2i}$	0
$\mathbf{b}_{2i-1,1}$	$d_{2i-1}\mathbf{e}_{2i-1} + d_{2i}\mathbf{e}_{2i}$	0
$\mathbf{t}_{2i-1}$	$d_{2i}\mathbf{e}_{2i}$	0